Licence Sciences et Techniques

L1 MATHS - PIM

Initiation à la programmation informatique avec

Recueil d'exercices corrigés et aide-mémoire.

Gloria Faccanoni

https://moodle.univ-tln.fr/course/view.php?id=4968

http://faccanoni.univ-tln.fr/enseignements.html

Année 2023 – 2024

Dernière mise-à-jour
Mardi 26 septembre 2023

Programme Indicatif			
Semaine	CM	TD	TP
39	CM-1 : Moodle, Intro., Ch. 1		
40		TD-1 : ✎ 1.4 - 1.5 - 1.6 - 1.7 - 1.8 - 1.9 - 1.17 - 1.18	
41	CM-2 : Ch. 2-3		
42	CM-3 : Ch. 4-5-6	TD-2 : ✎ 2.1 - 2.2 ✎ 3.1 - 3.2 ✎ 4.1 - 4.12 - 4.13 - 4.14	
43-44		Pause	
45		TD-3 : ✎ 5.1 - 5.2 - 5.5 - 5.11 - 5.16 - 5.29 ✎ 6.1 - 6.3 - 6.4 - 6.8 - 6.9 - 6.32 - 6.33 - 6.34 - 6.36	TP-1 : 🖵 1.1 - 1.2 - 1.12 - 1.13 - 1.14 - 1.15 - 1.19 - 1.20 - 1.21 - 1.22 🖵 2.3 - 2.4 - 2.5 - 2.6 - 2.7 - 2.8
46	CM-4 : Ch. 7-8		TP-2 : 🖵 2.10 - 2.12 🖵 3.3 - 3.4 - 3.5 - 3.6 - 3.7 - 3.8
47		TD-4 : ✎ 7.1 - 7.3 - 7.13 - 7.21 ✎ 8.2 - 8.10 - 8.11 - 8.19 ✎ annales	TP-3 : 🖵 4.3 - 4.5 - 4.7 - 4.9 - 4.10 - 4.11 - 4.15 - 4.16 - 4.17 - 4.22 - 4.23 - 4.25 - 4.27 - 4.29 - 4.30 - 4.57
48			TP-4 : 🖵 5.3 - 5.4 - 5.6 - 5.7 - 5.8 - 5.12 - 5.17 - 5.18 - 5.20 - 5.24 - 5.28 - 5.29 - 5.35
49			TP-5 : 🖵 6.5 - 6.7 - 6.10 - 6.11 - 6.12 - 6.15 - 6.17 - 6.19 - 6.20 - 6.22 - 6.25 - 6.27 - 6.28 - 6.38 - 6.42 - 6.50 - 6.60 🖵 Test Moodle d'entraînement
50			TP-6 : 🖵 7.5 - 7.6 - 7.7 - 7.9 - 7.14 - 7.18 - 7.20 - 7.25 - 7.26 - 7.33 - 7.34 - 7.38 🖵 Test Moodle d'entraînement
51			TP-7 : 🖵 8.1 - 8.3 - 8.9 - 8.12 - 8.14 - 8.16 - 8.17 - 8.18 - 8.22 🖵 Test Moodle d'entraînement

CC (en salle de TP), semaine 51
CT (en salle de TP), semaine 2

Gloria FACCANONI

IMATH Bâtiment M-117
Université de Toulon
Avenue de l'université
83957 LA GARDE - FRANCE

☎ 0033 (0)4 83 16 66 72

✉ gloria.faccanoni@univ-tln.fr
ℹ http://faccanoni.univ-tln.fr

Table des matières

Introduction **5**

1. Notions de base de Python **11**
1.1. Mode interactif et mode script 11
1.2. Commentaires 14
1.3. Indentation 14
1.4. Types primitifs 15
1.5. Variables et affectation 15
1.6. Nombres 18
1.7. Opérations arithmétiques 19
1.8. Type booléen, Opérateurs de comparaison et connecteurs logiques 21
1.9. Les chaînes de caractères (String) et la fonction `print` 22
1.10. ☆ La fonction `input` 29
1.11. Exercices 31

2. Structures de référence : Listes, Tuples, Dictionnaires et Ensembles **45**
2.1. Listes 45
2.2. Les tuples 50
2.3. L'itérateur `range` 51
2.4. ☆ Les dictionnaires (ou tableaux associatifs) 52
2.5. ☆ Les ensembles 54
2.6. Exercices 59

3. Structure conditionnelle **69**
3.1. Définir des conditions avec les instructions `if/else` 69
3.2. Exercices 73

4. Structures itératives **81**
4.1. Répétition `for` : boucle inconditionnelle (parcourir) 81
4.2. Boucle `while` : répétition conditionnelle 82
4.3. ☆ Ruptures de séquences 83
4.4. Exercices 87

5. Définitions en compréhension **123**
5.1. Listes en compréhension 123
5.2. ☆ Dictionnaires en compréhension 125
5.3. ☆ Ensembles en compréhension 125
5.4. Exercices 127

6. Fonctions **149**
6.1. Fonctions prédéfinies 149
6.2. Définition d'une fonction 150
6.3. Fonctions Lambda (fonctions anonymes) 154
6.4. ☆ Fonctions récursives 156
6.5. Exercices 159

7. Modules **215**
7.1. Importation des fonctions d'un module 215
7.2. Quelques modules 216
7.3. Exercices 225

8. Tracé de courbes **265**
8.1. Importation des modules `matplotlib` et `numpy` 265
8.2. Tracé d'une courbe sur un repère 266
8.3. Plusieurs courbes sur le même repère et options 267

8.4. Plusieurs “fenêtres” graphiques . . . 269
8.5. Une grille de repères dans la même fenêtre . . . 270
8.6. ☆ Animations . . . 271
8.7. ☆ Régression : polynôme de meilleure approximation . . . 272
8.8. Exercices . . . 275

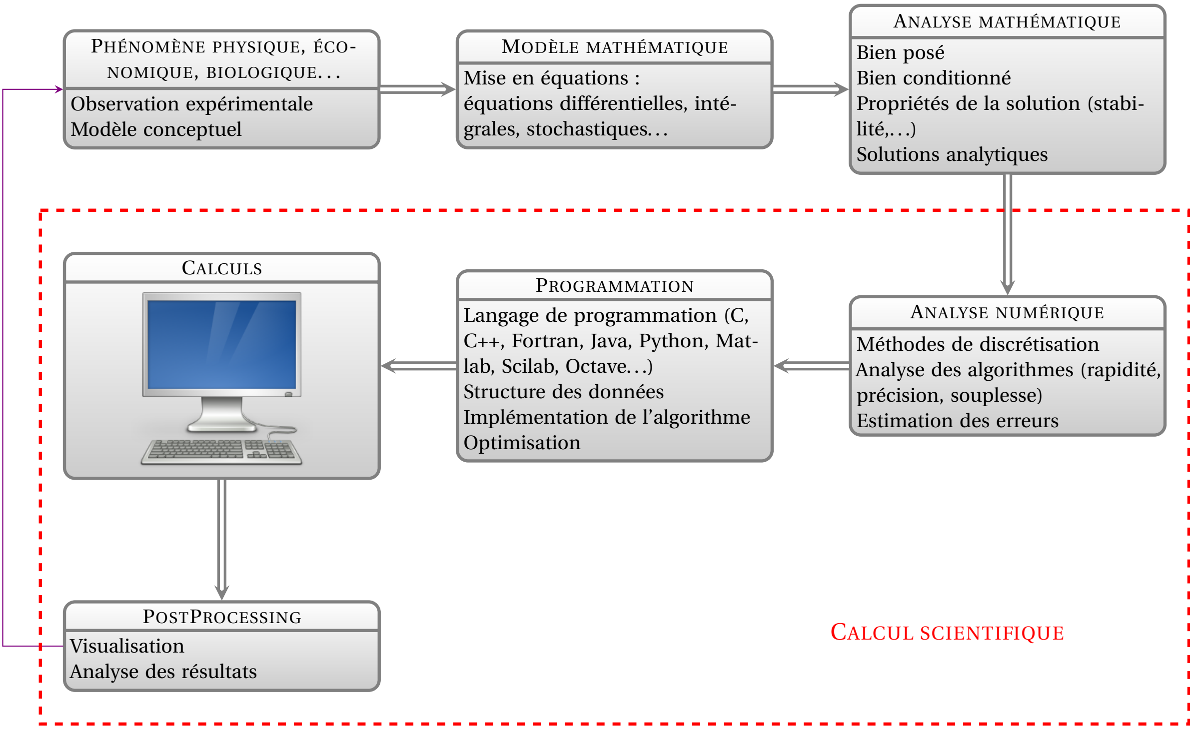
A. Les «mauvaises» propriétés des nombres flottants et la notion de précision 301
A.1. Ne jamais faire confiance aveuglément aux résultats d’un calcul obtenu avec un ordinateur… . . . 302
A.2. Exercices . . . 307

Introduction

Les **mathématiques appliquées** sont une discipline qui se situe à l'intersection de nombreuses domains scientifiques, telles que l'informatique, la physique, la chimie, la mécanique, la biologie, l'économie, les sciences de l'ingénieur... Ces dernières couvrent un large éventail d'applications industrielles telles que l'aéronautique, la production d'énergie, la finance, l'imagerie, et bien d'autres encore.

Introduction au calcul scientifique

Le CALCUL SCIENTIFIQUE peut être défini comme la pratique consistant à utiliser un ordinateur pour simuler un phénomène ou un processus décrit par un modèle mathématique.



Le choix et l'optimisation des algorithmes numériques sont essentiels en calcul scientifique. Ils permettent de réaliser à la fois des calculs industriels très rapides et des calculs de référence qui peuvent nécessiter beaucoup de temps. En utilisant une approche numérique standard par rapport à une approche soigneusement réfléchie et optimisée, il est souvent possible d'obtenir un gain de temps de calcul d'un facteur de 100, voire plus. Ainsi, il est crucial pour tout scientifique de bien connaître ces méthodes, leurs avantages et leurs limites, car cela peut faire la différence entre un calcul déraisonnable et un calcul banal.

Un scientifique de formation, que ce soit dans un environnement recherche ou industriel, peut difficilement ignorer ce domaine dont l'importance est de plus en plus massive, même s'il n'est pas directement impliqué dans les calculs. Cela est dû à l'importance croissante de ce domaine, qui nécessite une double compétence dans la discipline d'origine et en simulation pour une utilisation pertinente et efficace de la puissance de calcul disponible. Les cours de Calcul scientifique dispensés dans les cursus de Licence Mathématiques visent à fournir les bases et les approches nécessaires pour naviguer dans ce domaine en constante évolution.

Les besoins informatiques d'un scientifique

Les besoins informatiques d'un scientifique en général (et plus particulièrement d'un mathématicien appliqué) dans son travail quotidien, peuvent se résumer ainsi :

- simuler des systèmes complexes,
- acquérir des données (issues de simulations et/ou d'expériences),

- manipuler et traiter ces données,
- visualiser les résultats et les interpréter,
- communiquer les résultats (par exemple produire des figures pour des publications ou des présentations).

Un outils de programmation adapté à ce travail doit posséder les caractéristiques suivantes :

- disponibilité d'une riche collection d'algorithmes et d'outils de base,
- facile à apprendre (la programmation n'est pas son travail principal), par exemple parce qu'il se rapproche de son langage naturel, à savoir les mathématiques,
- un seul environnement/langage pour toutes les problèmes (simulations, traitement des données, visualisation),
- exécution et développement efficaces,
- facile à communiquer avec les collaborateurs (ou les étudiants, les clients).

Python répond à ce cahier de charges : il a une collection très riche de bibliothèques scientifiques, mais aussi beaucoup de bibliothèques pour des tâches non scientifiques, c'est un langage de programmation bien conçu et lisible, il est gratuit et open source.

Objectifs de ce cours

Ce cours vise deux objectifs : **apprendre à résoudre des problèmes (souvent issus des mathématiques mais pas que) en langage algorithmique et être capable d'écrire des petits programmes en Python qui implémentent ces algorithmes.**

Nous allons utiliser les ressources fournies par les ordinateurs pour tenter de traiter des problèmes aussi variés que possible, soit pour se simplifier la vie et gagner du temps en faisant faire par un ordinateur des calculs fastidieux que l'on pourrait sans doute faire soi même, soit au contraire pour s'attaquer à des questions que l'on ne pourrait en aucun cas traiter avec une feuille de papier et un crayon. Nous serons confrontés, dans différents domaines, à des problèmes qui sont souvent peu susceptibles d'être résolus de manière analytique. Ainsi, pour les traiter, nous devrons recourir à des méthodes numériques à l'aide de calculs sur ordinateur. L'objectif est d'apprendre comment aborder une question de manière à ce que le calcul permette d'obtenir une réponse satisfaisante, même si cela implique de reformuler le problème pour le rendre adapté à ce type de traitement.

Dans la licence Mathématiques à l'université de Toulon plusieurs ECUE s'appuieront sur des notions de base de la programmation informatique avec Python :

PIM-11 au semetre 1 (L1) : *programmation informatique pour les Mathématiques* (python)
M25 au semetre 2 (L1) : *modélisation informatique* (notebook IPython, modules SciPy, Matplotlib)
M43 au semetre 4 (L2) : *TP mathématiques* (notebook IPython, modules NumPy, SymPy, Panda)
M62 au semetre 6 (L3) : *analyse numérique* (notebook IPython, que se cache-t-il derrière `odeint` du module SymPy?)

Il est donc indispensable de bien acquérir des bases de programmation informatique en général et plus particulièrement en Python.

Bien noter que sous le mot "programmation" se cachent en fait deux activités : l'une consiste à faire l'analyse du problème à traiter afin d'élaborer un algorithme approprié, l'autre est la traduction de cet algorithme dans un langage de programmation compréhensible par l'ordinateur, dans notre cas Python.

Déroulement du cours et évaluation

CM 6h : 4 séances de 1h30
TD 6h : 4 séances de 1h30
TP 24h : 7 séances de 3h

CC 3h : 1 séance de 3h en salle de TP la dernière semaine de cours
CT 3h : 1 séance de 3h en salle de TP en janvier
Note finale $\max(\text{CT}; 0.3\text{CC} + 0.7\text{CT})$

Les CC et CT sont sous la forme de tests Moodle. Ils consistent en **une série de questions tirées aléatoirement dans une même banque de questions**. La **correction** est **automatique** : **vous devez écrire des codes python (scripts ou fonctions) qui seront vérifiés sur plusieurs tests tirés eux aussi aléatoirement.** Pour répondre à une question il suffira de copier-coller votre code dans la zone réponse puis de cliquer sur "vérifier".

Comment utiliser le polycopié

Ce polycopié ne dispense pas des séances de cours-TD-TP ni de prendre des notes complémentaires. Il est d'ailleurs important de **comprendre** et **apprendre** le cours **au fur et à mesure**. Ce document a pour but de vous épargner le travail de prendre des notes durant le cours et de vous permettre de vous concentrer sur les explications données oralement. Cependant, il ne contient pas toutes les informations présentées pendant le cours et peut comporter des erreurs. Si vous en trouvez, n'hésitez pas à me les signaler.

Il est possible que certaines parties de ce cours vous semblent simples et faciles à comprendre, tandis que d'autres parties peuvent sembler plus longues et complexes. Cela est tout à fait normal et ne doit pas vous inquiéter. Cependant, il est crucial de prendre le temps nécessaire pour maîtriser chaque concept abordé dans ce cours. Vous devez avancer à votre propre rythme, prendre le temps de digérer chaque information en relisant les parties plus complexes et les exemples, et en faisant les exercices correspondants.

Il est très important de pratiquer en parallèle de l'apprentissage, c'est le seul moyen d'assimiler réellement les concepts abordés ! Entraînez-vous sur les très nombreux exercices proposés. Dans ce texte il y a **327 exercices**, de difficulté variée et dont la **correction** est disponible sur la page moodle [1]. Veuillez bien noter que la meilleure méthode pour apprendre efficacement est de comprendre les concepts par soi-même en les mettant en pratique : **try it yourself!** Il est tout à fait normal de rencontrer des difficultés lors de la réalisation des exercices proposés. Dans ces cas-là, il est important de ne pas rester bloqué et de demander de l'aide à l'enseignant ou à ses camarades. Le cours est également une ressource précieuse pour trouver des explications et des exemples concrets pour résoudre les problèmes rencontrés. L'important est de ne pas abandonner et de persévérer pour progresser.

Enfin, il est normal que votre code diffère de celui proposé dans la correction, car il existe souvent plusieurs approches pour résoudre un même problème. Il est donc possible de répondre correctement à une question avec un code différent de celui proposé. De plus, la mise en forme d'une idée particulière peut impliquer de nombreux choix intermédiaires, ce qui peut donner lieu à des variations dans la manière de coder. Les différences entre deux solutions peuvent être sans importance réelle (variations mineures pour la réalisation d'une même démarche) mais peuvent aussi être significatives et les deux approches avoir des avantages et des qualités différentes relativement à des critères tels que l'efficacité à l'exécution, l'espace mémoire requis, la lisibilité, la facilité de généralisation pour résoudre un problème plus large, etc. Il est difficile de déterminer quelle est la meilleure solution algorithmique pour résoudre un exercice, car cela dépend des conditions particulières d'exploitation qui ne sont généralement pas mentionnées dans l'énoncé. De plus, une solution particulière ne reflète pas toutes les étapes de la démarche qui a permis de l'obtenir. Par ailleurs, une solution particulière est le résultat d'une démarche mais elle ne reflet pas toutes les étapes qui ont été suivies lors de cette démarche. Or, ce qu'il est important d'apprendre, c'est précisément la démarche qui permet de conduire l'analyse du problème, depuis la lecture de l'énoncé jusqu'à la formulation d'un algorithme. À cet égard, un corrigé parmi les nombreux corrigé possibles n'apporte quasiment rien mais peut vous aider d'une part à vérifier si votre code est fonctionnel et il est toujours instructif de regarder et comparer l'approche proposée avec son propre code.

Conventions pour la présentation des exercices

Les exercices marqués par le symbole ✍ devront être préparés avant d'aller en TP (certains auront été traités en TD). Les exercices marqués par le symbole ☆ sont des exercices un peu plus difficiles qui ne seront pas traités en TD (ni en TP sauf si vous avez terminé tous les autres exercices prévus). Enfin, les exercices marqués par le symbole ⚠ sont des Pydéfis. La correction de ces exercices n'est pas publique à la demande de l'administrateur du site. Je vous conseille de vous inscrire et vérifier vos réponses par vous même ☺.

Conventions pour la présentation du code

Pour l'écriture du code, plusieurs présentations seront utilisées :

- les instructions précédées de trois chevrons (>>>) sont à saisir dans une session interactive (si l'instruction produit un résultat, il est affiché une fois l'instruction exécutée)

```
>>> 1+1
2
```

- les instructions sans chevrons sont des bouts de code à écrire dans un fichier. Si le résultat d'exécution du script est présentés, elle apparaît immédiatement après le script :

```
print("Bonjour !")
```

1. https://moodle.univ-tln.fr/course/view.php?id=4968

```
Bonjour !
```

Utiliser Python en ligne

Il existe plusieurs sites où l'on peut écrire et tester ses propres programmes. Les programmes s'exécutent dans le navigateur sans qu'il soit nécessaire de se connecter, de télécharger des plugins ou d'installer des logiciels.

En particulier :

- pour écrire et exécuter des script contenant des graphes `matplotlib`
  https://console.basthon.fr/
- pour comprendre l'exécution d'un code pas à pas : Visualize code and get live help
  http://pythontutor.com/visualize.html

Version de référence

La version de Python qui a servi de référence pour le polycopié est la version 3.10

Certaines précautions sont à prendre si on utilise une version plus ancienne. La différence la plus visible pour vous apparaîtra avec les `f-strings` :

```
>>> # un exemple de f-string
>>> age = 10
>>> f"Jean a {age} ans"
'Jean a 10 ans'
```

Cette construction — que j'utilise très fréquemment dans le polycopié — n'a été introduite qu'en Python-3.6, ainsi si vous utilisez Python-3.5 vous verrez ceci

```
>>> age = 10
>>> f"Jean a {age} ans"
File "<stdin>", line 1
f"Jean a {age} ans"
^
SyntaxError: invalid syntax
```

Dans ce cas il faut remplacer ce code avec la méthode `format` et dans le cas présent il faudrait remplacer par ceci

```
>>> age = 10
>>> "Jean a {} ans".format(age)
'Jean a 10 ans'
```

Voici un premier fragment de code Python qui affiche la version de Python utilisée.

```
>>> import sys
>>> print(sys.version_info)
sys.version_info(major=3, minor=10, micro=6, releaselevel='final', serial=0)
```

Obtenir Python

Pour installer Python il suffit de télécharger la dernière version qui correspond au système d'exploitation (Windows ou Mac) à l'adresse www.python.org. Pour ce qui est des systèmes Linux, il est très probable que Python soit déjà installé.

Quel éditeur ?

La première étape pour utiliser n'importe quel langage de programmation consiste à configurer les outils nécessaires. Pour créer un script, vous devez d'abord utiliser un éditeur de texte. Attention ! Pas Word ni Libre Office Writer sans quoi les scripts ne fonctionneront pas. Il faut **un éditeur de texte pur**. En fonction du système sur lequel vous travaillez voici des nom d'éditeurs qui peuvent faire l'affaire : NOTEPAD, NOTEPAD++, EDIT, GEDIT, GEANY, KATE, VI, VIM, EMACS, NANO, SUBLIME TEXT.

IDE ?

On peut utiliser un simple éditeur de texte et un terminal ou **des environnements de développement spécialisés** (appelés **IDE** pour *Integrated Development Environment*) comme IDLE, THONNY ou SPYDER. Ces derniers se présentent sous la forme d'une application et se composent généralement d'un éditeur de code, d'une fenêtre appelée indifféremment *console*, *shell* ou *terminal* Python, d'un débogueur et d'un générateur d'interface graphique.

L'IDE **THONNY** [2] installe Python 3.7 en même temps et c'est peut-être le choix plus simple pour commencer (notamment sous Windows). Il gère aussi l'installation de modules tels que `scipy`, `sympy` et `matplotlib`.

Si vous n'êtes plus des néophytes en programmation, vous trouverez votre bonheur avec VSCODE [3].

Installer Anaconda

À partir du deuxième semestre, vous allez travailler dans des `notebook IPython` (documents "mixtes" contenant du texte et du code Python). Vous pouvez alors insteller Anaconda qui installera Python avec des modules utiles en mathématiques (`Matplotlib`, `NumPy`, `SciPy`, `SymPy` etc.), ainsi que `Idle3`, `IPython` et `Spyder`. Les procédures d'installations détaillées selon chaque système d'exploitation sont décrites à l'adresse : https://www.anaconda.com/products/individual. Les procédures suivantes sont un résumé rapide de la procédure d'installation.

- Installation sous Windows.
  1. Télécharger Anaconda 5.2 (ou plus récent) pour Python 3.6 (ou plus récent) à l'adresse : https://www.anaconda.com/download/#windows
  2. Double cliquer sur le fichier téléchargé pour lancer l'installation d'Anaconda, puis suivre la procédure d'installation (il n'est pas nécessaire d'installer VS Code).
  3. Une fois l'installation terminée, lancer Anaconda Navigator à partir du menu démarrer.
- Installation sous macOS.
  1. Télécharger Anaconda 5.2 (ou plus récent) pour Python 3.6 (ou plus récent) à l'adresse : https://www.anaconda.com/download/#macos
  2. Double cliquer sur le fichier téléchargé pour lancer l'installation d'Anaconda, puis suivre la procédure d'installation (il n'est pas nécessaire d'installer VS Code).
  3. Une fois l'installation terminée, lancer Anaconda Navigator à partir de la liste des applications.
- Installation sous Linux.
  1. Télécharger Anaconda 5.2 (ou plus récent) pour Python 3.6 (ou plus récent) à l'adresse : https://www.anaconda.com/download/#linux
  2. Exécuter le fichier téléchargé avec bash puis suivre la procédure d'installation (il n'est pas nécessaire d'installer VS Code).
  3. Une fois l'installation terminée, taper anaconda-navigator dans un nouveau terminal pour lancer Anaconda Navigator.

Le Zen de Python

Le "Zen de Python" résume la philosophie du langage :

> The Zen of Python, by Tim Peters
> Beautiful is better than ugly.
> Explicit is better than implicit.
> Simple is better than complex.
> Complex is better than complicated.
> Flat is better than nested.
> Sparse is better than dense.
> Readability counts.
> Special cases aren't special enough to break the rules.
> Although practicality beats purity.
> Errors should never pass silently.
> Unless explicitly silenced.
> In the face of ambiguity, refuse the temptation to guess.
> There should be one– and preferably only one –obvious way to do it.
> Although that way may not be obvious at first unless you're Dutch.

2. https://thonny.org/ et https://realpython.com/python-thonny/
3. https://docs.microsoft.com/fr-fr/learn/modules/python-install-vscode/1-introduction

Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

CHAPITRE 1

Notions de base de Python

Python est un langage développé en 1 989 par Guido VAN ROSSUM, aux Pays-Bas. Le nom est dérivé de la série télévisée britannique des *Monty Python's Flying Circus*. La dernière version de Python est la version 3. Plus précisément, la version 3.11 a été publiée en décembre 2022. [1]

La Python Software Foundation est l'association qui organise le développement de Python et anime la communauté de développeurs et d'utilisateurs.

Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

- Il est multi-plateforme. C'est-à-dire qu'il fonctionne sur de nombreux systèmes d'exploitation : Linux, Mac OS X, Windows, Android, etc. De plus, un programme écrit sur un système fonctionne sans modification sur tous les systèmes.
- Il est gratuit. Vous pouvez l'installer sur autant d'ordinateurs que vous voulez (même sur votre téléphone).
- C'est un langage de haut niveau. Il demande relativement peu de connaissance sur le fonctionnement d'un ordinateur pour être utilisé.
- C'est un langage interprété. Les programmes Python n'ont pas besoin d'être compilés en code machine pour être exécuté, mais sont gérés par un interpréteur (contrairement à des langages comme le C ou le C++).
- Il est orienté objet. C'est-à-dire qu'il est possible de concevoir en Python des entités qui miment celles du monde réel (une cellule, une protéine, un atome, etc.) avec un certain nombre de règles de fonctionnement et d'interactions. L'avantage d'un langage interprété est que les programmes peuvent être testés et mis au point rapidement, ce qui permet à l'utilisateur de se concentrer davantage sur les principes subjacents du programme et moins sur la programmation elle-même. Cependant, un programme Python peut être exécuté uniquement sur les ordinateurs qui ont installé l'interpréteur Python.
- Il est relativement simple à prendre en main.
- Il est très utilisé en sciences et plus généralement en analyse de données.

Toutes ces caractéristiques font que Python est désormais enseigné dans de nombreuses formations, depuis l'enseignement secondaire jusqu'à l'enseignement supérieur.

IDE : Idle et Thonny

Dans les salles de TP nous utiliserons l'IDE IDLE. [2] Si vous voulez utiliser vos ordinateurs personnels, je conseil d'installer l'IDE THONNY. [3] Ils se présentent sous la forme d'une application et se composent d'une fenêtre appelée indifféremment *console*, *shell* ou *terminal* Python, et d'un éditeur de code.

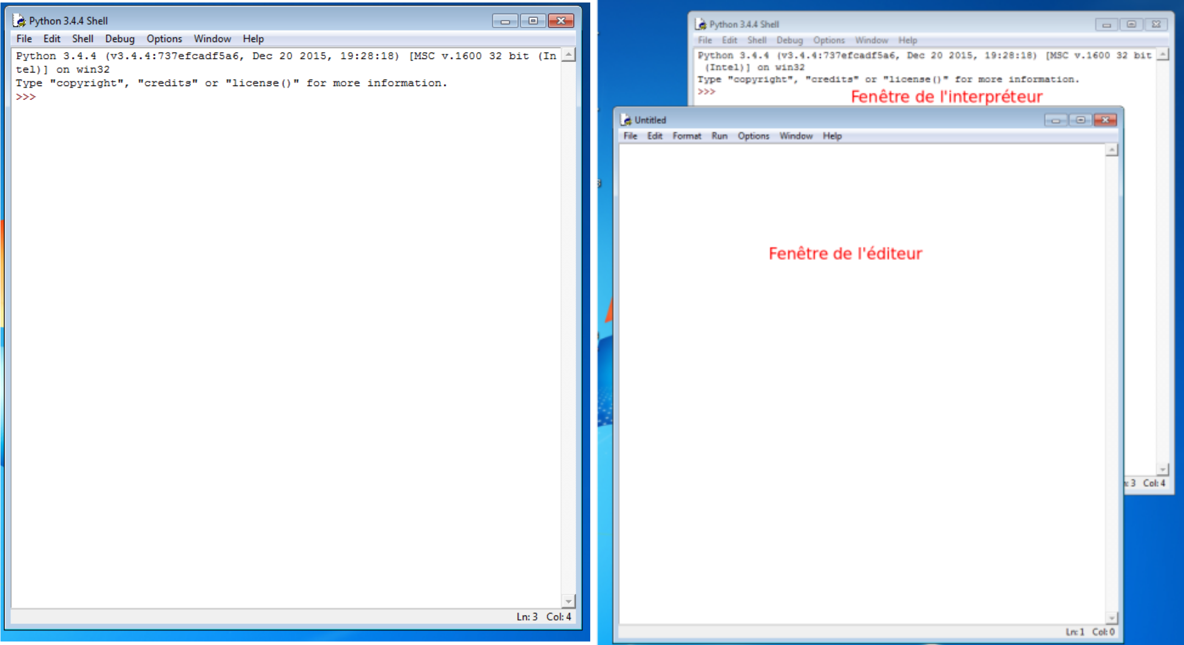
1.1. Mode interactif et mode script

L'exécution d'un programme Python se fait à l'aide d'un *interpréteur*. Il s'agit d'un programme qui va traduire les instructions écrites en Python en langage machine, afin qu'elles puissent être exécutées directement par l'ordinateur. Cette traduction se fait à la volée, tout comme les interprètes traduisent en temps réel les interventions des différents parlementaires lors des sessions du parlement Européen, par exemple. On dit donc que Python est un *langage interprété*.

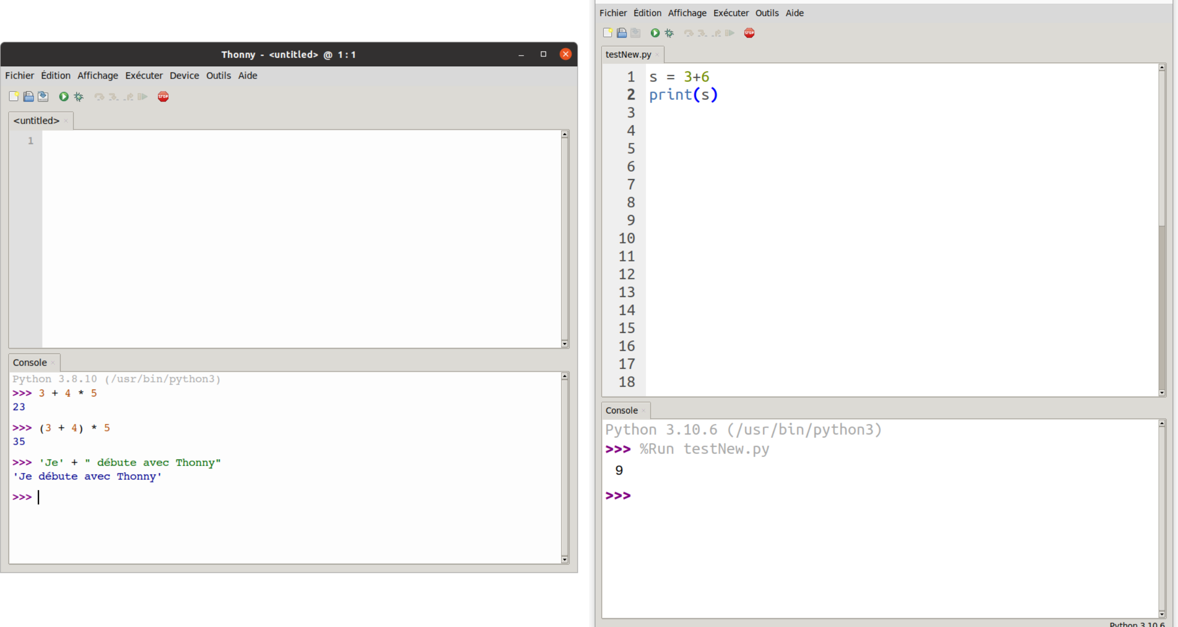
Il y a *deux modes d'utilisation de Python.*

- Dans le **mode interactif**, aussi appelé **mode console**, **mode shell** ou **terminal Python**, l'interpréteur vous permet d'encoder les instructions une à une. Aussitôt une instruction encodée, il suffit d'appuyer sur la touche «Entrée» pour que l'interpréteur l'exécute.

1. La version 2 de Python est désormais obsolète et a cessé d'être maintenue après le 1er janvier 2020. Dans la mesure du possible évitez de l'utiliser.
2. https://dane.ac-lyon.fr/spip/IMG/scenari/python/co/GC_4-2_prog_Idle.html
3. Voir par exemple https://thonny.org/ et https://realpython.com/python-thonny/. C'est probablement le choix plus simple pour commencer (notamment sous Windows). Il gère aussi très simplement l'installation de modules tels que `scipy`, `sympy` et `matplotlib`.



(a) Lors de son lancement, Idle démarre par défaut l'interpréteur Python et affiche une console permettant d'utiliser Python à la ligne de commande (mode interactif). Pour démarrer l'éditeur, cliquer sur `File`, puis `New File` et une nouvelle fenêtre apparaît (mode script).



(b) Lors de son lancement, Thonny montre la console (en bas) permettant de taper du code Python et de visualiser directement le résultat de son exécution (mode interactif). En haut se trouve l'éditeur pour la saisie et la sauvegarde des programmes (scripts).

FIGURE 1.1. – Captures d'écran de deux IDE

- Dans le **mode script**, il faut avoir préalablement écrit toutes les instructions du programme dans un fichier texte, et l'avoir enregistré sur l'ordinateur avec l'extension `.py`. Une fois cela fait, on demandera à Python de lire ce fichier et exécuter son contenu, instruction par instruction, comme si on les avait tapées l'une après l'autre dans le mode interactif.

1.1.1. Mode interactif

Pour commencer on va apprendre à utiliser Python directement : [4]

- ouvrir un IDE (ou écrire `python3` dans un terminal puis appuyer sur la touche «Entrée») ;
- un invite de commande, composé de trois chevrons (>>>), apparaît : cette marque visuelle indique que Python est prêt à lire une commande. Il suffit de saisir à la suite une instruction puis d'appuyer sur la touche «Entrée». Pour commencer, comme le veux la tradition informatique, on va demander à Python d'afficher les fameux mots «Hello world» : [5]

```
>>> print("Hello world")
Hello world
```

  Si l'instruction produit un résultat, il est affiché une fois l'instruction exécutée.
- La console Python fonctionne comme une simple calculatrice : on peut saisir une expression dont la valeur est renvoyée dès qu'on presse la touche «Entrée», par exemple

```
>>> 2*7+8-(100+6)
-84
>>> 2**5
32
>>> 7/2; 7/3
3.5
2.3333333333333335
>>> 34//5; 34%5 # quotient et reste de la
↪ division euclidienne de 34 par 5
6
4
```

```
>>> a = 4
>>> b = 10
>>> c = a*b
>>> print(a,b,c)
4 10 40
```

- Pour quitter le mode interactif, il suffit d'exécuter l'instruction `exit()`. Il s'agit de nouveau d'une fonction prédéfinie de Python permettant de quitter l'interpréteur.

Le mode interactif est très pratique pour rapidement tester des instructions et directement voir leurs résultats. Son utilisation reste néanmoins limitée à des programmes de quelques instructions. En effet, devoir à chaque fois retaper toutes les instructions s'avérera vite pénible.

1.1.2. Mode script

Dans le **mode script**, il faut avoir préalablement écrit toutes les instructions du programme dans un fichier texte, et l'avoir enregistré sur l'ordinateur. On utilise généralement l'extension de fichier `.py` pour des fichiers contenant du code Python. Une fois cela fait, l'interpréteur va lire ce fichier et exécuter son contenu, instruction par instruction, comme si on les avait tapées l'une après l'autre dans le mode interactif. Les résultats intermédiaires des différentes instructions ne sont par contre pas affichés ; seuls les affichages explicites (avec la fonction `print`, par exemple) se produisent.

- Tout d'abord, commençons par ouvrir un éditeur. On voit qu'il n'y a rien dans cette nouvelle fenêtre (pas d'en-tête ni de chevrons comme dans l'INTERPRÉTEUR). Ce qui veut dire que ce fichier est uniquement pour les commandes : Python n'interviendra pas avec ses réponses lorsque on écrira le programme et ce tant que on ne le lui demandera pas.
- Ce que l'on veut, c'est de sauver les quelques instructions qu'on a essayées dans l'interpréteur. Alors faisons-le soit en tapant soit en copiant-collant ces commandes dans ce fichier.

4. Il ne s'agit pas, pour l'instant, de s'occuper des règles exactes de programmation, mais seulement d'expérimenter le fait d'entrer des commandes dans Python.

5. Pour les curieux qui veulent découvrir mille et mille manière d'afficher "Hello, World!" dans tous les langages informatiques de la Terre, voyez ici http://fr.wikipedia.org/wiki/Liste_de_programme_Hello_world

- Sauvons maintenant le fichier sous le nom `primo.py` : la commande «Save» (Sauver) se trouve dans le menu «File» (Fichier), sinon nous pouvons utiliser le raccourcis Ctrl + S .
- Ayant sauvé le programme, pour le faire tourner et afficher les résultats dans la fenêtre de l'INTERPRÉTEUR il suffit d'utiliser le raccourci clavier F5 (ou de taper dans le terminal `python primo.py` puis appuyer sur la touche «Entrée»).
- Maintenant qu'on a sauvé le programme, on est capable de le recharger : on va tout fermer, relancer l'éditeur et ouvrir le fichier.

ATTENTION

Noter la différence entre l'output produit en mode **interactif** :

```
>>> a = 10
>>> a # cette instruction affiche la valeur de a en mode interactif
10
>>> print("J'ai fini")
J'ai fini
>>> print("a =",a)
a = 10
```

et l'output produit en mode **script**

```
a = 20
a # cette instruction n'a pas d'effet en mode script
print("J'ai fini")
print("a =",a)
```

dont l'output est

```
J'ai fini
a = 20
```

Dans le mode **interactif**, la valeur de la variable `a` est affichée directement tandis que dans le mode **script**, il faut utiliser `print(a)`.

1.2. Commentaires

Il est utile de laisser un commentaire pour expliquer les parties de votre code qui ne sont pas immédiatement évidentes. Le symbole dièse (`#`) indique le début d'un commentaire : tous les caractères entre `#` et la fin de la ligne sont ignorés par l'interpréteur. Les commentaires doivent être utilisés avec modération, maintenus à jour, et indentés de la même façon que la ligne de code suivante.

Pour commenter (resp. dé-commenter) plusieurs lignes avec nos IDE :

IDLE Souligner les lignes à commenter (resp. dé-commenter) et appuyer sur les touches Ctrl + D (resp. Ctrl + Shift ⇑ + D);

Thonny Souligner les lignes à commenter ou dé-commenter et appuyer sur les touches Ctrl + Shift ⇑ + 3 .

1.3. Indentation

En Python (contrairement aux autres langages) c'est l'indentation (les espaces en début de chaque ligne) qui détermine les blocs d'instructions (boucles, sous-routines, etc.). Cette obligation de coder en utilisant l'indentation permet d'obtenir du code plus lisible et plus propre.

Pour produire une indentation on peut soit appuyer 4 fois sur la barre [ ] ou appuyer une fois sur la touche tabulation ⇆ . L'indentation doit être homogène (soit des espaces, soit des tabulations, mais pas un mélange des deux). Dans ce polycopié on notera chaque tabulation avec une flèche comme suit :

```
for i in range(5):    # aucune tabulation
    for j in range(4): # une tabulation
        print(i+j)       # deux tabulations
```

Il est recommandé d'utiliser 4 espaces, pour que les développeurs n'obtiennent pas d'erreurs d'«indentation inattendue» lorsqu'ils copient du code.

Pour indenter (resp. dés-indenter) plusieurs lignes avec nos IDE :

IDLE Souligner les lignes à indenter (resp. dés-indenter) et appuyer sur les touches (resp.) ;
Thonny Souligner les lignes à indenter (resp. dés-indenter) et appuyer sur la touche [Tab] (resp. [Ctrl] + [Shift ⇑] + [Tab]).

1.4. Types primitifs

Les différents types primitifs sont :

- `int` : un entier(*Integers* en anglais)
- `float` : un nombre décimal (les virgules flottantes, *Float* en anglais)
- `str` : une chaîne de caractères (*Strings* en anglais)
- `bool` : un booléen (`True` ou `False`)

On peut identifier le type d'une variable en utilisant `type()` :

```
>>> type(10)
<class 'int'>
```

```
>>> type(3.5)
<class 'float'>
```

```
>>> type("Ciao")
<class 'str'>
```

```
>>> type(3>5)
<class 'bool'>
```

Noter que les nombre `100` et `100.0` ont l'air de se ressembler, mais dans Python, l'un est un entier et l'autre est une virgule flottante.

```
>>> type(100), type(100.0)
(<class 'int'>, <class 'float'>)
```

Il existe ensuite quatre structures de référence : les listes `list`, les tuples `tuple`, les dictionnaires `dict` et les ensembles `set`. Ces structures sont en fait des objets qui peuvent contenir d'autres objets. On les verra au prochaine chapitre.

```
>>> type( [ 10 , "toto" , 3.14 , True ] )
<class 'list'>
>>> type( ( 10 , "toto" , 3.14 , True ) )
<class 'tuple'>
```

```
>>> type( set( [10 , "toto" , 10 , True] ) )
<class 'set'>
>>> type( { "a":10 , "b":5 } )
<class 'dict'>
```

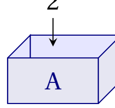
1.5. Variables et affectation

Une variable est une sorte de boîte contenant un objet, par exemple une valeur. Cette boîte est stockée dans un gigantesque entrepôt (la mémoire de l'ordinateur). Son emplacement est très précisément répertorié (adresse mémoire). À chaque boîte est attribué un nom afin de facilement l'identifier. On va également avoir besoin d'appliquer différentes opérations sur ces boîtes comme les vider, modifier le contenu, transférer le contenu de l'une à l'autre, etc.

La session interactive suivante avec l'INTERPRÉTEUR Python illustre ce propos (>>> est le prompt) :

```
>>> A = 2
>>> print(A)
2
```

L'affectation `A = 2` crée une association entre le nom `A` et le nombre entier 2 : la boîte de nom `A` contient la valeur 2.



Il faut bien prendre garde au fait que **l'instruction d'affectation** (=) **n'a pas la même signification que le symbole d'égalité (=) en mathématiques** (ceci explique pourquoi l'affectation de 2 à `A`, qu'en Python s'écrit `A = 2`, en algorithmique se note souvent $A \leftarrow 2$).

De la même façon qu'avec l'étiquetage des boîtes lors d'un déménagement, le nom d'une variable doit toujours représenter son contenu, avec des noms clairs et précis. Avec des noms bien choisis, on comprend tout de suite ce que calcule le code suivant :

```
base = 8
hauteur = 3
aire = base * hauteur / 2
print(aire)
```

Attention

Voici quelques recommandations générales pour choisir un nom :

- Utilisez des noms descriptifs dans votre code. Les noms de variables descriptifs et spécifiques simplifient la vie et facilitent la lecture et la modification du code.
- Les noms de variables sont sensibles à la casse : `age`, `Age` et `AGE` sont trois variables différentes.
- Utilisez uniquement des caractères alphanumériques et des tirets bas «_» (appelé *underscore* en anglais) et pas d'accents ! Par ailleurs, un nom de variable ne peut pas utiliser d'espace, ne doit pas débuter par un chiffre et il n'est pas recommandé de le faire débuter par le caractère _ (sauf cas très particuliers). De plus, il faut absolument éviter d'utiliser un mot «réservé» par Python comme nom de variable, par exemple : `and as assert break class continue def del elif else except False finally for from global if import in is lambda not or pass print raise range return True try while with yield`
- Le code est lu bien plus souvent qu'il n'est écrit : évitons donc de nous compliquer la tâche avec des abréviations, des mots hachés et des variables à une lettre. Le temps gagné sur le moment ne rattrapera jamais le temps perdu à relire ; et la qualité générale du code s'en ressentira. Voici quelque convention de nommage selon le PEP 8 :[6]
  - Écrivez les noms de variables en minuscules, avec des underscores (tirets bas). C'est ce qu'on appelle la convention « snake_case » :

    ```
    name = "Jeanne"
    fuel_level = 100
    famous_singers = ["Céline Dion", "Michael Jackson", "Edith Piaf"]
    ```

  - Écrivez les variables constantes en majuscules, avec des underscores :

    ```
    DAYS_PER_WEEK = 7
    PERSONAL_EMAIL = "myemail@email.com"
    ```

    En effet, en Python on ne crée pas de variables dont la valeur ne peut pas être modifiée. Cependant, si vous voulez qu'une valeur demeure constante, le simple fait de l'écrire TOUT_EN_MAJUSCULES signale aux autres développeurs de ne pas écrire du code qui la modifie.

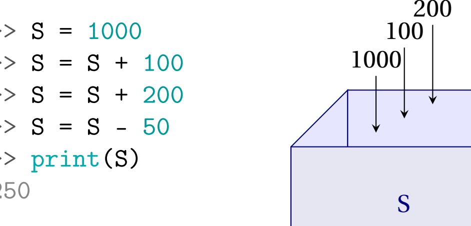
Une fois une variable initialisée, on peut modifier sa valeur en utilisant de nouveau l'opérateur d'affectation (=). La valeur actuelle de la variable est remplacée par la nouvelle valeur qu'on lui affecte. Le type de la variable va changer de lui-même en fonction de la valeur stockée dedans.

Dans l'exemple suivant, on initialise une variable à la valeur 12 et on remplace ensuite sa valeur par 99 :

```
>>> a = 12
>>> a = 99
>>> print(a)
99
```

Un autre exemple (on part d'une somme S = 1000, puis on lui ajoute 100, puis 200, puis on enlève 50). Il faut comprendre l'instruction `S=S+100` comme ceci : «je prends le contenu de la boîte S, je rajoute 100, je remets tout dans la même boîte».

```
>>> S = 1000
>>> S = S + 100
>>> S = S + 200
>>> S = S - 50
>>> print(S)
1250
```



6. PEP est l'abréviation de *Python Enhancement Proposal* (proposition d'amélioration de Python). Les PEP sont des documents de conception pour la communauté Python. Elles décrivent des nouvelles fonctionnalités pour Python, ses processus ou son environnement. La PEP qui sert de guide de style pour Python est la PEP 8. Il s'agit d'une longue liste de pratiques suggérées pour les développeurs Python. Créée en 2001, elle a été écrite par Guido van Rossum, Barry Warsaw et Nick Coghlan, et elle est mise à jour régulièrement pour refléter les développements du langage. Pour vérifier tout élément spécifique, vous pouvez consulter le document officiel sur le site de Python https://peps.python.org/pep-0008/.

Remarque

Il est important de garder le contrôle des valeurs des variables. En particulier, lors de l'apprentissage, il est important de s'assurer que le code fait ce qu'il est censé faire, et ce chaque fois que nous créons ou modifions une variable. L'affichage avec la fonction `print()` est un moyen facile de vérifier que les modifications de variables correspondent à nos intentions.

On souhaite parfois conserver en mémoire le résultat de l'évaluation d'une expression arithmétique en vue de l'utiliser plus tard. Par exemple, si on recherche les solutions de l'équation ax^2+bx+c, on doit mémoriser la valeur du discriminant pour pouvoir calculer les valeurs des deux racines réelles distinctes, lorsqu'il est strictement positif.

```
>>> a = 1
>>> b = 2
>>> c = 4
>>> # on peut aussi écrire les instructions sur une seule ligne
>>> # a = 1; b = 2; c = 4;
>>> delta = b**2-4*a*c
>>> print(delta)
-12
```

Avant de pouvoir accéder au contenu d'une variable, il faut qu'elle soit initialisée, c'est-à-dire qu'elle doit posséder une valeur. Si on tente d'utiliser une variable non initialisée, l'exécution du programme va s'arrêter et l'interpréteur Python va produire une **erreur** d'exécution. Voyons cela avec l'exemple de programme suivant :

```
>>> a = 178
>>> print('Sa taille est :')
Sa taille est :
>>> print(toto)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'toto' is not defined
```

L'avant-dernière ligne reprend l'instruction qui a causé l'erreur d'exécution (à savoir `print(toto)` dans notre cas). La dernière ligne fournit une explication sur la cause de l'erreur (celle qui commence par `NameError`). Dans cet exemple, elle indique que le nom `toto` n'est pas défini, c'est-à-dire qu'il ne correspond pas à une variable initialisée.

Remarque (Les messages d'erreur)

Lorsque on écrits du code, on fait inévitablement des erreurs et on obtient des messages d'erreur. C'est normal! Il est donc important d'apprendre à lire les messages d'erreur afin de pouvoir les corriger rapidement et de continuer à coder.

L'interprétation et le traitement des erreurs générées par un langage de programmation sont des compétences fondamentales. En Python, **les messages d'erreur se lisent de bas en haut** : la dernière ligne indique le type d'erreur et l'avant dernière donne l'instruction qui a généré l'erreur. Ce sont les deux informations les plus importantes. Le reste du message décrit la "pile" des appels ayant généré l'erreur. Attention, ce n'est pas forcément la ligne qui a généré le message d'erreur qu'il faut corriger pour que le programme fonctionne correctement.

Attention cependant : on est souvent confronté à des programmes qui n'engendrent pas de message d'erreur mais qui n'ont pas les comportement attendus. Il convient alors d'admettre que le comportement de l'interpréteur Python est correct et il faut essayer de détailler ce qu'il a généré comme sorties indépendamment du but recherché.

Affectations multiples en parallèle On peut aussi effectuer des affectations parallèles :[7]

```
>>> x = y = z = 50
>>> print(x)
50
>>> print(y)
50
>>> print(z)
50
```

```
>>> a, b = 128, 256
>>> print(a)
128
>>> print(b)
256
```

7. Il s'agit de tuples définies sans écrire explicitent les parenthèses

et ainsi échanger facilement les deux variables :

```
>>> a, b = 128, 256
>>> a, b = b, a
>>> print(a)
256
>>> print(b)
128
```

ATTENTION
Noter la différence lorsqu'on écrit les instructions suivantes :

```
>>> a = 10; b = 5
>>> a = b
>>> b = a
>>> print(a); print(b)
5
5
```

```
>>> a = 10; b = 5
>>> c = b
>>> b = a
>>> a = c
>>> print(a); print(b)
5
10
```

```
>>> a = 10; b = 5
>>> a = a+b
>>> b = a-b
>>> a = a-b
>>> print(a); print(b)
5
10
```

La première approche n'échange pas le contenu des deux variables. En effet, la première instruction `a = b` a pour effet de mettre le contenu de la variable `b` dans la variable `a` et d'y effacer le contenu précédent. Par conséquent, avant de mettre le contenu de `b` dans `a`, il faut sauvegarder dans une troisième variable le contenu de `a` pour pouvoir ensuite le récupérer afin de le mettre dans `b`. C'est le rôle de `c` dans le deuxième exemple.

Le troisième exemple exploite une propriété mathématique mais risque de donner quelque surprise si les nombres à échanger ne sont pas des entiers et se leur différence est très grande (voir l'annexe A).

1.6. Nombres

Il y a deux types numériques primitifs en Python.

- Le type `int` (nombres entiers) permet de représenter n'importe quel nombre entier, peu importe sa taille.
- Le type `float` (nombres décimaux) permet de représenter des nombres comportant une partie décimale (comme 3.14 ou $2.1e-23$) avec au plus 15 chiffres significatifs, compris entre 10^{-308} et 10^{308}. Nota bene : on utilise le point comme séparateur décimal lorsqu'on doit écrire des nombres à virgule. La valeur spéciale `math.inf` représente l'infini (voir chapitre sur les modules).

Pour afficher le plus petit/grand flottant on peut écrire

```
>>> import sys
>>> min_float =sys.float_info.min
>>> max_float =sys.float_info.max
>>> print(min_float, max_float)
2.2250738585072014e-308 1.7976931348623157e+308
>>> # en effet, si on ajoute 1 on a encore le meme résultat
>>> max_float_p1 = max_float+1
>>> print(max_float_p1-max_float)
0.0
```

Pour afficher le plus petit nombre qui peut être ajouté à 1, on peut écrire

```
>>> import sys
>>> eps_float =sys.float_info.epsilon
>>> print(eps_float)
2.220446049250313e-16
>>> # en effet, si on ajoute la motié de eps on a encore le meme résultat
>>> n = 1
>>> n1 = n+eps_float
>>> n2 = n+eps_float/2
>>> print(n,n1,n2)
```

```
1 1.0000000000000002 1.0
>>> print(n==n1,n==n2) # est-ce que n est égale à n1 ? est-ce que n est égale à n2 ?
False True
```

1.7. Opérations arithmétiques

Dans Python on a les opérations arithmétiques usuelles :

`+`	Addition	`**`	Puissance, Exponentiation
`-`	Soustraction	`//`	Division entière, Quotient de la division euclidienne
`*`	Multiplication	`%`	Modulo, Reste de la division euclidienne
`/`	Division		

Quelques exemples :

```
>>> a = 100
>>> b = 17
>>> c = a-b
>>> print(a,b,c)
100 17 83
```

```
>>> a = 2
>>> c = b+a
>>> print(a,b,c)
2 17 19
```

```
>>> a = 3
>>> b = 4
>>> c = a
>>> a = b
>>> b = c
>>> print(a,b,c)
4 3 3
```

Priorités. Les opérateurs arithmétiques possèdent chacun une **priorité** qui définit dans quel ordre les opérations sont effectuées. Par exemple, lorsqu'on écrit `1 + 2 * 3`, la multiplication va se faire avant l'addition. Le calcul qui sera effectué est donc `1 + (2 * 3)`. Dans l'ordre, l'opérateur d'exponentiation est le premier exécuté, viennent ensuite les opérateurs *, /, // et %, et enfin les opérateurs + et -. Lorsqu'une expression contient plusieurs opérations de même priorité, ils sont évalués de gauche à droite. Ainsi, lorsqu'on écrit `1 - 2 - 3`, le calcul qui sera effectué est `(1 - 2) - 3`. En cas de doutes, vous pouvez toujours **utiliser des parenthèses** pour rendre explicite l'ordre d'évaluation de vos expressions arithmétiques.

Typecasting. Si le résultat d'une opération entre deux entiers est censé être un nombre décimal, Python va automatiquement le convertir en `float`. De plus, la division (même si le résultat est censé être entier) renverra forcément un `float` également. Cependant, on peut forcer la conversion d'une variable dans un type bien défini. Ceci est appelé du *typecasting*, car en faisant ainsi, on remodèle (*cast* en anglais) le type d'une variable. Pour ce faire, on a besoin des fonctions correspondantes

1. `int(n)` pour convertir n en un entier,
2. `float(x)` pour convertir v en un nombre décimal.

```
>>> a = 100
>>> b = 10.0
>>> c = a-b
>>> print(a,type(a),b,type(b),c,type(c))
100 <class 'int'> 10.0 <class 'float'> 90.0 <class 'float'>
>>> a_bis = float(a)
>>> print(a,type(a),a_bis,type(a_bis))
100 <class 'int'> 100.0 <class 'float'>
>>> b_bis = int(b)
>>> print(b,type(b),b_bis,type(b_bis))
10.0 <class 'float'> 10 <class 'int'>
```

Division entière. Deux opérations arithmétiques sont exclusivement utilisées pour effectuer des **calculs en nombres entiers** : la division entière (//) et le reste de la division entière (%).

```
>>> print( 9//4 )
2
>>> print( 9%4 )
1
>>> print(divmod(9,4))
(2, 1)
```

Lorsqu'on divise un nombre entier D (appelé dividende) par un autre nombre entier d (appelé diviseur), on obtient deux résultats : un quotient q et un reste r, tels que $D = qd + r$ (avec $r < d$). La valeur q est le résultat de la division entière et la valeur r celui du reste de cette division. Par exemple, si on divise 17 par 5, on obtient un quotient de 3 et un reste de 2 puisque $17 = 3 \times 5 + 2$. Ces deux opérateurs sont très utilisés dans plusieurs situations précises. Par exemple, pour déterminer si un nombre entier est pair ou impair, il suffit de regarder le reste de la division entière par deux. Le nombre est pair s'il est nul et est impair s'il vaut 1. Une autre situation où ces opérateurs sont utiles concerne les calculs de temps. Si on a un nombre de secondes et qu'on souhaite le décomposer en minutes et secondes, il suffit de faire la division par 60. Le quotient sera le nombre de minutes et le reste le nombre de secondes restant. Par exemple, 175 secondes correspond à `175//60=2` minutes et `175%60=55` secondes.

Opérateurs augmentés. Lorsqu'on veut changer une variable en modifiant la valeur initiale via un opérateur basique, on peut utiliser une version plus courte. Il s'agit des **opérateurs augmentés** :

`a += b` équivaut à a = a+b   `a -= b` équivaut à a = a-b   `a *= b` équivaut à a = a*b
`a /= b` équivaut à a = a/b   `a **= b` équivaut à a = a**b   `a %= b` équivaut à a = a%b

Notions sur la précision des calculs flottants

Comme pour les entiers, les calculs sur les flottants sont, naturellement, réalisés par le processeur. Cependant contrairement au cas des entiers où les calculs sont toujours exacts, les flottants posent un problème de précision. Voyons tout d'abord comment se matérialise le problème :

```
>>> 0.2 + 0.4
0.6000000000000001
```

Il faut retenir que lorsqu'on écrit un nombre flottant sous forme décimale, la valeur utilisée en mémoire pour représenter ce nombre, parce que cette valeur est codée en binaire, ne représente pas toujours exactement le nombre entré. Il faut donc faire attention lorsqu'on veut comparer deux nombres flottant :

```
>>> 0.3 - 0.1 == 0.2
False
```

Dans ces cas, on préférera considérer que deux nombres flottants a et b coincident si $|a - b| <$ une valeur petite :

```
>>> a = 0.3 - 0.1
>>> b = 0.2
>>> abs(a-b) < 1.e-14
True
```

Les différentes erreurs d'arrondi qui se produisent à chaque étape du calcul s'accumulent et produisent un résultat parfois surprenant (voir l'annexe A). Dans une grande majorité des cas, ces erreurs d'arrondi ne sont pas pénalisantes. Il faut toutefois en être conscient car cela peut expliquer des comportements curieux.

Ce problème n'est pas spécifique au langage Python, il existe pour tous les langages, et il est bien connu. Il est dû à la technique de codage des nombres flottants sous forme binaire.

En corollaire de la discussion sur la précision des flottants, il faut savoir que le système de codage en mémoire impose aussi une limite. Les réels très petits, ou très grands, ne peuvent plus être représentés de cette manière. Estimons la valeur du plus petit flottant qui peut être représenté comme un flottant :

```
>>> 10**-323
1e-323
>>> 10**-324
0.0
```

Comme on le voit, 10^{-323} est correctement calculé alors que 10^{-324} est considéré comme nul.

La même limitation s'applique sur les grands nombres. Toutefois, cela est un peu moins évident, car comme toujours il faut faire attention aux types :

```
>>> 10**310  # c'est un int
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
>>> 10**310. # c'est un float
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result out of range')
```

On peut accéder à ces valeurs minimales et maximales pour les flottants comme ceci

```
>>> import sys
>>> print("Flottant minimum", sys.float_info.min)
Flottant minimum 2.2250738585072014e-308
>>> print("Flottant maximum", sys.float_info.max)
Flottant maximum 1.7976931348623157e+308
```

1.8. Type booléen, Opérateurs de comparaison et connecteurs logiques

Le type booléen est l'un des types de données intégrés fournis par Python, qui représente l'une des deux valeurs, à savoir `True` (vrai) ou `False` (faux).

Les opérateurs de comparaison renvoient `True` si la condition est vérifiée, `False` sinon. Ces opérateurs sont les suivants :

On écrit	<	>	<=	>=	==	!=	in
Ça signifie	$<$	$>$	$\leq$	$\geq$	$=$	$\neq$	$\in$

Attention
Bien distinguer l'instruction d'affectation = du symbole de comparaison ==.

Pour combiner des conditions complexes (par exemple $x > -2$ et $x^2 < 5$), on peut combiner des variables booléennes en utilisant les connecteurs logiques :

On écrit	`and`	`or`	`not`
Ça signifie	et	ou	non

Deux nombres de type différents (entier, à virgule flottante, etc.) sont convertis en un type commun avant de faire la comparaison. Dans tous les autres cas, deux objets de type différents sont considérés non égaux. Voici quelques exemples :

```
>>> a = 2     # Integer
>>> b = 1.99  # Floating
>>> c = '2'   # String
```

```
>>> print(a>b)
True
>>> print(a==c)
False
```

```
>>> print( (a>b) and (a==c) )
False
>>> print( (a>b) or (a==c) )
True
```

Remarque
Dans une condition formée de la conjonction (avec `and`) de plusieurs prédicats, dès qu'un prédicat est faux, les suivants ne sont même pas évalués. L'ordre des prédicats est donc important.

Remarque
Nous utiliserons les opérateurs `and` et `or` seulement avec des booléens mais attention, ils peuvent être utilisés avec n'importe quel objets.

- `and` renvoie le premier élément s'il est `False` ou `0` ou `None` ou `""` etc., sinon il renvoie le deuxième ;
- `or` renvoie le premier élément s'il est `True` ou `!=0` ou non vide etc., sinon il renvoie le deuxième.

```
>>> print( 1 and 2 )
2
```

```
>>> print( 0 and 2 )
0
```

```
>>> print( 1 or 2 )
1
```

```
>>> print( 0 or 2 )
2
```

1.9. Les chaînes de caractères (String) et la fonction `print`

Une *chaîne de caractères* est une séquence de caractères entre guillemets (doubles `"..."` ou simples `'...'`). On peut déclarer une chaîne de caractères de trois manières :

```
s = "Topolino et Minnie"
s = 'Topolino et Minnie'
s = """Topolino et Minnie"""
```

La dernière permet d'avoir des chaînes sur plusieurs lignes. On utilisera le plus souvent la première (notamment si on doit écrire une chaîne contenant des apostrophe car, si la chaîne contient un apostrophe, python considérera l'apostrophe comme la fin de la chaîne).

Quand on utilise des nombres dans une variable, il faut juste se rappeler que `'912'` ou `"912"` sont des chaînes simplement parce qu'ils sont entourés par des guillemets, alors que `912` est un nombre entier.

```
>>> type('912'), type("912"), type(912)
(<class 'str'>, <class 'str'>, <class 'int'>)
```

- `print`

  Pour afficher à l'écran des objets on utilise la fonction `print(object)` [8] qui **convertit** `object` en une chaîne de caractères et l'affiche :

  ```
  print('Ciao') # 'Ciao' est une chaîne de caractères
  print(2022) # 2022 est un entier converti en string par print
  ```

  ```
  Ciao
  2022
  ```

  Noter que la fonction `print()` affiche l'argument qu'on lui passe entre parenthèses **et ajoute un retour à ligne**. Si on ne veut pas afficher ce retour à la ligne, on peut utiliser l'argument par «mot-clé» `end` :

  ```
  print('Ciao',end="")
  print(2022)
  ```

  ```
  Ciao2022
  ```

- **Tabulations**

  On peut forcer la tabulation (pour aligner des nombres par exemple) par le caractère `\t` :

  ```
  s = "Coucou!Je suis là.\tEt là!"
  print(s)
  ```

  ```
  Coucou!Je suis là.→Et là!
  ```

- **Retours à la ligne dans la sortie**

  Les chaînes de caractères peuvent s'étendre sur plusieurs lignes.

  - Si on utilise les guillemets simples ou doubles, le retour à la ligne peut être forcé par le caractère `\n` :

    ```
    s = "Coucou!\nJe suis là."
    print(s)
    ```

    ```
    Coucou!
    Je suis là.
    ```

  - Si on utilise les triples guillemets, simples `'''...'''` ou doubles `"""..."""`, les retours à la ligne sont automatiquement inclus, mais on peut l'empêcher en ajoutant `\` à la fin de la ligne.

8. En passant de Python 2 à Python 3, la commande `print` est devenue une **fonction**. Si en Python 2 on écrivait `print "bonjour"` en Python 3 il faut maintenant écrire `print("bonjour")`.

```
>>> s = """
... Coucou!
... Je suis là.
... """
>>> print(s)

Coucou!
Je suis là.
```

```
>>> s = """\
... Coucou!\
... Je suis là.\
... """
>>> print(s)
Coucou!Je suis là.
```

- **Concaténations (=assemblages) et conversions**

  Assembler plusieurs `strings` ensemble est une des opérations les plus courantes : on appelle cette opération une concaténation. L'opérateur + concatène deux chaînes : `s1+s2` joint la chaîne `s1` à la chaîne `s2`. Attention à ne pas oublier les espaces lors des concaténations :[9]

```
>>> string1 = 'Press return to'
>>> string2 = 'exit the program !'

>>> s = string1 + string2 # il manque une espace
>>> print(s)
Press return toexit the program !

>>> s = string1 + " " + string2 # on ajoute l'espace manquante
>>> print(s)
Press return to exit the program !
```

  On ne peut cependant pas concaténer d'autres types avec des strings (comme des variables numériques par exemple). **L'opérateur de concaténation (+) ne fonctionne qu'entre deux données de type chaîne de caractères.** Dans l'exemple suivant l'interpréteur Python lève une erreur qui signale qu'il ne parvient pas à convertir implicitement la donnée de type `int` en une donnée de type `str` :

```
>>> year = 2019
>>> s = 'Nous sommes en ' + year
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

  Pour remédier à cela, on doit *caster* la variable numérique en `string` via la fonction `str()` qui transforme son argument en une chaîne de caractères. On pourra ensuite la concaténer :

```
>>> year = 2019
>>> s = 'Nous sommes en ' + str(year)
>>> print(s)
Nous sommes en 2019
```

  - L'instruction `print(object1, object2,...)` convertit automatiquement chaque objet en une chaîne de caractères et les affiche sur la même ligne **séparés par des espace** :

```
print("J'ai", 7, "ans") # conversion de 7 en str et ajout d'une espace
```

```
J'ai 7 ans
```

    Si on ne veut pas afficher cette espace automatique, on peut utiliser l'argument par «mot-clé» `sep` :

```
print("J'ai", 7, "ans", sep="")
```

```
J'ai7ans
```

  - Le contrôle des espaces est toutefois plus simple si on passe directement à la fonction `print` une seule chaîne de caractères générée avant par concaténation. Dans ce cas il ne faut pas oublier de convertir en string explicitement les objets qui ne le sont pas :

```
s = "J'ai " + str(7) + " ans"
print(s)
```

```
J'ai 7 ans
```

    Une façon très simple de produire cette unique chaîne avant de la passer à la fonction `print` sans se soucier des conversions est l'utilisation des `f-string` décrites à la page 25.

9. On vient de voir que l'opérateur "+" peut avoir différents buts en fonction des types de variables qu'on manipule :
   - avec des types numériques, il sert à additionner ;
   - avec des chaînes de caractères, il sert à concaténer.

Opérations, fonctions et méthodes Voici quelques autres opérations, fonctions et méthodes très courantes associées aux chaîne de caractères :

Opérations :

`s1 + s2`	concatène la chaîne `s1` à la chaîne `s2`
`s1*n`	concatène la chaîne `s1` n fois (avec n entier positif)
`"x" in s`	renvoi `True` si la chaîne `s` contient la chaîne `"x"`, `False` sinon
`"x" not in a`	renvoi `True` si la chaîne `s` ne contient pas la chaîne `"x"`, `False` sinon

Fonctions :

`str(n)`	transforme un nombre `n` en une chaîne de caractères
`len(s)`	renvoie le nombre d'éléments de la chaîne `s`

```
>>> s = 'Hello '
>>> t = 'to you'
>>> print(3*s) # Repetition
Hello Hello Hello
>>> print("H" in s)
True
>>> print("x" in s)
False
>>> print(len(t))
6
```

Méthodes :

`s.index("x")`	renvoie l'indice de la **première** occurrence de la sous-chaîne `"x"` dans la chaîne `s`
`s.count("x")`	renvoie le nombre d'occurrence de la sous-chaîne `"x"` dans la chaîne `s`
`s.replace("x","y")`	**renvoie une nouvelle chaîne** où chaque sous-chaîne `"x"` est remplacée par la sous-chaîne `"y"`
`s.lower()`	**renvoie une nouvelle chaîne** où tous les caractères de `s` sont en minuscule
`s.upper()`	**renvoie une nouvelle chaîne** où tous les caractères de `s` sont en majuscule
`s.capitalize()`	**renvoie une nouvelle chaîne** où la première lettre du premier mot de `s` est en majuscule
`s.title()`	**renvoie une nouvelle chaîne** où la première lettre de chaque mot de `s` est en majuscule
`s.split()`	renvoie une liste de chaînes (chaque mot de `s`)

```
>>> string1 = 'Minnie'
>>> string2 = 'Topolino'
>>> s3 = string1 + ' et ' + string2
>>> s4 = s3.replace('et','&')
>>> print(s3)
Minnie et Topolino
>>> print(s4)
Minnie & Topolino

>>> texte= "Une foncttttion ttttrès prattttique si vous répéttttez ttttrop les tttt"
>>> print(texte.replace("tttt","t"))
Une fonction très pratique si vous répétez trop les t

>>> s5 = s4.split(' & ')
>>> print(s5)
['Minnie', 'Topolino']

>>> s6 = ' ou '.join(s5)
>>> print(s6)
Minnie ou Topolino

>>> print(s6.center(30,'-'))
```

```
------Minnie ou Topolino------

>>> print(s6.index("n"))
2
>>> print(s6.count("n"))
3

>>> chaine = "Buon compleanno a te, buon compleanno a te, \
... buon compleanno caro Pluto, buon compleanno a te!"
>>> print(chaine.count("buon"))
3

>>> s = "Pablo Neruda Saint martin d'hères"
>>> l = s.split() # By default, splits on whitespace
>>> print(l)
['Pablo', 'Neruda', 'Saint', 'martin', "d'hères"]
>>> print(' - '.join(l))
Pablo - Neruda - Saint - martin - d'hères

>>> print( 'Pablo Neruda'.lower(), 'Pablo Neruda'.upper(), 'pablo neruda'.capitalize(),
↪ 'pablo neruda'.title() )
pablo neruda PABLO NERUDA Pablo neruda Pablo Neruda

>>> s = "J. M. Brown AND B. Mencken AND R. P. van't Rooden"
>>> print(s.split(' AND '))
['J. M. Brown', 'B. Mencken', "R. P. van't Rooden"]
```

1.9.1. f-string : écriture formatée

Pour concaténer des objets de différents types en contrôlant les espaces ou encore choisir combien de chiffres afficher (par exemple pour aligner en colonne des nombres), on pourra utiliser les `f-string`. [10]

Tous d'abord on écrit tout simplement la chaîne avec le nom des variables dont on voudra afficher le contenu. Ensuite on écrit la lettre `f` devant la chaîne puis on entoure les noms des variables par des accolades pour que chaque nom soit remplacé par la valeur. Voici un exemple :

```
n1,n2 = 8,9
s = f"Sara a {n1} ans et Lorenzo {n2}."
print(s)
# beaucoup plus simple à lire que
s =  "Sara a " + str(n1) + " ans et Lorenzo " + str(n2) + "."
print(s)
```

```
Sara a 8 ans et Lorenzo 9.
Sara a 8 ans et Lorenzo 9.
```

Comme on a déjà souligné, la fonction `print` concatène et converti directement les object en `str` mais ajoute aussi une espace. Il faut alors les supprimer pour obtenir le même résultat :

```
print( "Sara a", n1, "ans et Lorenzo", n2, "." ) # Pb !!!
print( "Sara a ", n1, " ans et Lorenzo ", n2, "." , sep="" )
```

```
Sara a 8 ans et Lorenzo 9 .
Sara a 8 ans et Lorenzo 9.
```

On peut choisir le formatage de l'affichage des nombres. Voici quelques exemples :

10. Cette commande n'est disponible qu'à partir de la version 3.6 de python. Si vous utilisez une version antérieur il faudra utiliser la commande `.format()`.

```
>>> a,n = -1234.56789, 9876

>>> print(f'a = {a}, n = {n}')
a = -1234.56789, n = 9876

>>> print(f'a = {a:g}, n = {n:g}') # choisit le format le plus approprie
a = -1234.57, n = 9876

>>> print(f'{a:.3e}') # notation scientifique
-1.235e+03

>>> print(f'{a:.2f}') # fixe le nombre de decimales, ici 2
-1234.57
>>> print(f'{a:12.2f}') # precise la longueur totale de la chaine, ici 12 avec 2 decimales
    -1234.57

>>> print(f'{123:{0}{6}}') # 6 chiffres en tout, avec des 0 pour compléter
000123

>>> print(f'{a:>12.2f}') # justifie a droite
    -1234.57
>>> print(f'{a:<12.2f}') # justifie a gauche
-1234.57
>>> print(f'{a:^12.2f}') # centre
  -1234.57

>>> print(f'{a:+.2f}') # affiche toujours le signe
-1234.57
```

Remarque (Remplissage)

```
>>> word = "Ciao"
>>> print(f"{word:=<20}")
Ciao================
>>> print(f"{word:->20}")
----------------Ciao
>>> print(f"{word:/^20}")
////////Ciao////////
```

Remarque (Self-documenting expressions)
Depuis la version Python 3.8 on peut écrire directement `f'{variable = }'` au lieu de `f'variable = {variable}'` :

```
>>> a = 3
>>> print(f'{a = }')
a = 3
```

1.9.2. Accès et modification de chaînes de caractères

Un seul caractère. On peut extraire un caractère par son indice : `s[i]` renvoie le $(i+1)^e$ caractère de la chaîne `s` (les éléments d'une chaîne sont *indexés à partir de* 0 et non de 1).

```
>>> s = "Ciao"
>>> print(s[0])
C
```

1. Si on tente d'extraire un élément avec un index dépassant la taille de la chaîne, Python renvoie un message d'erreur :
```
>>> s = 'Topolino'
>>> print(s[8])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```
On verra que, lorsqu'on utilise des tranches, les dépassements d'indices sont licites.
2. Une chaîne de caractères est un objet **immuable**, *i.e.* ses caractères ne peuvent pas être modifiés par une affectation et sa longueur est fixe. Si on essaye de modifier un caractère d'une chaîne de caractères, Python renvoie une erreur comme dans l'exemple suivant :

```
>>> s    = 'Press return to exit'
>>> s[0] = 'p'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Il faudra alors écrire

```
>>> s = 'Press return to exit'
>>> s = 'p' + s[1:]
```

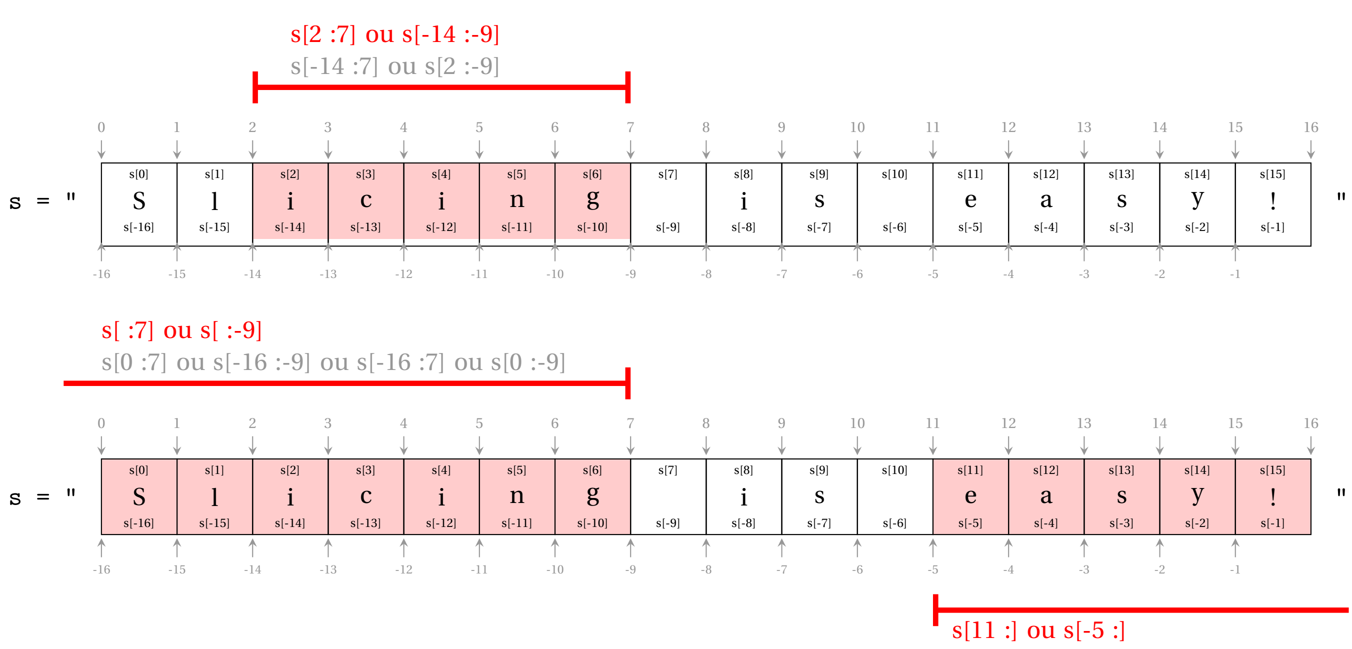
Une sous-chaînes. Soit s une chaîne de caractères. On peut extraire une sous-chaîne en déclarant

- l'indice i de **début (inclus)** et l'indice j de **fin (exclu)**, séparés par deux-points :
  `s[i:j]` équivaut à `s[i]+s[i+1]+s[i+2]+...+s[j-1]`
- l'indice i de **début (inclus)**, l'indice j de **fin (exclu)** et le **pas** k, séparés par des deux-points : `s[i:j:k]` équivaut à
  `s[i]+s[i+k]+s[i+2k]+...+s[i+mk]` avec `i+mk<j`
  Le pas peut-être négatif. Dans ce cas, il faut que j soit inférieur à i.

Cette opération est connue sous le nom de *slicing* (découpage en tranches). Voici quelques exemples pour cette notation (on suppose que la chaîne contient 9 éléments) :

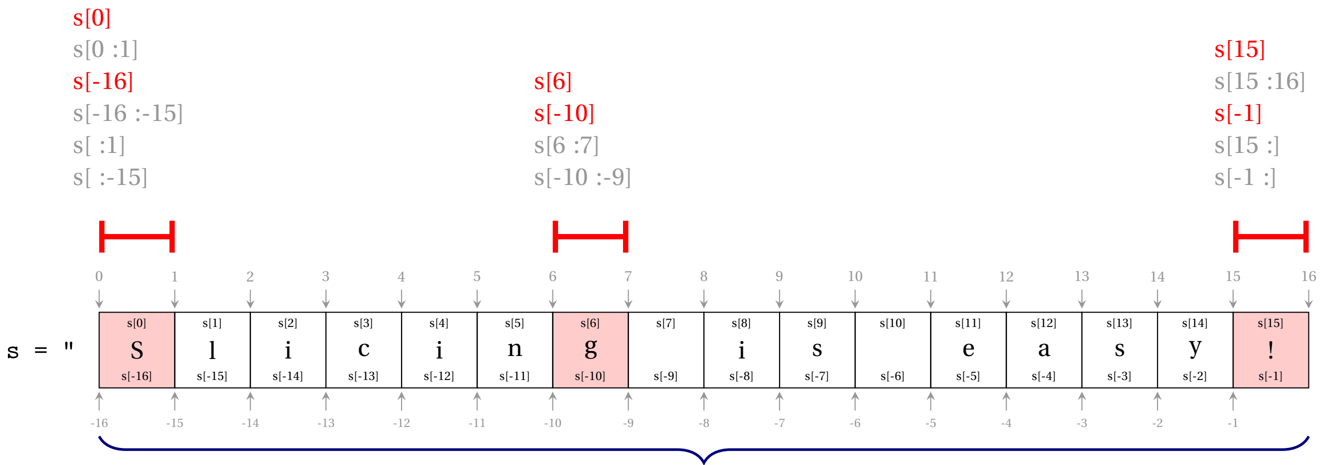
- $2:7 \rightsquigarrow 2,3,4,5,6$
- $2:7:2$ et $2:8:2 \rightsquigarrow 2,4,6$
- $2: \rightsquigarrow 2,4,\dots$ jusqu'à la fin (à droite) de la chaîne
- $:4 \rightsquigarrow 0,1,2,3$ du début (à gauche) jusqu'à 3
- $2::3 \rightsquigarrow 2,5,8$ jusqu'à la fin (à droite) de la chaîne avec un pas de 3
- $:6:2 \rightsquigarrow 0,2,4$ du début (à gauche) jusqu'à 5 avec un pas de 2
- $7:5:-1 \rightsquigarrow 7,6$
- $7:2:-2 \rightsquigarrow 7,5,3$
- $7::-1 \rightsquigarrow 7,6,5,4,3,2,1,0$
- $:3:-1 \rightsquigarrow 8,7,6,5,4$

Des petits dessins permettront de bien comprendre cette opération :



Attention aux instructions suivantes qui renvoient une chaîne vide :

```
>>> s = "Slicing is easy!"
>>> print( s[-5:0], s[11:0] )
```

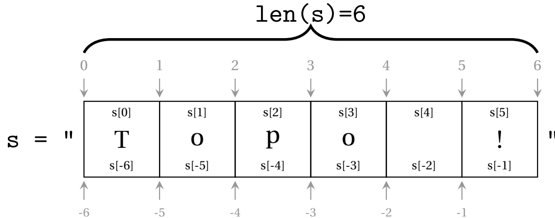


Attention à l'instruction suivante qui renvoie une chaîne vide :

```
>>> s = "Slicing is easy!"
>>> print(s[-1:0])
```

Remarque (Pourquoi la règle du premier inclus, dernier exclu ?)
Jusqu'à présent, on a appris que chaque élément d'une liste est associé à un indice ou à une position. Cependant, en Python, chaque élément est en fait considéré entre deux positions, comme le montrent les flèches numérotées dans les exemples précédents. En utilisant cette représentation, nous pouvons voir que dans ra a les éléments sont ceux compris entre la tranche 2 et la tranche 7 donc les éléments de 2 à 6. Pour beaucoup de gens, il est plus simple de considérer que les éléments sont situés entre deux indices. Pour d'autres, considérer que les éléments ont un seul indice. Je recommande de choisir une représentation et de s'y tenir.

Un autre exemple avec indication explicite du pas (négatif ou positif, sans indication il est égale à 1).



Nous avons une chaîne de 6 éléments dont les indices vont de 0 à 5 :

```
>>> s='Topo !'

>>> print(s[0],s[1],s[2],s[3],s[4],s[5])
T o p o   !
>>> print(s[-6],s[-5],s[-4],s[-3],s[-2],s[-1])
T o p o   !

>>> print(s[6]) # n'existe pas !
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Avec le slicing le dépassement des indices est permis :

```
>>> print(s[2:9])  # dépassement à droite (>len(s))
po !
>>> print(s[-4:9]) # dépassement à droite (>len(s))
po !
```

 © G. Faccanoni

```
>>> print(s[-8:-3]) # dépassement à gauche (<-len(s))
Top
>>> print(s[-8:3])  # dépassement à gauche (<-len(s))
Top
```

On peut utiliser un pas différent de 1 :

```
>>> print(s[1:6:2])
oo!
>>> print(s[-5:-1:2])
oo
```

On peut utiliser un pas négatif pour parcourir la chaîne à l'envers :

```
>>> print(s[:]) # idem s, s[::], s[::1]
Topo !
>>> print(s[::-1])
! opoT
```

Attention

Il y a deux méthodes pour comprendre ce processus d'extraction de sous-chaînes : soit en considérant les indices des éléments (en prenant tous les éléments dont les indices se situent entre le début inclus et la fin exclue), soit en envisageant les bornes délimitant les éléments. Cependant, il faut être prudent lorsqu'il y a des pas négatifs.

```
>>> print(s[-1:-5:1])  # vide !


>>> print(s[-1:-5:-1]) # il contient s[-1] s[-2] s[-3] s[-4]
! op
>>> print(s[5:1:-1])   # il contient s[5] s[4] s[3] s[2]
! op

>>> print(s[:-5:-1])   # le début est s[-1] donc il contient s[-1] s[-2] s[-3] s[-4]
! op
>>> print(s[:1:-1])    # le début est s[5] donc il contient s[5] s[4] s[3] s[2]
! op

>>> print(s[-1::-1])   # fin = s[-6] donc il contient s[-1] s[-2] s[-3] s[-4] s[-5] s[-6]
! opoT
>>> print(s[5::-1])    # fin = s[0] donc il contient s[5] s[4] s[3] s[2] s[1] s[0]
! opoT
```

On remarque que l'élément correspondant au premier indice est contenu, le dernier est exclu : si on veux utiliser la notation par tranches, il faudrait décaler nos tranches à droite pour s'y retrouver ! Dans ce cas, mieux penser élément plutôt que tranche.

1.10. ☆ La fonction input

La fonction `input()` prend en argument un message (sous la forme d'une chaîne de caractères), demande à l'utilisateur d'entrer une donnée et renvoie celle-ci **sous forme d'une chaîne de caractères**. Par exemple, on écrira

```
>>> x = input("Choisi un nombre : ")
Choisi un nombre : 3
>>> print(f"Tu a choisi {x}")
Tu a choisi 3
```

Si la donnée est une valeur numérique, il faut ensuite convertir cette dernière en entier ou en float (avec la fonction `eval()` ou `int()` ou `float()`).

```
x = input("Entrer une valeur pour x : ")
# print(f"x^2 = {x**2}") # error
print(f"x^2 = {eval(x)**2}")
```

 © G. Faccanoni

1.11. Exercices

Attention

Quand on apprend à coder, il est crucial de saisir chaque commande plutôt que de céder à la tentation du copier/coller. La saisie contribue à la mémorisation des commandes de deux manières :

- Tout d'abord, lorsque nous tapons une commande, nous la prononçons mentalement, la répétons dans notre esprit et l'ancrons dans notre mémoire.
- Deuxièmement, nos doigts peuvent mémoriser des schémas de frappe. Par exemple, lors de la saisie de `print()`, nos doigts retiendront automatiquement qu'il faut taper les parenthèses juste après `print`.

De plus, à l'instar d'un pianiste qui regarde la partition plutôt que le clavier pendant qu'il joue, nous ne voulons pas regarder le clavier mais l'écran lorsque nous codons. Cette méthode de saisie est appelée la "frappe tactile" (ou frappe aveugle). Elle nous permet d'être plus rapides et de minimiser le nombre d'erreurs que nous commettons car nous n'avons pas à déplacer nos yeux entre le clavier et l'écran. Comment apprendre la "frappe tactile" ? C'est très facile : il suffit de s'entraîner.

Exercice 1.1 (Mode interactif)

Lancer le logiciel `Idle3` à partir du menu "Applications". On voit apparaître la version de python (3.8.10 dans nos salles de TP) et les trois chevrons `>>>` indiquant qu'on a lancé l'interpréteur.

Essayer les instructions suivantes et noter la priorité des instructions. Attention si les deux premières instructions ne donnent pas le même résultat, cela signifie qu'on a lancé Python2 au lieu de Python3 (Idle au lieu Idle3).

```
13/2
13/2.
```

```
42
2*12
2**10
((19.9*1.2)-5)/4
```

```
.6*9+.6
round(.6*9+.6)
```

```
13%2
13//2
divmod(13,2)
exit()
```

On constate qu'on peut utiliser l'interpréteur Python en mode interactif comme une calculatrice. En effet, si vous y tapez une expression mathématique, cette dernière est évaluée et son résultat affiché comme résultat intermédiaire (autrement dit, il n'est pas nécessaire d'utiliser `print`, ce qui n'est pas le cas avec le mode script).

Correction

```
>>> 13/2
6.5
>>> 13/2.
6.5
```

```
>>> 2*12
24
>>> 2**10
1024
>>> ((19.9*1.2)-5)/4
4.72
```

```
>>> .6*9+.6
5.999999999999999
>>> round(.6*9+.6)
6
```

```
>>> 13%2
1
>>> 13//2
6
>>> divmod(13,2)
(6, 1)
```

Exercice 1.2 (Mode script)

Le mode interactif ne permet pas de développer des programmes complexes : à chaque utilisation il faut réécrire le programme. La modification d'une ligne oblige à réécrire toutes les lignes qui la suivent. Pour développer un programme plus complexe on saisit son code dans un fichier texte : plus besoin de tout retaper pour modifier une ligne ni de tout réécrire à chaque lancement. Le programme s'écrit dans un fichier texte que l'on sauvegarde avec l'extension `.py`. On peut modifier le programme aisément, en rajoutant des commentaires, des espaces et de nouvelles lignes d'instruction. Vous pouvez utiliser de nombreux éditeurs de texte, mais vous pouvez commencer avec l'éditeur de texte intégré à Idle.

Lancer le logiciel `Idle3`. Ouvrir l'éditeur de texte intégré : menu "File", puis "New File". Y écrire les lignes suivantes

```
a = 5
b = 6
c = a+b
```

```
print("c =",c)
```

Sauvegarder le fichier (par exemple sous le nom `exo_1_2.py`) puis appuyer sur la touche F5 qui execute le fichier.

Correction
On obtient c = 11)

Exercice Bonus 1.3 (Rendre un script exécutable : la ligne *shebang*)

1. Ouvrir le fichier `first.py` dans un éditeur de texte pur.
2. Modifier ce fichier en ajoutant comme premières lignes :

```
#! /usr/bin/env python3
# coding: utf-8
```

3. Sauvegarder le fichier.
4. Ouvrir un terminal et y écrire `chmod +x first.py` puis appuyer sur la touche Enter .
5. Dans le même terminal écrire `./first.py` puis appuyer sur la touche Enter . [Il n'est plus nécessaire d'écrire `python3 first.py`]

Correction
Il est fortement conseillé d'ajouter deux lignes en haut de chacun de vos scripts :

```
#! /usr/bin/env python3
# coding: utf-8
```

Ces lignes sont des commentaires pour Python mais peuvent également contenir des instructions systèmes.

Le commentaire `#! /usr/bin/env python3` indique que ce script doit être exécuté à l'aide de Python 3. Cela permet au système d'exploitation de connaître le chemin d'accès vers l'interpréteur Python. Sans cette ligne, vous pouvez rencontrer des problèmes lors de l'exécution du script (plusieurs versions de Python peuvent coexister). Cette première ligne s'appelle un *Shebang* dans le jargon Unix. Unix stipule que le *Shebang* doit être en première position dans le fichier.

Le commentaire `# coding: utf-8` spécifie l'encodage du code source de notre script. Afin de prendre en compte les accents de la langue française, nous utilisons le très commun `utf-8`.

Attention

① À partir de ce moment, pour chaque exercice **on écrira les instructions dans un fichier script** nommé par exemple `exo_1_11.py` (sans espaces et sans points sauf pour l'extension .py). On pourra bien-sûr utiliser le mode interactif pour simplement vérifier une commande mais chaque exercice devra in fine être résolu dans un fichier de script. Ne pas oublier la différence entre l'output produit en mode *interactif* et l'output produit en mode *script* (*cf.* page 14) ; dans le doute, utiliser toujours la fonction `print`).

② La version de Python qui a servi de référence pour le polycopié est la version 3.10. . Les `f-strings` n'ayant été introduites qu'en Python-3.6, si vous exécutez les lignes suivantes

```
>>> age = 10
>>> print(f"Jean a {age} ans")
Jean a 10 ans
```

avec un version antérieure à la 3.6 vous verrez ceci

```
age = 10
print(f"Jean a {age} ans")
File "<stdin>", line 1
f"Jean a {age} ans"
^
SyntaxError: invalid syntax
```

Il faut remplacer ce code avec la méthode `format` :

```
>>> age = 10
>>> print("Jean a {} ans".format(age))
```

```
Jean a 10 ans
```

ou, plus simplement, par

```
>>> age = 10
>>> print("Jean a",age,"ans")
Jean a 10 ans
```

Exercice 1.4 (Devine le résultat – affectations)

Prédire le résultat de chacune des instructions suivantes, puis vérifier-le dans l'interpréteur Python :

```
a = 1
b = 100
b = a
a = b
print(f"a = {a} b = {b}")
```

```
a = 1
b = 100
a = b
b = a
print(f"a = {a} b = {b}")
```

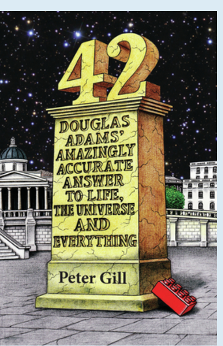
Correction
Le premier script donne a = 1 b = 1 , tandis que le deuxième a = 100 b = 100

Pour visualiser le contenu de chaque variable pas à pas on peut utiliser pythontutor.com

Exercice 1.5 (Devine le résultat – affectations)

Prédire le résultat de chacune des instructions suivantes, puis vérifier-le dans l'interpréteur Python :

```
Hitchhiker, Guide, to, the, Galaxy = 0, 1, 2, 3, 4
The = the-to
Answer = Galaxy**0.5
to = to-1
the = The
Ultimate = 3*Guide
Question = 0*Hitchhiker + 0*Guide + 0*to + 0*the + Galaxy / 4
of = Galaxy**Hitchhiker
Life = Galaxy*the + Answer*Guide + Ultimate - 2
number = The*Answer*to*the*Ultimate*Question*of*Life
```



Correction
Visualiser pas à pas sur pythontutor.com

```
Hitchhiker, Guide, to, the, Galaxy = 0, 1, 2, 3, 4
The = the-to           # 1
Answer = Galaxy**0.5   # 2
to = to-1              # 1
the = The              # 1
Ultimate = 3*Guide     # 3
Question = 0*Hitchhiker + 0*Guide + 0*to + 0*the + Galaxy / 4  # 1
of = Galaxy**Hitchhiker  # 4^0 = 1
Life = Galaxy*the + Answer*Guide + Ultimate - 2  # 4×1+2×1+3−2=7
number = The*Answer*to*the*Ultimate*Question*of*Life  # 1×2×1×1×3×1×1×7=42
print(number)
```

```
42.0
```

Exercice 1.6 (Devine le résultat – échanges)

Prédire le résultat de chacune des instructions suivantes, puis vérifier-le dans l'interpréteur Python :

```
a = 1; b = 100;
t = b
b = a
a = t
print(f"a = {a} b = {b}")
```

```
a = 1; b = 100;
a,b = b,a
print(f"a = {a} b = {b}")
```

```
a = 1; b = 100;
a = a+b
b = a-b
a = a-b
print(f"a = {a} b = {b}")
```

Correction
Le premier script donne a = 100 b = 1 , le deuxième a = 100 b = 1 et le troisième a = 100 b = 1 Remarque : on peut écrire `a += b` au lieu de `a = a+b`.

Exercice 1.7 (Devine le résultat – logique)

Quel résultat donnent les codes suivants ?

Cas 1
```
x = 3
print( x == 3 )
print( x != 3 )
print( x >= 4 )
print( not(x<4) )
```

Cas 2
```
a = 7
print( a>5 and a<10 )
print( 5<a<10 )
```

Cas 3
```
a = 15
print( a>5 or a<10 )
print( a<5 or a>10 )
```

Cas 4
```
a, b, c = 1, 10, 100
print( a<b<c )
print( a>b>c )
```

Correction
• Cas 1 : True False False False • Cas 2 : True True • Cas 3 : True True • Cas 4 : True False

Exercice 1.8 (Séquentialité)

Écrire un script qui ne contient que les lignes suivantes après les avoir remises dans l'ordre de sorte qu'à la fin x ait la valeur 46. Et si on veut que x vaut 49 ?

```
y = y-1     # instr. a
y = 2*x     # instr. b
print(x)    # instr. c
x = x+3*y   # instr. d
x = 7       # instr. e
```

Correction
On ne peut pas utiliser x tant que on ne l'a pas affecté, donc l'instruction e doit précéder les instructions b, c et d. Pour la même raison, l'instruction b doit précéder les instructions a et d. Le code

```
x = 7       # x ← 7
y = 2*x     # y ← 2x = 14
y = y-1     # y ← y−1 = 14−1 = 13 (on peut réécrire l'instruction y-=1)
x = x+3*y   # x ← x+3y = 7+3×13 = 46 (on peut réécrire l'instruction x+=3*y)
print(x) # on affiche 46
```

affichera

```
46
```

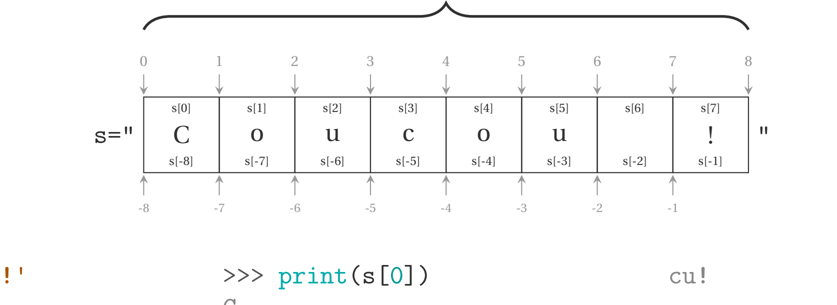
Pour obtenir 49 on enlève l'instruction `y=y-1`.

Exercice 1.9 (Devine le résultat – `string`)

Quel résultat donne le code suivant ? Si une instruction lève une erreur, expliquer pourquoi et proposer une correction.

```
s = 'Coucou !'
print(s)
print(s*2)
print(s+" TEST")
print(s[0])
print(s[0:])
print(s[3:])
print(s[3::1])
print(s[3::2])
print(s[::-1])
print(s[3:0:-1]) #!
print(s[7:0:-1]) #!
s[0] = 'T'
```

Correction




```
>>> s = 'Coucou !'

>>> print(s)
Coucou !
>>> print(s*2)
Coucou !Coucou !
>>> print(s+" TEST")
Coucou ! TEST
>>> print(s[0])
C
>>> print(s[0:])
Coucou !
>>> print(s[3:])
cou !
>>> print(s[3::1])
cou !
>>> print(s[3::2])
cu!
>>> print(s[3:0:-1])
cuo
>>> print(s[7:0:-1])
! uocuo
>>> print(s[::-1])
! uocuoC
```

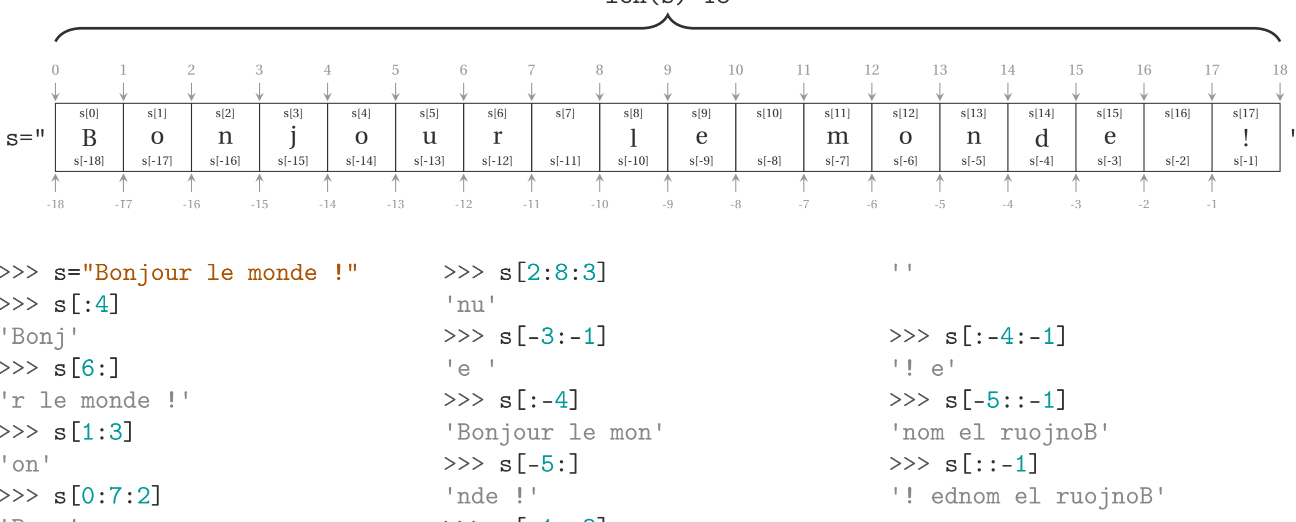
La ligne `s[0]='T'` lève une erreur car on ne peut pas modifier une chaîne. On pourra utiliser l'instruction `s='T'+s[1:]`.

Exercice 1.10 (Sous-chaînes de caractères)

On considère la chaîne de caractères `s="Bonjour le monde !"`. Déterminer les sous-chaînes suivantes : `s[:4]`, `s[6:]`, `s[1:3]`, `s[0:7:2]`, `s[2:8:3]`, `s[-3:-1]`, `s[:-4]`, `s[-5:]`, `s[-1:-3]`, `s[:-4:-1]`, `s[-5::-1]`, `s[::-1]`.

Rappel : en mode interactif il n'est pas nécessaire d'utiliser la fonction `print` (*cf.* page 14).

Correction




```
>>> s="Bonjour le monde !"
>>> s[:4]
'Bonj'
>>> s[6:]
'r le monde !'
>>> s[1:3]
'on'
>>> s[0:7:2]
'Bnor'
>>> s[2:8:3]
'nu'
>>> s[-3:-1]
'e '
>>> s[:-4]
'Bonjour le mon'
>>> s[-5:]
'nde !'
>>> s[-1:-3]
''

>>> s[:-4:-1]
'! e'
>>> s[-5::-1]
'nom el ruojnoB'
>>> s[::-1]
'! ednom el ruojnoB'
```

Exercice 1.11 (Devine le résultat – ASCII art)

```
print("/\_/\ \n>^.^< \n / \ \n(___)__/")
```

Correction

```
/\_/\
>^.^<
 / \
(___)__/
```

Exercice 1.12 (Devine le résultat – opérations et conversions de types)

Prédire le résultat de chacune des instructions suivantes, puis vérifier-le dans l'interpréteur Python. Attention : une instruction lève une erreur. Expliquer pourquoi.

```
print( str(4) * int("3") )
print( int("3") + float("3.2") )
print( str(3) * float("3.2") )
print( str(3/4) * 2 )
print( 3/4 * 2 )
```

Correction

```
>>> print( str(4) * int("3") ) # idem que "4"*3
444
>>> print( int("3") + float("3.2") )  # idem que 3*3.2
6.2
>>> print( str(3) * float("3.2") ) # idem que "3"*3.2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
>>> print( str(3/4) * 2 ) # idem que "0.75"*3.2
0.750.75
>>> print( 3/4 * 2 ) # idem que 0.75*3.2
1.5
```

Exercice 1.13 (Happy Birthday)

Soit les chaînes de caractères suivantes :

```
h = 'Happy birthday'
t = 'to you'
p = 'Prenom'
```

Imprimer la chanson *Happy birthday* par **concatenation** de ces chaînes (aller à la ligne à la fin de chaque couplet mais utiliser une seule instruction `print`). Le résultat devra être le suivant (bien noter les espaces et les retours à la ligne). On utilisera des *f-string*.

```
Happy birthday to you
Happy birthday to you
Happy birthday Prenom
Happy birthday to you
```

Correction

```
h = 'Happy birthday'
t = 'to you'
p = 'Prenom'
print(f"{h} {t}\n{h} {t}\n{h} {p}\n{h} {t}")
```

```
# idem que
# couplet_1 = h+' '+t+'\n'
# couplet_2 = h+' '+p+'\n'
# print(couplet_1*2+couplet_2+couplet_1)
```

 © G. Faccanoni

```
Happy birthday to you
Happy birthday to you
Happy birthday Prenom
Happy birthday to you
```

Exercice 1.14 (Compter le nombre de caractères blancs)
Écrire un script qui compte le nombre de caractères blancs `" "` contenus dans une chaîne. Par exemple, si la chaîne de caractères est `s="Bonjour le monde !"`, on devra obtenir 3.

Correction
On peut utiliser la méthode `count` comme suit :

```
s = "Bonjour le monde !"
n = s.count(" ")
print(f"La chaîne '{s}' contient {n} caractères blancs.")
# print("La chaîne '{}' contient {} caractères blancs.".format(s,n))
# print("La chaîne '" + s + "' contient " + str(n) + " caractères blancs.")
# print("La chaîne '", s, "' contient", n, "caractères blancs.") # Pb espaces
```

```
La chaîne 'Bonjour le monde !' contient 3 caractères blancs.
```

Exercice 1.15 (String + et *)
Que vaut l'expression suivante ?

```
len("lo"+("la"*5+" ")*4)
```

Correction
`len` renvoie le nombre de caractères d'une chaîne :

- `"la"*5` ⇝ une chaîne de $2 \times 5 = 10$ caractères
- `"la"*5+" "` ⇝ une chaîne de 11 caractères
- `("la"*5+" ")*4` ⇝ une chaîne de $11 \times 4 = 44$ caractères
- `("la"*5+" ")*4+"lo"` ⇝ une chaîne de 46 caractères

```
>>> len("lo"+("la"*5+" ")*4)
46
```

En effet :

```
>>> "lo"+("la"*5+" ")*4
'lolalalalala lalalalala lalalalala lalalalala '
```

Exercice 1.16 (String concatenation)
Considérons les affectations

```
a = "six"
b = a
c = " > "
d = "ty"
e = "1"
```

Modifier les variables en utilisant exclusivement les valeurs de `a`, `b`, `c`, `d`, `e` de sort à ce que l'expression `a+c+e+b` affiche `sixty > 11 > six`.

Correction

```
a = "six"
b = a
c = " > "
d = "ty"
```

```
e = "1"
a += d # a -> 'sixty'
e *= 2 # equivalente a e = 2*e, e -> '11'
e += c # e -> '11 > '
print(a + c + e + b)
```

```
sixty > 11 > six
```

Exercice 1.17 (Calculer l'age)

Affecter la variable `year` avec l'année courante et `birthyear` avec l'année de votre naissance. Afficher la phrase "Né en xxxx, j'ai xx ans." où xx sera calculé automatiquement.

Correction

Pour composer la chaîne avec une f-string :

- on écrit d'abord la phrase en français en respectant la ponctuation et en écrivant le nom des variables :

  `"Né en birthyear, j'ai age ans."`

- on ajoute ensuite un `f` avant les guillemets et on entoure les variables dont on veut afficher la valeur par des **accolades** :

  `f"Né en {birthyear}, j'ai {age} ans."`

Comparons la simplicité des f-string avec les autres méthodes :

```
year = 2021
birthyear = 2000
age = year - birthyear

s1 = f"Né en {birthyear}, j'ai {age} ans" # f-string, si python > 3.6
# s1 = "Né en {}, j'ai {} ans".format(birthyear, age)
s2 = 'Né en ' + str(birthyear) + ", j'ai " + str(age) + ' ans.' # concatenation + conversion
print(s1)
print(s2)

print('Né en', birthyear, ", j'ai", age, 'ans.') # Pb
print('Né en ', birthyear, ", j'ai ", age, ' ans.', sep="")
```

```
Né en 2000, j'ai 21 ans
Né en 2000, j'ai 21 ans.
Né en 2000 , j'ai 21 ans.
Né en 2000, j'ai 21 ans.
```

Noter qu'il faut utiliser `"` au lieu de `'` lorsqu'on a un apostrophe dans la chaîne à afficher.

Exercice 1.18 (Nombre de chiffres de l'écriture d'un entier)

Pour $n \in \mathbb{N}$ donné, calculer le nombre de chiffres qui le composent.

Correction

1. Première idée : on transforme le nombre en chaîne de caractères (fonction `str`) et on compte la longueur de la chaîne (fonction `len`) puis on affiche le résultat (fonction `print`) :

   ```
   >>> n = 123456
   >>> print(len(str(n)))
   6
   ```

2. Deuxième idée : en se rappelant que $\log_{10}(10^k) = k$ pour $k \in \mathbb{N}$ et que $\log_{10}$ est une fonction croissante, on a

$$\underbrace{\log_{10}(10^\ell)}_{=\ell} \le \log_{10}(n) \le \underbrace{\log_{10}(10^{\ell+1})}_{=\ell+1}$$

où $\ell = E(\log_{10}(n))$ est la partie entière de $\log_{10}(n)$ et $()\ell + 1)$ le nombre de chiffres de n (pour utiliser les fonctions $\log_{10}$ et E il faut importer le module `math` (voir à la page 218) :

```
>>> import math
>>> n = 123456
>>> print(1+int(math.log(n,10)))
6
```

3. Troisième idée : on pense "base 10" et on divise par 10 (division entière) jusqu'à ce que le nombre est supérieur ou égale à 10 (on verra au chapitre 3 la syntaxe d'une disjonction de cas et au chapitre 6 la syntaxe d'une fonction lambda, ici récursive)

```
NbChiffres = lambda n : 1 if n<10 else 1+NbChiffres(n//10)
n = 123456
print(NbChiffres(n))
```

```
6
```

Exercice 1.19 (Chaîne de caractères palindrome)

Une chaîne de caractères est dite "palindrome" si elle se lit de la même façon de gauche à droite et de droite à gauche. Par exemple : `"a reveler mon nom mon nom relevera"` est palindrome (en ayant enlevé les accents, les virgules et les espaces blancs). Le plus long mot palindrome de la langue française est "ressasser".

Pour une chaîne donné, afficher `True` ou `False` selon que la chaîne de caractères est palindrome ou pas.

Correction

```
>>> s = "bob"
>>> print(s==s[::-1])
True

>>> s = 'banana'
>>> s == s[::-1]
False
```

```
>>> s = "a reveler mon nom mon nom relevera"
>>> s = s.replace(" ","")
>>> print(s)
arevelermonnommonnomrelevera
>>> print(s==s[::-1])
True
```

Exercice 1.20 (Nombre palindrome)

Un nombre est dit "palindrome" s'il se lit de la même façon de gauche à droite et de droite à gauche. Par exemple 12321 est palindrome.

Pour $n \in \mathbb{N}$ donné, afficher `True` ou `False` selon que le nombre est palindrome ou pas. Nota bene : ne pas écrire `n="12321"`.

Correction

Première méthode : on transforme le nombre en chaîne de caractères, on la retourne et on compare.

```
>>> n = 12321
>>> s = str(n)
>>> print(s==s[::-1])
True
```

```
>>> n = 123210
>>> s = str(n)
>>> print(s==s[::-1])
False
```

Deuxième méthode (en utilisant une boucle `while`, cf. chapitre 4) : on extrait les chiffres une par une en utilisant la division par 10 et on crée petit à petit le nouveau nombre puis on les compare.

```
>>> n = 12321
>>> tmp = n
>>> rev_n = 0
>>> while tmp > 0:
...     tmp,digit = divmod(tmp,10)
...     rev_n = rev_n * 10 + digit
...
>>> print( n==rev_n )
True
```

```
>>> n = 123210
>>> tmp = n
>>> rev_n = 0
>>> while tmp > 0:
...     tmp,digit = divmod(tmp,10)
...     rev_n = rev_n * 10 + digit
...
>>> print( n==rev_n )
False
```

Exercice 1.21 (Conversion h/m/s ⇝ s)

Affecter les variables heures, minutes et secondes. Calculer le nombre de secondes correspondants.

Exemple de output pour `hour, minutes, secondes = 1,1,30` :

"1 h 1 m 30 s correspondent à 3690 secondes "

Correction

Attention à ne pas utiliser le nom `min` pour les minutes car `min` est une fonction prédéfinie :

```
hour, minutes, secondes = 1,1,30
secTOT = hour*60*60+minutes*60+secondes
print(f"{hour} h {minutes} m {secondes} s correspondent à {secTOT} secondes")
```

Exercice 1.22 (Conversion h/m/s ⇝ s — *cf.* cours N. Meloni)

Affecter à la variable `secTOT` un nombre de secondes. Calculer le nombre d'heures, de minutes et de secondes correspondants.

Exemple de output pour `secTOT=12546` :

"12546 secondes correspondent à 3 h 29 m 6 s "

Correction

Attention à ne pas utiliser le nom `min` pour les minutes car `min` est une fonction prédéfinie :

```
secTOT = 12546 # 3*60*60 + 29*60 + 6
hour, secTOT = divmod(secTOT, 60*60)
minutes, sec = divmod(secTOT,60)
print(f"{secTOT} secondes correspondent à {hour} h {minutes} m {sec} s")
```

☆ Exercice Bonus 1.23 (Format table)

En utilisant les affectations

```
heading = '| Index of Dutch Tulip Prices |'
line = '+' + '-'*16 + '-'*13 + '+'
```

reproduire l'output

```
+-----------------------------+
| Index of Dutch Tulip Prices |
+-----------------------------+
|    Nov 23 1636 |        100 |
|    Nov 25 1636 |        673 |
|    Feb  1 1637 |       1366 |
+-----------------------------+
```

Correction

```
print(line,
heading,
line,
'|    Nov 23 1636 |        100 |',
'|    Nov 25 1636 |        673 |',
'|    Feb  1 1637 |       1366 |',
line,
sep='\n')
```

```
+-----------------------------+
| Index of Dutch Tulip Prices |
+-----------------------------+
|    Nov 23 1636 |        100 |
|    Nov 25 1636 |        673 |
|    Feb  1 1637 |       1366 |
+-----------------------------+
```

Source : https://scipython.com/book/chapter-2-the-core-python-language-i/examples/a-simple-text-table-using-python-strings/

☆ Exercice Bonus 1.24 (Tables de vérité)

La méthode des tables de vérité est une méthode élémentaire pour tester la validité d'une formule du calcul propositionnel. Les énoncés étant composés à partir des connecteurs «non», «et», «ou», «si…, alors», «si et seulement si», notés respectivement $\neg$, $\land$, $\lor$, $\implies$, $\iff$, les fonctions de vérités du calcul propositionnel classique sont données par la table de vérité suivante :

P	Q	$\neg$(P)	$\neg$(Q)	$P\land Q$	$P\lor Q$	$P\implies Q$	$Q\implies P$	$P\iff Q$
F	F	V	V	F	F	V	V	V
F	V	V	F	F	V	V	F	F
V	F	F	V	F	V	F	V	F
V	V	F	F	V	V	V	V	V

Générer cette table automatiquement avec python.

Correction

On peut écrire explicitement chaque ligne du tableau bien-sûr, sinon, lorsqu'on aura vu les boucles, on pourra écrire :

```
print(f'P\t Q\t non P\t non Q\t P et Q\t P ou Q\t P => Q\t Q => P\t P ssi Q')
for P in [False,True]:
	for Q in [False,True]:		
		print(f'''{P}\t {Q}\t {not P}\t {not Q}\t {P and Q}\t {P or Q}\t {(not P) or Q}\t\
{(not Q) or P}\t {((not P) or Q) and ((not P) or Q)}''')
```

```
P	 Q	 non P	 non Q	 P et Q	 P ou Q	 P => Q	 Q => P	 P ssi Q
False	 False	 True	 True	 False	 False	 True	True	 True
False	 True	 True	 False	 False	 True	 True	False	 True
True	 False	 False	 True	 False	 True	 False	True	 False
True	 True	 False	 False	 True	 True	 True	True	 True
```

Bonus : lorsqu'on aura introduit les listes, les boucles et les modules, on pourra utiliser le module `tabulate` qui prend en entrée une liste :

```
from tabulate import tabulate
L = []
L.append(['P', 'Q', 'non P', 'non Q', 'P et Q', 'P ou Q', 'P => Q', 'Q => P', 'P ssi Q'])
for P in [False,True]:
	for Q in [False,True]:		
		L.append([P,Q,not P,not Q,P and Q,P or Q,(not P) or Q,(not Q) or P,((not P) or Q) and
		 ↪ ((not P) or Q)])
```

```
print(tabulate(L, headers='firstrow'))
```

```
P      Q      non P    non Q    P et Q    P ou Q    P => Q    Q => P    P ssi Q
-----  -----  -------  -------  --------  --------  --------  --------  ---------
False  False  True     True     False     False     True      True      True
False  True   True     False    False     True      True      False     True
True   False  False    True     False     True      False     True      False
True   True   False    False    True      True      True      True      True
```

Exercice Bonus 1.25 (Années martiennes — *cf.* cours N. Meloni)
Affecter à la variable aT un nombre d'années terrestres. Calculer le nombre de jours et d'années martiens correspondants sachant que

- 1 an terrestre = 365.25 jours terrestres
- 1 jour terrestre = 86400 secondes
- 1 jour martien = 88775.244147 secondes
- 1 an martien = 668.5991 jours martiens

On arrondira les valeurs à l'entier le plus proche en utilisant la fonction `round(x)` (*e.g.* `round(3.7)` vaut 4).

Exemple de output pour `aT=36.5` : "36.5 an(s) terrestre(s) correspond(ent) à 19 an(s) et 272 jour(s) martiens "

Correction
On ne peut pas utiliser `divmod` car ici on ne travaille pas avec des diviseurs entiers.

```
aT = 36.5
jT = aT*365.25
s  = jT*86400
jM = s/88775.244147
aM = round(jM/668.5991)
print(f"""{aT} an(s) terrestre(s) correspond(ent) à \
{aM} an(s) et {round(jM-(aM*668.599))} jour(s) martiens""")
```

Exercice Bonus 1.26 (Changement de casse & Co.)
Avec `s = "Un EXEMPLE de Chaîne de Caractères"`, que donneront les commandes suivantes ? Notez bien que s n'est pas modifiée, c'est une nouvelle chaîne qui est créée. `s.lower()`, `s.upper()`, `s.capitalize()`, `s.title()`, `s.swapcase()`, `s.center(50)`.

Correction

```
>>> s = "Un EXEMPLE de Chaîne de Caractères"
>>> s.lower() # mise en minuscules
'un exemple de chaîne de caractères'
>>> s.upper() # mise en majuscules
'UN EXEMPLE DE CHAÎNE DE CARACTÈRES'
>>> s.capitalize() # mise en majuscule de l'initiale
'Un exemple de chaîne de caractères'
>>> s.title() # mise en majuscule de l'initiale de chaque mot
'Un Exemple De Chaîne De Caractères'
>>> s.swapcase()
'uN exemple DE cHAÎNE DE cARACTÈRES'
>>> s.center(len(s)+3)
'  Un EXEMPLE de Chaîne de Caractères '
```

Exercice Bonus 1.27 (Comptage, recherche et remplacement)
Avec `s = "Monsieur Jack, vous dactylographiez bien mieux que votre ami Wolf"`, que donneront les commandes suivantes ? Une instruction lève une erreur, pourquoi ?

```
len(s) ; s.count('e') ; s.count('ie')
```

```
s.find('i') ; s.find('i',40) ; s.find('i',20,30) ; s.rfind('i')
s.index('i') ; s.index('i',40) ; s.index('i',20,30) ; s.rindex('i')
"ab;.bc;.cd".replace(';.','-'); " abcABC       ".strip();
```

Correction

```
>>> s = "Monsieur Jack, vous dactylographiez bien mieux que votre ami Wolf"
>>> len(s)
65
>>> s.count('e') # nombre de 'e'
6
>>> s.count('ie') # nombre de 'ie'
4
>>> s.find('i')  # place du premier 'i' rencontré (-1 si absent)
4
>>> s.find('i',40) # place du premier 'i' rencontré à partir de la position 40
42
>>> s.find('i',20,30) # place du premier 'i' entre 20 inclus et 30 exclu
-1
>>> s.rfind('i') # comme find, mais à rebours
59
>>> s.index('i') # comme "find" mais erreur si absent
4
>>> s.index('i',40)
42
>>> s.index('i',20,30)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> s.rindex('i') # comme index, mais à rebours
59
>>> "ab;.bc;.cd".replace(';.','-')
'ab-bc-cd'
>>> " abcABC       ".strip()
'abcABC'
```

Exercice 1.28 (Devine le résultat – Yoda)

Quel résultat donne le code suivant ?

```
txt = "Tu dois redouter le côté obscur de la Force"
# txt = "Nous sommes en danger"
# txt = "Il est ton père"

mo = txt.split(' ')
pr = mo[0].lower()
ve = mo[1]
r = ' '.join(mo[2:])
re = r[0].upper()+r[1:]
sol = ' '.join([re,pr,ve])
print(sol)
```

Correction

```
Redouter le côté obscur de la Force tu dois
```

La formation des phrases est une résultante d'une figure stylistique appelée anastrophe.

CHAPITRE 2

Structures de référence : Listes, Tuples, Dictionnaires et Ensembles

Python propose différents structures pour stocker des éléments :

- `list` : tableau d'éléments **indexés de** 0 **à** n (exclu) **modifiable** (on peut ajouter ou retirer des éléments) ;
- `tuple` : tableau d'éléments **indexés de** 0 **à** n (exclu) qu'**on ne peut pas modifier** ;
- `dict` : tableau d'éléments **indexés par des types immuables** auquel on peut ajouter ou retirer des éléments ;
- `set` : tableau d'**éléments uniques** non indexés.

https://docs.python.org/fr/3/tutorial/datastructures.html

2.1. Listes

Une liste est une suite d'objets, rangés dans un certain ordre. Chaque objet est séparé par une virgule et la suite est encadrée par des crochets.

```
>>> L = [1,2,3]
>>> print(L)
[1, 2, 3]
```

```
>>> L = ["a","d","m"]
>>> print(L)
['a', 'd', 'm']
```

Une liste n'est pas forcement homogène : elle peut contenir des objets de types différents les uns des autres. On peut d'ailleurs mettre une liste dans une liste.

```
>>> L = [1,'ok',[5,10]]
>>> print(L)
[1, 'ok', [5, 10]]
```

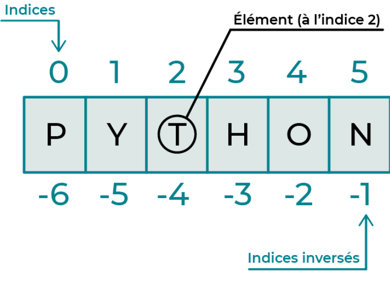
Pour définir une liste vide (qu'on remplira ensuite) on écrit

```
>>> L = []
>>> # idem que
>>> # L = list()
>>> print(L)
[]
```

La première manipulation que l'on a besoin d'effectuer sur une liste, c'est d'en extraire et/ou modifier un élément : la syntaxe est `ListName[index]`.

Comme pour les string, les éléments d'une liste sont *indexés à partir de* 0 et non de 1.

Pour la chaîne "Python" de l'image ci-contre, on doit utiliser l'indice "2" ou l'indice inversé "-4" pour accéder au troisième caractère qui est la lettre "T".



```
>>> L = [12, 10, 18, 7, 15, 3]
>>> print(L[2])
18

>>> liste = [1,'ok',[5,10]]
>>> print(liste[1])
ok
>>> print(liste[2])
[5, 10]
>>> print(liste[2][0])
5
```

On peut modifier les éléments d'une liste :

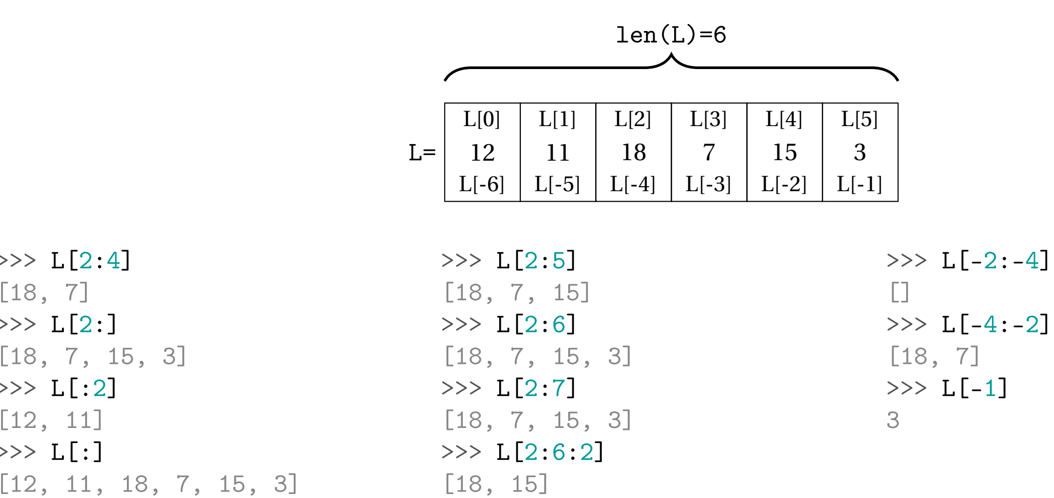
```
>>> L = [12, 10, 18, 7, 15, 3]
>>> L[1] = 11
>>> print(L)
[12, 11, 18, 7, 15, 3]
```

Si on tente d'extraire un élément avec un indice dépassant la taille de la liste, Python renvoi un message d'erreur :

```
>>> print( L[0], L[1], L[2], L[3], L[4], L[5] )
12 11 18 7 15 3
>>> print( L[6] )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Comme pour les chaînes de caractères, on peut extraire une sous-liste en déclarant l'indice de **début (inclus)** et l'indice de **fin (exclu)**, séparés par deux-points : `ListName[i:j]`, ou encore une sous-liste en déclarant l'indice de début (inclus), l'indice de fin (exclu) et le pas, séparés par des deux-points : `ListName[i:j:k]`.

Le fonctionnement est le même que pour les chaînes de caractères, voici un petit dessin et quelques exemples de rappel (on utilisera cette fois-ci juste une approche par élément au lieu que par tranche) :




```
>>> L[2:4]
[18, 7]
>>> L[2:]
[18, 7, 15, 3]
>>> L[:2]
[12, 11]
>>> L[:]
[12, 11, 18, 7, 15, 3]
```

```
>>> L[2:5]
[18, 7, 15]
>>> L[2:6]
[18, 7, 15, 3]
>>> L[2:7]
[18, 7, 15, 3]
>>> L[2:6:2]
[18, 15]
```

```
>>> L[-2:-4]
[]
>>> L[-4:-2]
[18, 7]
>>> L[-1]
3
```

À noter que, lorsqu'on utilise le slicing, les dépassements d'indices sont licites.

2.1.1. Matrice : liste de listes

Les matrices peuvent être représentées comme des listes imbriquées : chaque ligne est un élément d'une liste. Par exemple, le code

```
A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
```

définit `A` comme la matrice 3×3

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix}.$$

La commande `len` (comme *length*) renvoie la longueur d'une liste. On obtient donc le nombre de ligne de la matrice avec `len(A)` et son nombre de colonnes avec `len(A[0])`. Chaque élément de la liste à deux dimensions est accessible avec la notation `A[i][j]` où l'indice i représente celui de la ligne et l'indice j celui de la colonne. En effet,

```
>>> A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
>>> print(A)
[[11, 12, 13], [21, 22, 23], [31, 32, 33]]
>>> print(A[1])
[21, 22, 23]
>>> print(A[1][2])
23
>>> print(len(A))
3
>>> print(len(A[0]))
3
```

Attention
Dans Python les indices commences à zéro, ainsi `A[0]` indique la première ligne, `A[1]` la deuxième etc.

$$\mathbb{A} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \dots \\ a_{10} & a_{11} & a_{12} & \dots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

2.1.2. Fonctions et méthodes

Voici quelques opérations, fonctions et méthodes très courantes associées aux listes.

Opérations :		
	`x in a`	renvoi `True` si la liste `a` contient l'élément `x`, `False` sinon
	`x not in a`	renvoi `True` si la liste `a` ne contient pas l'élément `x`, `False` sinon

Les opérations `*` et `+` sont aussi définies pour les listes. Cependant, l'utilisation de la méthode `append` est plus efficace que l'instruction `a+=[x]` (voir plus bas) et l'opération `*` doit être utilisée avec beaucoup d'attention (*cf.* la section "Copie d'une liste").

```
>>> a = [1, 2, 3]
>>> print( 2 in a, 5 in a )
True False
```

```
>>> print( 3*a )
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print( a + [4, 5] )
[1, 2, 3, 4, 5]
```

Fonctions :		
	`sum(a)`	renvoie la somme des éléments de la liste `a` si elle ne contient que des nombres
	`len(a)`	renvoie le nombre d'éléments de la liste `a`
	`del a[i]`	supprime l'élément d'indice `i` dans la liste `a`
	`sorted(a)`	renvoi une liste triée (avec les mêmes éléments que la liste `a`). Par défaut tri croissant; un argument optionnel `reverse=true` permet d'inverser le tri.
	`max(a)`	renvoi le plus grand élément de la liste `a`, si la liste est vide on a une erreur
	`min(a)`	renvoi le plus petit élément de la liste `a`, si la liste est vide on a une erreur

```
>>> a = [2, 37, 20, 83, -79, 21]
>>> print(sum(a), len(a), max(a))
84 6 83
```

```
>>> del a[1]
>>> print(a)
```

```
[2, 20, 83, -79, 21]

>>> a = [1, 2, 3]
>>> a[0] = 21
>>> print(a)
[21, 2, 3]

>>> a[2:4] = [-2,-5,-1978]
>>> print(a)
[21, 2, -2, -5, -1978]

>>> a = [1, 3, 2, 5]
>>> L = sorted(a)
>>> M = sorted(a,reverse=True)
>>> print(L,M)
[1, 2, 3, 5] [5, 3, 2, 1]
```

Remarque

Les fonctions `max` et `min` permettent un paramètre optionnel qui est renvoyé lorsqu'on essaie de les appliquer à une liste vide (sinon on lève une erreur). Il s'agit de l'option `default` :

```
>>> a = []
>>> print(max(a,default=100))
100
>>> print(max(a))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: max() arg is an empty sequence
```

Méthodes :

`a.append(x)`	**ajoute** l'élément `x` en fin de la liste `a`
`a.extend(L)`	**ajoute** les éléments de la liste `L` en fin de la liste `a`, équivaut à `a + L`
`a.insert(i,x)`	**ajoute** l'élément `x` en position `i` de la liste `a`, équivaut à `a[i:i]=x` Attention, c'est différent de `a[i]=x`
`a.remove(x)`	**supprime** la première occurrence de l'élément `x` dans la liste `a`
`a.pop(i)`	renvoie l'élément d'indice `i` dans la liste `a` puis le **supprime**
`a.index(x)`	renvoie l'indice de la première occurrence de l'élément `x` dans la liste `a`
`a.clear(x)`	**efface** la liste, équivaut à `a = []`
`a.count(x)`	renvoie le nombre d'occurrence de l'élément `x` dans la liste `a`
`a.sort()`	**modifie** la liste `a` en la triant. Par défaut tri croissant; un argument optionnel `key` permet de trier suivant le résultat du calcul d'une fonction appliquée à chacune des valeurs; un argument optionnel `reverse=true` permet d'inverser le tri
`a.reverse()`	**modifie** la liste `a` en inversant les éléments

Les méthodes `remove`, `pop`, `index` génèrent une erreur si l'élément ne se trouve pas dans la liste.

Attention : les méthodes ne renvoient pas une nouvelle liste mais modifient la liste initiale!

```
>>> a = [2, 37, 20, 83, -79, 21]
>>> print(a)
[2, 37, 20, 83, -79, 21]

>>> a.append(100)
>>> print(a)
[2, 37, 20, 83, -79, 21, 100]

>>> L = [17, 34, 21]
>>> a.extend(L)
>>> print(a)
[2, 37, 20, 83, -79, 21, 100, 17, 34, 21]

>>> a.count(21)
2
>>> a.remove(21)
>>> a.count(21)
1
>>> print(a)
[2, 37, 20, 83, -79, 100, 17, 34, 21]
```

```
>>> a.pop(4)
-79
>>> print(a)
[2, 37, 20, 83, 100, 17, 34, 21]

>>> idx = a.index(100)
>>> print(idx)
4

>>> a.reverse()
```

```
>>> print(a)
[21, 34, 17, 100, 83, 20, 37, 2]

>>> a.sort()
>>> print(a)
[2, 17, 20, 21, 34, 37, 83, 100]

>>> a.insert(2,7)
>>> print(a)
[2, 17, 7, 20, 21, 34, 37, 83, 100]
```

2.1.3. Copie d'une liste

Si a est une liste, la commande `b = a` ne crée pas un nouvel objet b mais simplement une référence (pointeur) vers a. Ainsi, tout changement effectué sur b sera répercuté sur a aussi et viceversa! Observer les deux exemples suivants :

Exemple 1

```
>>> a = [1, 2, 3]
>>> b = a
>>> print(a == b)
True
>>> print(a is b)
True
>>> a[0] = 5
>>> print(a)
[5, 2, 3]
>>> print(b) # Pb
[5, 2, 3]
```

Exemple 2

```
>>> a = [1, 2, 3]
>>> b = a
>>> print(a == b)
True
>>> print(a is b)
True
>>> b[0] = 5
>>> print(a) # Pb!!!
[5, 2, 3]
>>> print(b)
[5, 2, 3]
```

Qu'est-ce qui se passe lorsque on copie une liste a avec la commande `b = a`? En effet, une liste fonctionne comme un carnet d'adresses qui contient les emplacements en mémoire des différents éléments de la liste. Lorsque on écrit `b = a` on dit que b contient les mêmes adresses que a (on dit que les deux listes «pointent» vers le même objet). Ainsi, lorsqu'on modifie la valeur de l'objet, la modification sera visible depuis les deux alias.

Une première solution pour effectuer une copie peut être d'utiliser le *slicing*. En effet, l'opération `[:]` renvoie une nouvelle liste, ce qui résout le problème des pointeurs vers la même zone mémoire, comme le montrent l'exemple 3 suivant. Cette copie peut paraître satisfaisante… et elle l'est, mais à condition de ne manipuler que des listes de premier niveau ne comportant aucune sous-liste, ce qui n'est pas le cas dans l'exemple 4 :

Exemple 3

```
>>> a = [1, 2, 3]
>>> c = a[:] # idem que c = list(a)
>>> print(a == c)
True
>>> print(a is c)
False
>>> c[0] = 5
>>> print(a)
[1, 2, 3]
>>> print(c)
[5, 2, 3]
```

Exemple 4

```
>>> a = [1, 2, 3, [4,5,6]]
>>> c = a[:] # idem que c = list(a)
>>> print(a == c)
True
>>> print(a is c)
False
>>> print(a[3] is c[3]) # Pb !!
True
>>> c[3][0] = 20
>>> print(a)
[1, 2, 3, [20, 5, 6]]
>>> print(c)
[1, 2, 3, [20, 5, 6]]
```

En effet, le slicing n'effectue pas de copie récursive : en cas de sous-liste, on retombe dans la problématique des pointeurs mémoire. La solution est alors d'utiliser un module spécifique, le module `copy` avec sa fonction `deepcopy`, qui permet

d'effectuer une copie récursive. Bien que n'ayant pas encore approfondi la manipulation des modules, voici comment réaliser une copie c de la liste a qui soit vraiment indépendante :

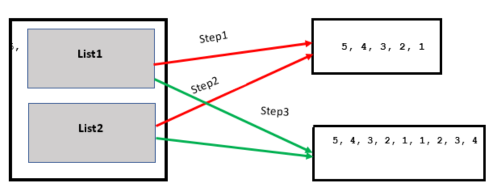
Exemple 5

```
>>> import copy
>>> a = [1, 2, 3, [4,5,6]]
>>> c = copy.deepcopy(a)
>>> print(a == c)
True
>>> print(a is c)
False
>>> print(a[3] is c[3])
False
>>> c[3][0] = 20
>>> print(a)
[1, 2, 3, [4, 5, 6]]
>>> print(c)
[1, 2, 3, [20, 5, 6]]
```

Attention (A+=B)
Les opérateurs augmentés += et *= aussi peuvent donner quelque surprise :

```
list1 = [5, 4, 3, 2, 1]
list2 = list1
list1 += [1, 2, 3, 4]
print(list1, id(list1))
print(list2, id(list2))
```

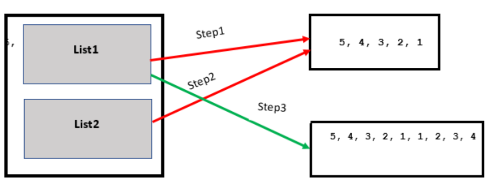
```
[5, 4, 3, 2, 1, 1, 2, 3, 4] 140557459169536
[5, 4, 3, 2, 1, 1, 2, 3, 4] 140557459169536
```



L'expression `list1 += [1, 2, 3, 4]` modifie la liste originale ; comme `list1` et `list2` pointent à la même référence, la liste `list2` aussi sera modifiée.

```
list1 = [5, 4, 3, 2, 1]
list2 = list1
list1 = list1 + [1, 2, 3, 4]
print(list1, id(list1))
print(list2, id(list2))
```

```
[5, 4, 3, 2, 1, 1, 2, 3, 4] 140557457995328
[5, 4, 3, 2, 1] 140557457817920
```



L'expression `list1 = list1 + [1, 2, 3, 4]` crée une nouvelle liste ; après cette instruction `list1` a une nouvelle adresse tandis que `list2` pointe toujours vers l'ancienne référence et ne sera pas modifiée.

2.2. Les tuples

Pour simplifier, les tuples sont des listes particulières qui **ne peuvent pas être modifiées** (on dit immuables). Un tuple est donc une suite d'objets, rangés dans un certain ordre (comme une liste). Chaque objet est séparé par une virgule (comme une liste) et la suite est encadrée par des parenthèses. Un tuple n'est pas forcement homogène (comme une liste) : il peut contenir des objets de types différents les uns des autres.

```
>>> T = ( 2 , 3 , ('a',5) , [4,'toto'] )
>>> print(T)
(2, 3, ('a', 5), [4, 'toto'])

>>> T = ( ) # tuple vide, idem que
```

```
>>> # T = tuple()
>>> print(T)
()
```

ATTENTION

Pour lever toute ambiguïté, **un tuple ne contenant qu'un seul élément doit être noté** `(élément,)`. Si on omet la virgule, Python pensera qu'il s'agit d'un parenthésage superflu et on n'aura donc pas créé un tuple.

```
>>> T = (1)
>>> print(type(T))
<class 'int'>
>>> T = (1,)
>>> print(type(T))
<class 'tuple'>
```

La première manipulation que l'on a besoin d'effectuer sur un tuple, c'est d'en extraire (mais pas modifier) un élément : la syntaxe est `TupleName[index]`. Voici un exemple :

```
>>> mytuple = (12, 10, 18, 7, 15, 3, "toto", 1.5)
>>> print(mytuple)
(12, 10, 18, 7, 15, 3, 'toto', 1.5)
>>> print(mytuple[2])
18
```

ATTENTION

Si on essaye de modifier les éléments d'un tuple on a un message d'erreur :

```
>>> mytuple[1] = 11
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> print(mytuple)
(12, 10, 18, 7, 15, 3, 'toto', 1.5)
```

Toute opération, fonction ou méthode qui s'applique à une liste **sans** la modifier peut être utilisée pour un tuple :

```
>>> a = (2, 37, 20, 83, -79, 21)
>>> print(a)
(2, 37, 20, 83, -79, 21)
>>> a.count(21)
1
>>> a.index(20)
2
>>> print(len(a))
6
>>> print(a+a)
(2, 37, 20, 83, -79, 21, 2, 37, 20, 83, -79, 21)
>>> print(3*a)
(2, 37, 20, 83, -79, 21, 2, 37, 20, 83, -79, 21, 2, 37, 20, 83, -79, 21)
>>> print(a+(4,5))
(2, 37, 20, 83, -79, 21, 4, 5)
```

2.3. L'itérateur `range`

La fonction `range` crée un itérateur. Au lieu de créer et garder en mémoire une liste d'entiers, cette fonction génère les entiers au fur et à mesure des besoins :

- `range(n)` renvoi un itérateur parcourant $[\![0;n-1]\!] = 0,1,2,\dots,n-1$;

- `range(n,m)` renvoi un itérateur parcourant $[\![n;m-1]\!] = n, n+1, n+2, \dots, m-1$;
- `range(n,m,p)` renvoi un itérateur parcourant $n, n+p, n+2p, \dots, m-1$.

```
>>> A = range(0,10)
>>> print(A)
range(0, 10)
```

Pour les afficher on crée une liste avec la fonction `list` :

```
>>> A = range(0,10)
>>> A = list(A)
>>> print(A)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(0))
[]
>>> list(range(1))
[0]
>>> list(range(3,7))
[3, 4, 5, 6]
>>> list(range(0,20,5))
[0, 5, 10, 15]
>>> list(range(0,20,-5))
[]
>>> list(range(0,-20,-5))
[0, -5, -10, -15]
>>> list(range(20,0,-5))
[20, 15, 10, 5]
```

2.4. ☆ Les dictionnaires (ou tableaux associatifs)

Un dictionnaire est une collection modifiable de couples `<clé non modifiable, valeur modifiable>` permettant un accès à la valeur si on fournit la clé. On peut le voir comme une liste dans laquelle l'accès à un élément se fait par un code au lieu d'un indice. L'accès à un élément est optimisé en Python.

Pour ajouter des valeurs à un dictionnaire il faut indiquer une clé ainsi qu'une valeur :

```
>>> Mon1Dico = {} # dictionnaire vide, comme L=[] pour une liste
>>> Mon1Dico = dict() # idem, comme L=list() pour une liste
>>> Mon1Dico["a"] = 1
>>> Mon1Dico["b"] = 2
>>> Mon1Dico["toto"] = 3
>>> print(Mon1Dico)
{'a': 1, 'b': 2, 'toto': 3}

>>> Mon2Dico = {'a':1, 'b':2, 'toto':3}
>>> print(Mon2Dico)
{'a': 1, 'b': 2, 'toto': 3}

>>> Mon3Dico = dict( [ ('a',1) , ('b',2) , ('toto',3) ] )
>>> print(Mon3Dico)
{'a': 1, 'b': 2, 'toto': 3}

>>> % Mon4Dico = dict(a = 1, b = 2, toto = 3) # surprenant
  File "<stdin>", line 1
    % Mon4Dico = dict(a = 1, b = 2, toto = 3) # surprenant
    ^
```

```
SyntaxError: invalid syntax
>>> % print(Mon4Dico)
  File "<stdin>", line 1
    % print(Mon4Dico)
    ^
SyntaxError: invalid syntax
```

On peut utiliser **tout objet non modifiable comme clé**, typiquement une chaîne de caractère comme dans l'exemple précédent ou encore des tuples comme lors de l'utilisation de coordonnées :

```
>>> b = {} # dictionnaire vide
>>> b[(3,2)] = 12 # on ajoute un élément de clé (3,2) et valeur 12
>>> b[(4,5)] = 13 # on ajoute un élément de clé (4,5) et valeur 13
>>> print(b)
{(3, 2): 12, (4, 5): 13}
```

La méthode `get` permet de récupérer une valeur dans un dictionnaire et, si la clé est introuvable, de donner une valeur à retourner par défaut :

```
>>> ficheFG = {}
>>> ficheFG = {"nom": "Faccanoni", "prenom":"Gloria", "batiment": "M", "bureau": 117}
>>> print(ficheFG.get("nom"))
Faccanoni
>>> print(ficheFG.get("telephone", "Numéro inconnu"))
Numéro inconnu
```

Pour vérifier la présence d'une clé on utilise `in` :

```
>>> a = {}
>>> a["nom"] = "Engel"
>>> a["prenom"] = "Olivier"
>>> vf = "nom" in a
>>> print(vf)
True
>>> vf = "age" in a
>>> print(vf)
False
```

Il est possible de supprimer une entrée en indiquant sa clé, comme pour les listes :

```
>>> a = {}
>>> a["nom"] = "Engel"
>>> a["prenom"] = "Olivier"
>>> print(a)
{'nom': 'Engel', 'prenom': 'Olivier'}
>>> del a["nom"]
>>> print(a)
{'prenom': 'Olivier'}
```

- Pour récupérer les clés on utilise la méthode `keys`
- Pour récupérer les valeurs on utilise la méthode `values`
- Pour récupérer les clés et les valeurs en même temps, on utilise la méthode `items` qui retourne un tuple.

On peut alors créer des listes qui contiennent toutes les clés ou toutes les valeurs ou tous les couples clés/valeurs (on verra au chapitre 5 l'utilisation des listes en compréhension) :

```
>>> fiche = {"nom":"Engel","prenom":"Olivier"}
>>> # liste des clés
>>> print( [cle for cle in fiche.keys()] )
['nom', 'prenom']
>>> # liste des valeurs
>>> print( [valeur for valeur in fiche.values()] )
```

```
['Engel', 'Olivier']
>>> # liste de tuples
>>> print( [cv for cv in fiche.items()] )
[('nom', 'Engel'), ('prenom', 'Olivier')]
```

Comme pour les listes, pour créer une copie indépendante utiliser la méthode `copy` :

```
>>> d = {"k1":"Olivier", "k2":"Engel"}
>>> e = d.copy()
>>> print(d)
{'k1': 'Olivier', 'k2': 'Engel'}
>>> print(e)
{'k1': 'Olivier', 'k2': 'Engel'}
>>> d["k1"] = "XXX"
>>> print(d)
{'k1': 'XXX', 'k2': 'Engel'}
>>> print(e)
{'k1': 'Olivier', 'k2': 'Engel'}
```

2.5. ☆ Les ensembles

Les ensembles sont une collection modifiable de valeurs immuables distinctes (uniques) qui ne sont pas ordonnées.

Les listes et les tuples sont des types de données Python standard qui stockent des valeurs dans une séquence. Les ensembles sont un autre type de données Python standard qui stockent également des valeurs. La différence majeure est que les ensembles, contrairement aux listes ou aux tuples, ne peuvent pas avoir plusieurs occurrences du même élément ni stocker des valeurs non ordonnées.

Étant donné que l'ensemble ne peut pas avoir plusieurs occurrences du même élément, il rend l'ensemble très utile pour supprimer efficacement les valeurs en double d'une liste ou d'un tuple et pour effectuer des opérations mathématiques courantes comme les unions et les intersections.

Vous pouvez initialiser un ensemble vide à l'aide de `set()` ou avec des valeurs en lui passant une liste :

```
emptySet = set()
ECUE_MATHS_L1_S1 = set(['M11','M12','M13','M14','I11','P111'])
ECUE_INFO_L1_S1 = set(['M11','M12','I11','I12','P111'])
ECUE_PC_L1_S1 = set(['M11','I11','P111','C111'])
ECUE_SI_L1_S1 = set(['M11','M12','I11','P111'])
print(emptySet)
print(ECUE_MATHS_L1_S1)
print(ECUE_INFO_L1_S1)
print(ECUE_PC_L1_S1)
print(ECUE_SI_L1_S1)
```

```
set()
{'P111', 'M13', 'M12', 'I11', 'M11', 'M14'}
{'P111', 'M12', 'I11', 'M11', 'I12'}
{'P111', 'M11', 'I11', 'C111'}
{'M12', 'P111', 'M11', 'I11'}
```

Si vous regardez la sortie des variables, notez que les valeurs des ensembles ne sont pas dans l'ordre ajouté. En effet, les ensembles ne sont pas ordonnés.

Nota bene : les accolades ne peuvent être utilisées que pour initialiser un ensemble contenant des valeurs car l'utilisation d'accolades sans valeurs est l'un des moyens d'initialiser un dictionnaire et non un ensemble. En revanche, les ensembles contenant des valeurs peuvent également être initialisés à l'aide d'accolades.

```
ECUE_MATHS_L1_S1 = {'M11','M12','M13','M14','I11','P111'}
print(ECUE_MATHS_L1_S1)
```

```
{'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
```

Pour ajouter ou supprimer des valeurs d'un ensemble, vous devez d'abord initialiser un ensemble. Vous pouvez utiliser la méthode `add` pour ajouter une valeur à un ensemble .

```
ECUE_MATHS_L1_S1 .add('MDM')
print(ECUE_MATHS_L1_S1)
```

```
{'P111', 'M13', 'M12', 'M11', 'I11', 'M14', 'MDM'}
```

Nota bene : **un ensemble ne peut contenir que des objets immuables (comme une chaîne ou un tuple)**. Si on essaye d'ajouter une liste à un ensemble on obtient une `TypeError`.

Il existe plusieurs façons de supprimer une valeur d'un ensemble.

Option 1. utiliser la méthode `remove` :

```
ECUE_MATHS_L1_S1 = {'M11','M12','M13','M14','I11','P111','MCM'}
print(ECUE_MATHS_L1_S1)
ECUE_MATHS_L1_S1.remove('MCM')
print(ECUE_MATHS_L1_S1)
```

```
{'MCM', 'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
{'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
```

L'inconvénient de cette méthode est que si vous essayez de supprimer une valeur qui n'est pas dans votre ensemble, vous obtiendrez une `KeyError`.

Option 2. utiliser la méthode `discard` :

```
ECUE_MATHS_L1_S1 = {'M11','M12','M13','M14','I11','P111','MCM'}
print(ECUE_MATHS_L1_S1)
ECUE_MATHS_L1_S1.discard('MCM')
print(ECUE_MATHS_L1_S1)
```

```
{'MCM', 'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
{'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
```

L'avantage de cette approche sur la précédente est que si vous essayez de supprimer une valeur qui ne fait pas partie de l' ensemble, vous n'obtiendrez pas de `KeyError`. Cela fonctionne de manière similaire à la méthode `get` de dictionnaire.

Vous pouvez utiliser la méthode `clear` pour supprimer toutes les valeurs d'un ensemble.

```
ECUE_MATHS_L1_S1 = {'M11','M12','M13','M14','I11','P111','MCM'}
print(ECUE_MATHS_L1_S1)
ECUE_MATHS_L1_S1.clear()
print(ECUE_MATHS_L1_S1)
```

```
{'MCM', 'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
set()
```

Comme pour les listes, on peut parcourir les éléments d'un ensemble :

```
ECUE_MATHS_L1_S1 = {'M11','M12','M13','M14','I11','P111'}
for ecue in ECUE_MATHS_L1_S1:
    print(ecue)
```

```
P111
M13
M12
M11
I11
M14
```

Bien sûr, les valeurs imprimées ne sont pas dans l'ordre dans lequel elles ont été ajoutées (**les ensembles ne sont pas ordonnés**).

Si vous trouvez que vous devez obtenir les valeurs de votre ensemble sous une forme ordonnée, vous pouvez utiliser la fonction `sorted` qui génère une **liste** ordonnée (ici dans l'ordre alphabétique croissant).

```
ECUE_MATHS_L1_S1 = {'M11','M12','M13','M14','I11','P111'}
L=sorted(ECUE_MATHS_L1_S1)
print(L)
```

```
['I11', 'M11', 'M12', 'M13', 'M14', 'P111']
```

Attention

Les ensembles sont le moyen le plus rapide pour supprimer les doublons d'une liste :

```
L=[1,2,3,4,3,2,3,1,2]
print(L)
L=list(set(L))
print(L)
```

```
[1, 2, 3, 4, 3, 2, 3, 1, 2]
[1, 2, 3, 4]
```

ou pour établir si une liste contient de doublons :

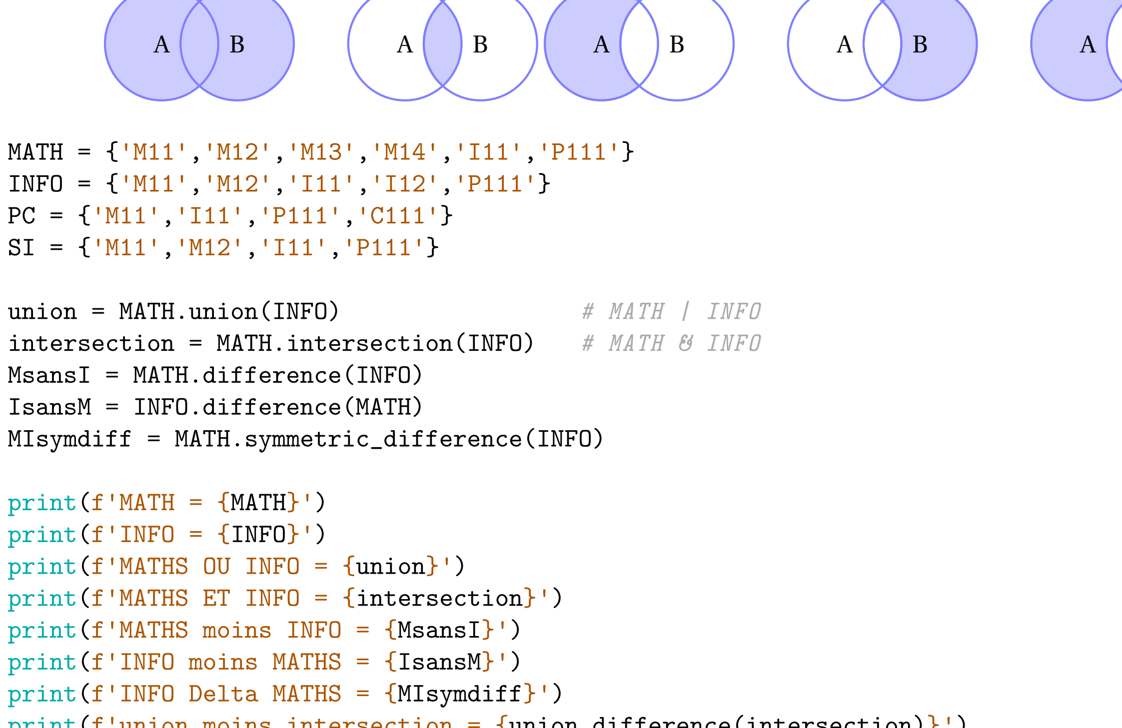
```
L=[1,2,3,4,3,2,3,1,2]
print(len(L)==len(set(L)))
```

```
False
```

Une utilisation courante des ensembles consiste à calculer des opérations mathématiques standard telles que l'union, l'intersection, la différence et la différence symétrique.

Soit E un ensemble. On note $\mathscr{P}(E)$ l'ensemble des parties de E. Soient A et B deux éléments de $\mathscr{P}(E)$. Les quatre éléments $A\cup B$, $A\cap B$, $A\setminus B$, $A\triangle B$ de $\mathscr{P}(E)$ sont définies de la façon suivante : pour tout $x\in E$,

- $x\in A\cup B \iff x\in A$ ou $x\in B$, [réunion des ensembles A et B]
- $x\in A\cap B \iff x\in A$ et $x\in B$, [intersection des ensembles A et B]
- $x\in A\setminus B \iff x\in A$ et $x\notin B$, [différence]
- $x\in A\triangle B \iff x\in A\setminus B$ ou $x\in B\setminus A$. [différence symétrique]




```
MATH = {'M11','M12','M13','M14','I11','P111'}
INFO = {'M11','M12','I11','I12','P111'}
PC = {'M11','I11','P111','C111'}
SI = {'M11','M12','I11','P111'}

union = MATH.union(INFO)                # MATH | INFO
intersection = MATH.intersection(INFO)  # MATH & INFO
MsansI = MATH.difference(INFO)
IsansM = INFO.difference(MATH)
MIsymdiff = MATH.symmetric_difference(INFO)

print(f'MATH = {MATH}')
print(f'INFO = {INFO}')
print(f'MATHS OU INFO = {union}')
print(f'MATHS ET INFO = {intersection}')
print(f'MATHS moins INFO = {MsansI}')
print(f'INFO moins MATHS = {IsansM}')
print(f'INFO Delta MATHS = {MIsymdiff}')
print(f'union moins intersection = {union.difference(intersection)}')
```

 © G. Faccanoni

```
MATH = {'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
INFO = {'P111', 'M12', 'M11', 'I11', 'I12'}
MATHS OU INFO = {'P111', 'M13', 'M12', 'I11', 'M11', 'M14', 'I12'}
MATHS ET INFO = {'M12', 'P111', 'M11', 'I11'}
MATHS moins INFO = {'M14', 'M13'}
INFO moins MATHS = {'I12'}
INFO Delta MATHS = {'M13', 'M14', 'I12'}
union moins intersection = {'M14', 'M13', 'I12'}
```

Vous pouvez vérifier si un ensemble est un sous-ensemble d'un autre en utilisant la méthode `issubset`.

```
MATH = {'M11','M12','M13','M14','I11','P111'}
Mxx = {'M11','M12','M13','M14'}


print(f'MATH = {MATH}')
print(f'Mxx = {Mxx}')
print(f'Mxx sous-ensemble de MATH ? {Mxx.issubset(MATH)}')
```

```
MATH = {'P111', 'M13', 'M12', 'M11', 'I11', 'M14'}
Mxx = {'M12', 'M11', 'M13', 'M14'}
Mxx sous-ensemble de MATH ? True
```

2.6. Exercices

Conversions entre types : `int`, `float`, `str`, `list`, `tuple`

On rappelle ici quelque conversion entre types :

- Pour transformer une liste L ou une chaîne de caractères s en un tuple on utilisera `tuple()`

```
>>> L = [1, "cool", 1.41]
>>> tuple(L)
(1, 'cool', 1.41)
>>> s = "cool"
>>> tuple(s)
('c', 'o', 'o', 'l')
```

- Pour transformer un tuple T ou une chaîne de caractères s en une liste on utilisera `list()`

```
>>> T = (1, "cool", 1.41)
>>> list(T)
[1, 'cool', 1.41]
>>> s = "cool"
>>> list(s)
['c', 'o', 'o', 'l']
```

- Pour transformer un nombre n ou r en une chaîne de caractères on utilisera `str()`

```
>>> n = 123
>>> str(n)
'123'
>>> r = -1.41
>>> str(r)
'-1.41'
```

- Pour transformer une chaîne de caractères en un nombre (si cela a du sens) on utilisera `int()` ou `float()` ou `eval()` (use-case typique : lecture de données via `input`)

```
>>> s = '123'
>>> int(s)
123
>>> float(s)
123.0
>>> eval(s)
123
>>> s = "-1.41"
>>> float(s)
-1.41
>>> eval(s)
-1.41
```

Exercice 2.1 (Listes et sous-listes)

On considère la liste

```
L = [ (0 ,1), "3.1415", 2, 3, [0.1, 0.2, "trois"] ]
```

Comment obtenir le floating 3.1415 ? À quelle expression correspond la chaîne `'roi'` ? À quelle expression correspond la chaîne `'5'` ? Comment obtenir l'entier 5 ?

Correction

La liste L contient `len(L)` = 5 éléments. Les indices vont de 0 à `len(L)-1` = 4. Analysons chaque élément, en indiquant en particulier son type :

L[0]	L[1]	L[2]	L[3]	L[4]
(0,1) tuple	"3.1415" str	2 int	3 int	[0.1, 0.2, "trois"] list

Les éléments `L[0]`, `L[1]` et `L[4]` sont des structures qu'on peut encore analyser.

Nota bene : l'élément `L[1]` n'est pas un floating mais une chaîne de caractères (qui peut être transformée en un floating).

- Tuple `L[0] = (0,1)`

L[0][0]	L[0][1]
0 int	1 int

- Chaîne de caractères `L[1] = "3.1415"`

L[1][0]	L[1][1]	L[1][2]	L[1][3]	L[1][4]	L[1][5]
'3' str	'.' str	'1' str	'4' str	'1' str	'5' str

- Liste `L[4] = [0.1, 0.2, "trois"]`

L[4][0]	L[4][1]	L[4][2]
0.1 float	0.2 float	"trois" str

Le dernier élément de cette liste peut encore être décomposé :

L[4][2][0]	L[4][2][1]	L[4][2][2]	L[4][2][3]	L[4][2][4]
't' str	'r' str	'o' str	'i' str	's' str

On peut alors répondre aux questions de l'exercice :

```
>>> L = [ (0 ,1), "3.1415", 2, 3, [0.1, 0.2, "trois"] ]
>>> float(L[1])
3.1415
>>> L[4][2][1:4] # idem que L[-1][2][1:4]
'roi'
>>> L[1][-1] # idem que L[1][5], c'est un string
'5'
>>> int(L[1][-1]) # c'est un entier
5
```

Exercice 2.2 (Devine le résultat)

```
L = [ 1, 2, 3, "a", "b", "toto", "zoo", "-3.14", -3.14, [8,9,5] ]
```

```
print(L[2])          print(L[2]*2)        print(float(L[7])*3)
print(L[5:7])        print(L[2]+2)        print(L[8]+2)
print(L[5:])         print(L[8]*3)        print(L[7]+2)
print(L[9][2])       print(L[7]*3)        print(L[7]+"2")
```

Attention : certaines instructions lèvent une erreur. Expliquer pourquoi.

Correction

```
>>> L = [1,2,3,"a","b","toto","zoo","-3.14",-3.14,[8,9,5]]
>>> print(L[2])
3
>>> print(L[5:7])
['toto', 'zoo']
>>> print(L[5:])
['toto', 'zoo', '-3.14', -3.14, [8, 9, 5]]
>>> print(L[9][2]) # L[9]=[8,9,5] et l'élément d'indice 2 de cette sous-liste est 5
5
>>> print(L[2]*2) # L[2] est un nombre
6
>>> print(L[2]+2)
```

 © G. Faccanoni

```
5
>>> print(L[8]*3) # L[8] étant un nombre, on multiplie tout simplement
-9.42
>>> print(L[7]*3) # L[7] est un string donc on concatène 3 fois
-3.14-3.14-3.14
>>> print(float(L[7])*3) # float(L[7]) est un nombre donc on multiplie tout simplement
-9.42
>>> print(L[8]+2) # L[8] étant un nombre, on additionne tout simplement
-1.1400000000000001
>>> print(L[7]+2) # L[7] est un string, on ne peut pas lui concaténer un int
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> print(L[7]+"2") # L[7] est un string et "2" aussi, on peut les concaténer
-3.142
```

Exercice 2.3 (Insertions)
Considérons la liste

```
planets = ["Mercury", "Mars", "Earth", "Neptune"]
```

1. Ajouter `"Jupiter"` à la fin de la liste.
2. Ajouter `"Uranus"` et `"Saturn"` à la fin de la liste.
3. Ajouter `"Venus"` entre `"Mars"` and `"Earth"`.

Correction
Pour ajouter un seul élément, on utilise la méthode `append()` ; pour concaténer une liste à une autre liste on utilise la méthode `extend()` ; pour insérer un (ou plusieurs) élément(s) au milieux d'une liste, on extrait les deux morceaux de la liste avec un slicing puis on concatène les différents listes avec l'opérateur + :

```
planets = ["Mercury", "Mars", "Earth", "Neptune"]
print(planets)
planets.append("Jupiter")
print(planets)
planets.extend(["Uranus","Saturn"])
print(planets)
planets = planets[:2] + ["Venus"] + planets[2:]
print(planets)
```

```
['Mercury', 'Mars', 'Earth', 'Neptune']
['Mercury', 'Mars', 'Earth', 'Neptune', 'Jupiter']
['Mercury', 'Mars', 'Earth', 'Neptune', 'Jupiter', 'Uranus', 'Saturn']
['Mercury', 'Mars', 'Venus', 'Earth', 'Neptune', 'Jupiter', 'Uranus', 'Saturn']
```

Exercice 2.4 (Moyenne)
Soit L une liste de nombres. Calculer la somme des éléments de L, puis le nombre d'éléments de L et en déduire la moyenne arithmétique des éléments de L. Par exemple, si `L = [0,1,2,3]`, on doit obtenir 1.5. Vérifier votre code sur d'autres exemples.

Correction
Pour calculer la somme des éléments de la liste, on utilise la fonction `sum()` ; pour calculer combien d'éléments contient la liste on utilise la fonction `len()` :

```
>>> L = list(range(4))
>>> moy = sum(L)/len(L)
>>> print(f"L = {L}, Moyenne = {moy}")
L = [0, 1, 2, 3], Moyenne = 1.5
```

Exercice 2.5 (Effectifs et fréquence)

Soit L une liste de nombres entiers donnés. Soit n un élément dans L. Calculer l'effectif et la fréquence de n dans cette liste.

Par exemple, si `L=[0,10,20,10,20,30,20,30,40,20]`, et $n = 20$ alors son effectif est 4 (nombre de fois où il apparaît) et sa fréquence est $4/10 = 0.4$ (effectif de la valeur / effectif total).

Correction

Pour calculer combien de fois un élément apparaît, on utilise la méthode `count()` ; pour calculer combien d'éléments contient la liste on utilise la fonction `len()` :

```
>>> L = [0,10,20,10,20,30,20,30,40,20]
>>> n = 20
>>> eff = L.count(n)
>>> print(f"L'élément n = {n} apparaît {eff} fois avec une fréquence de {eff/len(L)}")
L'élément n = 20 apparaît 4 fois avec une fréquence de 0.4
```

Exercice 2.6 (Max-Min)

La fonction `min` (resp. `max`) renvoie la valeur la plus petite (resp. grande) d'une liste d'éléments numériques ou la première (resp. dernière) chaîne de caractères (selon l'ordre alphabétique) si la liste contient des chaînes de caractères. La syntaxe est `min(iterable, default)`

- `iterable` : une liste, un tuple, un ensemble, un dictionnaire, etc.
- `default` (facultatif) : valeur par défaut si l'itérable donné est vide (sans ce paramètre, si la liste est vide la fonction lève une erreur).

Soit L une liste de nombres. Vous devez trouver la différence entre les éléments maximal et minimal. Pour une liste vide, il faut afficher 0 (sans utiliser `if`).

Par exemple, si `L = [1,2,3]`, on doit obtenir 2; si `L = [5,-5]`, on doit obtenir 10; si `L = []`, on doit obtenir 0.

Correction

```
>>> L = [1,2,3]; print(f"L = {L}, max-min = {max(L,default=0)-min(L,default=0)}")
L = [1, 2, 3], max-min = 2
>>> L = [5,-5]; print(f"L = {L}, max-min = {max(L,default=0)-min(L,default=0)}")
L = [5, -5], max-min = 10
>>> L = [5]; print(f"L = {L}, max-min = {max(L,default=0)-min(L,default=0)}")
L = [5], max-min = 0
>>> L = []; print(f"L = {L}, max-min = {max(L,default=0)-min(L,default=0)}")
L = [], max-min = 0
>>> L = []; print(f"L = {L}, max-min = {max(L)-min(L)}") # Erreur !!!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: max() arg is an empty sequence
```

Exercice 2.7 (Range)

En utilisant l'itérateur `range`, générer et afficher les listes suivantes :

- $A = [2,3,4,5,6,7,8,9]$
- $B = [9,18,27,36,45,54,63,72,81,90]$
- $C = [2,4,6,8,\dots,38,40]$
- $D = [1,3,5,7,\dots,37,39]$
- $E = [19,17,15,13,\dots,3,1]$

Correction

`range(a,b,p)` génère la suite a, $a+p$, $a+2p$, ..., $a+np<b$.
`range(a,b)` équivaut à `range(a,b,1)` et
`range(b)` équivaut à `range(0,b,1)`.

range est une itérateur (la suite n'est générée que lorsqu'on en a besoin). Pour pouvoir afficher la suite on la transforme d'abord en une liste.

```
A = range(2,10)
print(f"A = {list(A)}")
B = range(9,91,9)
print(f"B = {list(B)}")
C = range(2,41,2)
print(f"C = {list(C)}")
D = range(1,40,2)
print(f"D = {list(D)}")
E = range(19,0,-2)
print(f"E = {list(E)}")
```

```
A = [2, 3, 4, 5, 6, 7, 8, 9]
B = [9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
C = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40]
D = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39]
E = [19, 17, 15, 13, 11, 9, 7, 5, 3, 1]
```

Exercice 2.8 (Note ECUE)

Soit CT, CC, TP respectivement les notes de contrôle terminal, de contrôle continu et de travaux pratiques d'un ECUE. La note finale est calculée selon la formule

$$0.3\text{TP} + \max\left(0.7\text{CT}, 0.5\text{CT} + 0.2\text{CC}\right).$$

Écrire un script qui calcule la note finale dans les cas suivants (vérifier les résultats !) :

1. TP=10, CT=10, CC=10 ;
2. TP=10, CT=10, CC=20 ;
3. TP=10, CT=20, CC=10 ;
4. TP=20, CT=10, CC=10 ;
5. TP=20, CT=10, CC=20 ;
6. TP=20, CT=20, CC=10.

Correction

```
print( 0.3*TP+max( 0.7*CT , 0.5*CT+0.2*CC ) )
```

Cas 1 : (TP,CT,CC) = (10, 10, 10), Note = 10.0

Cas 2 : (TP,CT,CC) = (10, 10, 20), Note = 12.0

Cas 3 : (TP,CT,CC) = (10, 20, 10), Note = 17.0

Cas 4 : (TP,CT,CC) = (20, 10, 10), Note = 13.0

Cas 5 : (TP,CT,CC) = (20, 10, 20), Note = 15.0

Cas 6 : (TP,CT,CC) = (20, 20, 10), Note = 20.0

Noter que

$$0.3\text{TP} + \max(0.7\text{CT}, 0.5\text{CT} + 0.2\text{CC}) = 0.3\text{TP} + 0.5\text{CT} + 0.2\max(\text{CT}, \text{CC})$$

ainsi on aurait pu écrire

```
print( 0.3*TP + 0.5*CT + 0.2*max(CT,CC) )
```

Exercice 2.9 (Devine le résultat - tuples)

```
a = 'Quattro TT'
print( tuple(a) )
print( tuple([a]) )
print( tuple(list(a)) )
print( (a,) )
print( (a)  )
print( tuple(a.split()) )
```

Correction

Les tuples vides sont construits par une paire de parenthèses vides

```
>>> empty = ()
>>> print(empty, type(empty), len(empty))
() <class 'tuple'> 0
```

Les tuples contenant un unique élément sont construits en ajoutant une virgule à l'élément (il ne suffit pas de mettre l'élément entre parenthèses) :

```
>>> s = 'hello'

>>> # t n'est pas un tuple !
>>> t = (s)
>>> print(t, type(t), len(t))
hello <class 'str'> 5
```

```
>>> # avec la virgule, t devient un tuple!
>>> t = s,
>>> print(t, type(t), len(t))
('hello',) <class 'tuple'> 1
```

Ce qu'il faut retenir : **les tuples (non vides) ne sont pas formés par les parenthèses, mais par l'utilisation de l'opérateur de virgule.**

Pour notre exercice on obtient

```
>>> a = 'Quattro TT'
>>> print( tuple(a) )   # <-- Tuple from a sequence (which is a string)
('Q', 'u', 'a', 't', 't', 'r', 'o', ' ', 'T', 'T')
>>> print( tuple([a]) ) # <-- Tuple from a sequence (which is a list containing a string)
('Quattro TT',)
>>> print( tuple(list(a)) ) # <-- Tuple from a sequence (which you create from a string)
('Q', 'u', 'a', 't', 't', 'r', 'o', ' ', 'T', 'T')
>>> print( (a,) )  # <-- Tuple containing only one element, the string
('Quattro TT',)
>>> print( (a)   )  # <-- it's just the string wrapped in parentheses
Quattro TT
>>> print( tuple(a.split()) )
('Quattro', 'TT')
```

Exercice 2.10 (LE piège avec la copie de listes – I)
Devine le résultat.

1. Modification de la première liste
```
a = list(range(5))
b = a
print(f"{a=}\n{b=}")
a[0] = 99
print(f"{a=}\n{b=}")
```
2. Modification de la deuxième liste
```
a = list(range(5))
b = a
print(f"{a=}\n{b=}")
b[0] = 99
print(f"{a=}\n{b=}")
```
3. *Shallow copy* (copie superficielle)
```
a = list(range(5))
b = a[:]
print(f"{a=}\n{b=}")
b[0] = 99
print(f"{a=}\n{b=}")
```

Correction
Avant d'exécuter les instructions données, essayons de comprendre comment Python gère une liste et ses copies :

```
>>> l = list(range(3))  # l *pointe* vers la liste [0,1,2]
>>> m = l
>>> print(f"l = {l}\nm = {m}")
l = [0, 1, 2]
m = [0, 1, 2]
>>> # En réalité m n'est pas une copie de la liste l mais un *alias* :
>>> print(id(l)) # id(obj) retourne le n° d'identification en mémoire
140558345334272
>>> print(id(m)) # on voit qu'ils correspondent bien au même objet en mémoire
140558345334272

>>> l[0] = 'a' # puisqu'on modifie l, m aussi est modifiée !
```

 © G. Faccanoni

```
>>> print(f"l = {l}\nm = {m}")
l = ['a', 1, 2]
m = ['a', 1, 2]

>>> n = l[:] # ici on dit que n n'est pas une alias mais une copie des *éléments* de l
↪   (clonage)
>>> print(f"l = {l}\nn = {n}")
l = ['a', 1, 2]
n = ['a', 1, 2]
>>> print(id(l))
140558345334272
>>> print(id(n)) # n a un id différent de l : il s'agit de 2 objets distincts
140558364236992

>>> del l[-1] # on efface le dernier élément de l mais les éléments de n n'ont pas été
↪   modifiés
>>> print(f"l = {l}\nn = {n}")
l = ['a', 1]
n = ['a', 1, 2]
```

On peut alors deviner les résultats de l'exercice :

1. Modification de la première liste

```
a = list(range(5))
b = a
print(f"{a = }\n{b = }")
a[0] = 99
print(f"{a = }\n{b = }")
```

```
a = [0, 1, 2, 3, 4]
b = [0, 1, 2, 3, 4]
a = [99, 1, 2, 3, 4]
b = [99, 1, 2, 3, 4]
```

2. Modification de la deuxième liste

```
a = list(range(5))
b = a
print(f"{a = }\n{b = }")
b[0] = 99
print(f"{a = }\n{b = }")
```

```
a = [0, 1, 2, 3, 4]
b = [0, 1, 2, 3, 4]
a = [99, 1, 2, 3, 4]
b = [99, 1, 2, 3, 4]
```

3. *Shallow copy* (copie superficielle)

```
a = list(range(5))
b = a[:]
print(f"{a = }\n{b = }")
b[0] = 99
print(f"{a = }\n{b = }")
```

```
a = [0, 1, 2, 3, 4]
b = [0, 1, 2, 3, 4]
a = [0, 1, 2, 3, 4]
b = [99, 1, 2, 3, 4]
```

☆ Exercice Bonus 2.11 (LE piège avec la copie de listes – II)
Devine le résultat.

1. Copie d'une variable scalaire

```
a = 2
b = a
print(f"{a=}, {b=}")
a = 3
print(f"{a=}, {b=}")
```

2. Copie et ré-affectation d'une liste

```
a = [2]
b = a
print(f"{a=}, {b=}")
a = [3]
print(f"{a=}, {b=}")
```

3. Copie et modification d'une liste

```
a = [2]
b = a
print(f"{a=}, {b=}")
a[0] = 3
print(f"{a=}, {b=}")
```

Correction
Pour bien comprendre on affiche les adresses où sont stockés les objets :

1. Copie d'une variable scalaire

```
a = 2
b = a
print(f"{a = }, {b = }")
print(f"{id(a) = }")
print(f"{id(b) = }")

a = 3
print(f"{a = }, {b = }")
print(f"{id(a) = }")
print(f"{id(b) = }")
```

```
a = 2, b = 2
id(a) = 140050256183568
id(b) = 140050256183568
a = 3, b = 2
id(a) = 140050256183600
id(b) = 140050256183568
```

2. Copie et ré-affectation d'une liste

```
a = [2]
b = a
print(f"{a = }, {b = }")
print(f"{id(a) = }")
print(f"{id(b) = }")
a = [3]
print(f"{a = }, {b = }")
print(f"{id(a) = }")
print(f"{id(b) = }")
```

```
a = [2], b = [2]
id(a) = 140050253820416
id(b) = 140050253820416
a = [3], b = [2]
id(a) = 140050253820544
id(b) = 140050253820416
```

3. Copie et modification d'une liste

```
a = [2]
b = a
print(f"{a = }, {b = }")
print(f"{id(a) = }")
print(f"{id(b) = }")

a[0] = 3
print(f"{a = }, {b = }")
print(f"{id(a) = }")
print(f"{id(b) = }")
```

```
a = [2], b = [2]
id(a) = 140050253820480
id(b) = 140050253820480
a = [3], b = [3]
id(a) = 140050253820480
id(b) = 140050253820480
```

Exercice 2.12 (Copie de listes de listes)
Devine le résultat.

1.
```
a = [ [1,2] , [3,4] ]
b = a
print(f"{a = }\n{b = }")
b[0] = 99
print(f"{a = }\n{b = }")
```

2. *Shallow copy*
```
a = [ [1,2] , [3,4] ]
b = a[:]
print(f"{a = }\n{b = }")
b[0] = 99
b[1][0] = 88
print(f"{a = }\n{b = }")
```

3. *Deep copy*
```
import copy
a = [ [1,2] , [3,4] ]
b = copy.deepcopy(a)
print(f"{a = }\n{b = }")
b[0] = 99
b[1][0] = 88
print(f"{a = }\n{b = }")
```

4. L'opérateur `*`
```
a = [[1,2]]*5
print(f"a = {a}")
a[0][0] = 99
print(f"a = {a}")
```

Correction
Pour bien comprendre on affiche les adresses où sont stockés les objets :

1.
```
>>> a = [ [1,2] , [3,4] ]
>>> b = a
>>> print(f"{a = }\n{b = }\n{id(a) = }\n{id(b) = }")
a = [[1, 2], [3, 4]]
b = [[1, 2], [3, 4]]
id(a) = 140558344571840
id(b) = 140558344571840
>>> b[0] = 99
>>> print(f"{a = }\n{b = }\n{id(a) = }\n{id(b) = }")
a = [99, [3, 4]]
b = [99, [3, 4]]
id(a) = 140558344571840
id(b) = 140558344571840
```

2. *Shallow copy*
```
>>> a = [ [1,2] , [3,4] ]
>>> b = a[:]        # copie superficielle
>>> print(f"{a = }\n{b = }\n{id(a) = }\n{id(b) = }")
```

```
a = [[1, 2], [3, 4]]
b = [[1, 2], [3, 4]]
id(a) = 140558345433088
id(b) = 140558346294592
>>> b[0] = 99      # affecte juste b
>>> b[1][0] = 88   # affecte b et a
>>> print(f"{a = }\n{b = }\n{id(a) = }\n{id(b) = }\n{id(a[1]) = }\n{id(b[1]) = }")
a = [[1, 2], [88, 4]]
b = [99, [88, 4]]
id(a) = 140558345433088
id(b) = 140558346294592
id(a[1]) = 140558345434368
id(b[1]) = 140558345434368
```

3. *Deep copy*

```
>>> import copy
>>> a = [ [1,2] , [3,4] ]
>>> b = copy.deepcopy(a)
>>> print(f"{a = }\n{b = }\n{id(a) = }\n{id(b) = }")
a = [[1, 2], [3, 4]]
b = [[1, 2], [3, 4]]
id(a) = 140558344574528
id(b) = 140558346296320
>>> b[0] = 99
>>> b[1][0] = 88
>>> print(f"{a = }\n{b = }\n{id(a) = }\n{id(b) = }\n{id(a[1]) = }\n{id(b[1]) = }")
a = [[1, 2], [3, 4]]
b = [99, [88, 4]]
id(a) = 140558344574528
id(b) = 140558346296320
id(a[1]) = 140558345433728
id(b[1]) = 140558346565248
```

4. L'opérateur `*`

```
>>> a = [[1,2]]*5
>>> print(f"{a = }")
a = [[1, 2], [1, 2], [1, 2], [1, 2], [1, 2]]
>>> a[0][0] = 99
>>> print(f"{a = }")
a = [[99, 2], [99, 2], [99, 2], [99, 2], [99, 2]]
```

Exercice 2.13 (Devine le résultat)

```
x = list(range(1,-1,-1)) + list(range(1))
y = x[:]
z = y
z[0] = [y[k] for k in x]
x[1:3] = y[0][1:3]
z[len(y[0])-1] = 0
```

Correction

```
>>> x = list(range(1,-1,-1)) + list(range(1)) # [1,0]+[0]=[1,0,0]
>>> y = x[:] # copie superficielle
>>> z = y # z et y sont la même liste
>>> print(f"{x=}, {id(x)=}\n{y=}, {id(y)=}\n{z=}, {id(z)=}")
x=[1, 0, 0], id(x)=140558344574528
y=[1, 0, 0], id(y)=140558345365760
z=[1, 0, 0], id(z)=140558345365760
>>> z[0] = [y[k] for k in x] # en modifiant z[0] on modifie aussi y[0]
```

```
>>> print(f"{x=}, {id(x)=}\n{y=}, {id(y)=}\n{z=}, {id(z)=}")
x=[1, 0, 0], id(x)=140558344574528
y=[[0, 1, 1], 0, 0], id(y)=140558345365760
z=[[0, 1, 1], 0, 0], id(z)=140558345365760
>>> x[1:3] = y[0][1:3] # on ne modifie que x
>>> print(f"{x=}, {id(x)=}\n{y=}, {id(y)=}\n{z=}, {id(z)=}")
x=[1, 1, 1], id(x)=140558344574528
y=[[0, 1, 1], 0, 0], id(y)=140558345365760
z=[[0, 1, 1], 0, 0], id(z)=140558345365760
>>> z[len(y[0])-1] = 0 # y[0]=[0,1,1], len(...)=3, z[2]=0
>>> print(f"{x=}, {id(x)=}\n{y=}, {id(y)=}\n{z=}, {id(z)=}")
x=[1, 1, 1], id(x)=140558344574528
y=[[0, 1, 1], 0, 0], id(y)=140558345365760
z=[[0, 1, 1], 0, 0], id(z)=140558345365760
```

☆ Exercice Bonus 2.14 (`str`⇝`list` : `split` et `join`)

Parfois il peut être utile de transformer une chaîne de caractère en liste. Cela est possible avec la méthode `split`. L'inverse est possible avec la méthode `join`.

Tester les commandes suivantes :

```
s = "Gloria:FACCANONI:Université de Toulon"
print(s)
L = s.split(":")
print(L)
print("--"*30)
L = ["Gloria","FACCANONI","Université de Toulon"]
print(L)
s = "-".join(L)
print(s)
```

Correction

```
Gloria:FACCANONI:Université de Toulon
['Gloria', 'FACCANONI', 'Université de Toulon']
------------------------------------------------------------
['Gloria', 'FACCANONI', 'Université de Toulon']
Gloria-FACCANONI-Université de Toulon
```

☆ Exercice Bonus 2.15 (Liste de tuples et Tuple de listes)

Un tuple est immuables, un liste est mutable. Que se passe-t-il si on modifie un tuple de listes ? Et une liste de tuples ?

Correction

- tuple de listes

```
>>> T = (  [1,2] , ["trois","quatre"] )
>>> T[1].append("cinq") # on modifie une liste
>>> print(T)
([1, 2], ['trois', 'quatre', 'cinq'])
```

- liste de tuples

```
>>> L = [  (1,2) , ("trois","quatre") ]
>>> L[1].append("cinq") # on essaye de modifier un tuple, c'est interdit
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> print(L)
[(1, 2), ('trois', 'quatre')]
```

CHAPITRE 3

Structure conditionnelle

Le déroulement d'un programme est l'ordre dans lequel les lignes de code sont exécutées. Certaines lignes seront lues une fois seulement, d'autres plusieurs fois. D'autres encore pourraient être complètement ignorées, tout dépend de la façon dont vous les avez codées. Dans ce premier chapitre sur le déroulement du programme, nous allons regarder comment programmer un code avec des instructions conditionnelles. Les instructions conditionnelles sont un moyen de contrôler la logique et le déroulement du code avec des conditions.

3.1. Définir des conditions avec les instructions `if/else`

L'un des blocs essentiels à la structure d'un déroulement conditionnel est l'instruction `if`. Avec cette instruction, on peut exécuter certaines lignes de code uniquement si une certaine condition est vraie (`True`). Si cette condition est fausse (`False`), le code ne s'exécutera pas. Les instructions `if/elif/else` vous permettent de définir des conditions multiples. Le mot-clé `elif` permet d'ajouter autant de conditions qu'on veut. On peut ensuite terminer avec une instruction `else`. Voici un exemple :

$$y = \begin{cases} x & \text{si } x \le -5, \\ 100 & \text{si } -5 < x \le 0, \\ x^2 & \text{si } 0 < x < 10, \\ x-2 & \text{sinon.} \end{cases}$$

Exemples :

Avec x=-10 on a y=-10

Avec x= -5 on a y=-5

Avec x= -1 on a y=100

Avec x= 5 on a y=25

Avec x= 15 on a y=13

Code :

```
x = ...

if x<=-5:
    y = x
elif x<=0: # -5<x<=0
    y = 100
elif x<10:
    y = x**2
else:
    y = x-2

print(x,y)
```

Ce code va vérifier si $x \le -5$ est vrai. Si c'est le cas, il affecte $y = x$ et passe à l'instruction `print` ; si c'est faux, il va vérifier si $x \le 0$ (inutile de lui redemander si $x > -5$). Si c'est le cas, il affecte $y = 100$ et passe à l'instruction `print` ; si c'est faux, il va vérifier si $x < 10$ (inutile de lui redemander si $x > 0$). Si c'est le cas, il affecte $y = x^2$ et passe à l'instruction `print` ; si c'est faux, il affecte $y = x - 2$ et passe à l'instruction `print`.

La syntaxe générale est la suivante :

```
if condition_1:
    instruction_1.1
    instruction_1.2
    ...
elif condition_2: # bloc facultatif
    instruction_2.1
    instruction_2.2
    ...
elif condition_3: # bloc facultatif
    instruction_3.1
    instruction_3.2
    ...
...
else:             # bloc facultatif
    instruction_n.1
    instruction_n.2
    ...
```

où `condition_1`, `condition_2`... représentent des ensembles d'instructions dont la valeur est `True` ou `False` (on les obtient en général en utilisant les opérateurs de comparaison).

La première condition `condition_i` ayant la valeur `True` entraîne l'exécution des instructions `instruction_i.1`, `instruction_i.2`... et la non exécution des autres instructions `instruction_j.1`, `instruction_j.2`... avec $j \neq i$.

Si toutes les conditions sont fausses, les instructions `instruction_n.1`, `instruction_n.2`... sont exécutées.

Les conditions font intervenir les opérateurs de comparaison ==, !=, <, <=, >, >=. Pour vérifier plusieurs conditions pour un seul résultat dans la même instruction `if`, on utilisera les connecteurs logiques `and` (qui vérifie si deux conditions sont toutes les deux vraies), `or` (qui vérifie si au moins une condition est vraie), `not` (qui vérifie si une condition n'est pas vraie, c'est-à-dire si elle est fausse). On a vu ces opérateurs et connecteurs à la page 21.

ATTENTION
Bien noter le rôle essentiel de l'**indentation** qui permet de délimiter chaque bloc d'instructions et la présence des **deux points** après la condition du choix (mot clé `if` et mot clé `elif`) et après le mot clé `else`.

Pour comprendre l'exécution d'un code pas à pas on pourra utiliser : Visualize code and get live help
http://pythontutor.com/visualize.html

Voici une série d'exemples.

1. Dans cette exemple on avertit un conducteur s'il dépasse une vitesse donnée :
```
if vitesse > 130:
    print("Attention : tu dépasses la limitation de vitesse!")
```
2. Dans cette exemple on calcule une note (A, B, C, D ou E) en fonction du score obtenu à un test selon le schéma suivant :

Score	≥ 90	$\in [80;90[$	$\in [70;80[$	$\in [60;70[$	< 60
Note	A	B	C	D	E

```
if score >= 90:
    print("A")
elif score >= 80: # inutile d'écrire 80<=score<90 car le cas score>=90 a déjà été
↪ considéré
    print("B")
elif score >= 70:
    print("C")
elif score >= 60:
    print("D")
else:
    print("E")
```

 © G. Faccanoni

3. Exemple avec juste le mot clé `if` :

```
a = 3
if a > 5:
    a = a + 1
print(f"a={a}")
```

a=3

```
a = 10
if a > 5:
    a = a + 1
print(f"a={a}")
```

a=11

```
a = 10
if 5<a<10:
    a = a + 1
print(f"a={a}")
```

a=10

4. On ajoute un bloc `else` et un bloc `elif` :

```
a = 3
if a > 5:
    a = a + 1
else:
    a = a-1
print(f"a={a}")
```

a=2

```
a = 5
if a > 5:
    a = a + 1
elif a==5:
    a = a+1000
else:
    a = a-1
print(f"a={a}")
```

a=1005

```
a = 10
if 5<a<10:
    a = a + 1
print(f"a={a}")
```

a=10

5. Dans cette exemple on établi si un nombre est positif :

```
if a < 0:
    print('a is negative')
elif a > 0:
    print('a is positive')
else:
    print('a is zero')
```

et on teste le code pour différentes valeurs de a :

```
a = 2
if a < 0:
    print('a is negative')
elif a > 0:
    print('a is positive')
else:
    print('a is zero')
```

a is positive

```
a = 0
if a < 0:
    print('a is negative')
elif a > 0:
    print('a is positive')
else:
    print('a is zero')
```

a is zero

```
a = -2
if a < 0:
    print('a is negative')
elif a > 0:
    print('a is positive')
else:
    print('a is zero')
```

a is negative

Remarque (Syntaxe compacte d'une alternative : opérateur ternaire)
L'opérateur ternaire est une expression qui fournit une valeur que l'on peut utiliser dans une affectation ou un calcul. Par exemple, pour trouver le minimum de deux nombres on peut utiliser l'opérateur ternaire :

```
x, y = 4, 3
# écriture classique
if x < y :
    plus_petit = x
else :
    plus_petit = y
print(f"Plus petit : {plus_petit}")
```

Plus petit : 3

```
x, y = 4, 3
# utilisation de l'opérateur ternaire
plus_petit = x if x < y else y
print(f"Plus petit : {plus_petit}")
```

Plus petit : 3

Avec l'opérateur ternaire on ne peut pas utiliser `elif`. Il faudra alors imbriquer un autre opérateur ternaire :

$$y = \begin{cases} x^2 & \text{si } x < 10 \\ x & \text{si } 10 \le x < 20 \\ 1 & \text{sinon.} \end{cases}$$

```
# écriture classique
if x < 10 :
    y = x**2
elif x<20 :
    y = x
else :
    y = 1
```

```
# utilisation de l'opérateur ternaire
y = x**2 if x < 10 else ( x if x<20 else 1 )
```

Parfois nous pouvons éviter le `if` explicite : `y = x**2 * (x < 10) +  x * (10<=x<20) + 1 * (x>=20)`

☆ Traitement des erreurs — les exceptions

Afin de rendre les applications plus robustes, il est nécessaire de gérer les erreurs d'exécution des parties sensibles du code. Le mécanisme des exceptions sépare d'un côté la séquence d'instructions à exécuter lorsque tout se passe bien et, d'un autre côté, une ou plusieurs séquences d'instructions à exécuter en cas d'erreur. Lorsqu'une erreur survient, un objet exception est passé au mécanisme de propagation des exceptions, et l'exécution est transférée à la séquence de traitement ad hoc. Ce n'est pas exactement une structure de test mais elle s'en rapproche.

La syntaxe complète est la suivante : la séquence normale d'instructions est placée dans un bloc `try`. Si une erreur est détectée (levée d'exception), elle est traitée dans le bloc `except` approprié (le gestionnaire d'exception).

```
try:
    # commandes
except:
    # traitement des erreurs
finaly:
    # commandes à exécuter dans tous les cas
```

Exemple :

```
a = 1
b = 0
try:
    c = a/b
    print(c)
except (ZeroDivisionError):
    print('Vous ne pouvez pas diviser par 0')
except:
    print('il y a une autre erreur')
```

```
Vous ne pouvez pas diviser par 0
```

```
a = 1
b = 'toto'
try:
    c = a/b
    print(c)
except (ZeroDivisionError):
    print('Vous ne pouvez pas diviser par 0')
except:
    print('il y a une autre erreur')
```

```
il y a une autre erreur
```

 © G. Faccanoni

3.2. Exercices

Exercice 3.1 (Devine le résultat)

Quel résultat donnent les codes suivants ? Après avoir écrit votre réponse, vérifiez-là avec l'ordinateur.

Cas 1 :

```
a = 2
b = 4
if b>10:
    b = a*b
    a = b
c = a+b
print(a,b,c)
```

Cas 2 :

```
a = 2
b = 10
if b>10:
    b = a*b
    a = b
c = a+b
print(a,b,c)
```

Cas 3 :

```
a = 2
b = 13
if b>10:
    b = a*b
    a = b
c = a+b
print(a,b,c)
```

Cas 4 :

```
a = 2
b = 4
if b>10:
    b = a*b
else:
    a = b
c = a+b
print(a,b,c)
```

Cas 5 :

```
a = 2
b = 13
if b>10:
    b = a*b
else:
    a = b
c = a+b
print(a,b,c)
```

Cas 6 :

```
a = 2
b = 4
if b>10:
    b = a*b
a = b
c = a+b
print(a,b,c)
```

Correction

Cas 1 : 2 4 6 Cas 2 : 2 10 12 Cas 3 : 26 26 52 Cas 4 : 4 4 8 Cas 5 : 2 26 28 Cas 6 : 4 4 8

Exercice 3.2 (Blanche Neige)

Soient a, b et $c \in \mathbb{N}$ et considérons le code

```
if a<b<c:
    if 2*a<b:
        s = "Prof"
    else:
        s = "Timide"
    if 2*c<b:
        s = "Atchoum"
else:
    if a<b:
        s = "Joyeux"
    if a<c: # ♡
        s = "Simplet"
    elif b<c:
        s = "Dormeur"
    else:
        s = "Grincheux"
print(s)
```

1. Quel résultat obtient-on dans les 5 cas suivants :

Cas 1 : $a = 1, b = 1, c = 1$
Cas 2 : $a = 2, b = 1, c = 2$
Cas 3 : $a = 4, b = 5, c = 2$
Cas 4 : $a = 1, b = 4, c = 7$
Cas 5 : $a = 4, b = 5, c = 6$

2. Trouver, s'il existe, un triplet $(a, b, c) \in \mathbb{N}^3$ tel que le code affichera `Prof` et `Timide` en même temps.
3. Trouver, s'il existe, un triplet $(a, b, c) \in \mathbb{N}^3$ tel que le code affichera `Atchoum`.
4. Même question pour `Simplet`.

Correction

1. Notons que $a < b < c$ équivaut à "$a < b$ AND $b < c$". Sa négation (cas `else` de la ligne 8) est donc "$a \geq b$ OR $b \geq c$".
   Cas 1 : (a,b,c)= (1, 1, 1) Grincheux
   Cas 2 : (a,b,c)= (2, 1, 2) Dormeur
   Cas 3 : (a,b,c)= (4, 5, 2) Joyeux Grincheux
   Cas 4 : (a,b,c)= (1, 4, 7) Prof
   Cas 5 : (a,b,c)= (4, 5, 6) Timide
   Notons que, si à la ligne ♡ on avait écrit `elif` au lieu de `if`, dans le cas 3 on aurait obtenu juste "Joyeux".
   Plus généralement, `Joyeux` sera toujours suivi d'un parmi `Simplet`, `Dormeur` ou `Grincheux`. Ces trois en revanche ne pourrons jamais être affichés en même temps.
2. Il n'est pas possible d'afficher `Prof` et `Timide` en même temps car pour afficher `Prof` il faut choisir a, b, c tels que $2a < b$ et pour afficher `Timide` il faut choisir a, b, c tels que $2a \geq b$.
3. Pour afficher `Atchoum` il faut que $a < b < c$ et $2c < b$, donc $b < c$ et $b > 2c$ ce qui est impossible si $b, c \geq 0$.
4. Pour afficher `Simplet` il faut que les deux conditions soient satisfaites :
$$\begin{cases} a \geq b \text{ OU } b \geq c \\ a < c \end{cases}$$
   ce qui correspond à
$$\begin{cases} a \geq b \\ a < c \end{cases} \quad \text{OU} \quad \begin{cases} b \geq c \\ a < c \end{cases}$$
   soit encore
$$b \leq a < c \quad \text{OU} \quad a < c \leq b$$
   Nous cherchons donc un triplet a, b, c tel que soit $b \leq a < c$ soit $a < c \leq b$. Notons que dans le cas $a < c \leq b$ on affichera aussi `Joyeux`. Voici deux exemples :
   Le triplet (a,b,c)= (1, 1, 3) donne Simplet
   Le triplet (a,b,c)= (1, 2, 2) donne Joyeux Simplet

Exercice 3.3 (Température)

Écrire un script qui, pour une température T donnée, affiche l'état de l'eau à cette température, c'est à dire `"SOLIDE"`, `"LIQUIDE"` ou `"GAZEUX"`. On prendra comme conditions les suivantes :

- si la température est strictement négative alors l'eau est à l'état solide,
- si la température est entre 0 et 100 (compris) l'eau est à l'état liquide,
- si la température est strictement supérieure à 100.

Correction

```
if T<0:
    s = "SOLIDE"
elif T<=100:
    s = "LIQUIDE"
else:
    s = "GAZEUX"
print(s)
```

 © G. Faccanoni

Versions abrégées :

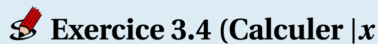
```
print( "SOLIDE" if T<0 else ( "LIQUIDE" if T<=100 else "GAZEUX" ) )
print( "SOLIDE"*(T<0) + "LIQUIDE"*(0<=T<=100) + "GAZEUX"*(T>100) )
```

Testons le code :

Si T=-2 l'eau est à l'état SOLIDE

Si T=10 l'eau est à l'état LIQUIDE

Si T=110 l'eau est à l'état GAZEUX



Exercice 3.4 (Calculer $|x|$)

Afficher la valeur absolue d'un nombre x sans utiliser la fonction `abs`.

Correction

Idée 1 : On peut bien sur écrire

```
if x>=0:
    ax = x
else:
    ax = -x
print(f"|{x}| = {ax}")
```

Idée 2 : En réalité il suffit de changer x avec $-x$ si $x<0$:

```
ax = x
if x<0:
    ax = -x
print(f"|{x}| = {ax}")
```

qu'on peut réécrire

```
ax = -x if x<0 else x
print(f"|{x}| = {ax}")
```

Idée 3 : Ou encore, sans utiliser de `if` explicite :

```
ax = (-x)*(x<0) + (x)*(x>=0)
print(f"|{x}| = {ax}")
```

Exercice 3.5 (Indice IMC)

Écrire un script qui, connaissant la taille (en mètres) et la masse (en kg) d'un individu, lui renvoie son IMC (indice de masse corporelle) avec un petit commentaire :

- si l'indice IMC est inférieur à 25, le commentaire pourra être : «vous n'êtes pas en surpoids»
- sinon le commentaire pourra être : «vous êtes en surpoids».

Cet algorithme utilisera 3 variables : `masse`, `taille` et `IMC`=$\frac{\texttt{masse}}{\texttt{taille}\times\texttt{taille}}$.

Pour valider le script, tester avec différentes valeurs en s'assurant de tester chaque cas possible.

Correction

```
imc = masse/taille**2
if imc<25:
    print(f"Votre IMC est égal à {imc:.1f}, vous n'êtes pas en surpoids")
else:
    print(f"Attention, votre IMC est égal à {imc:.1f}, vous êtes en surpoids")
```

Voici deux exemples :

masse = 60, taille = 1.6 Votre IMC est égal à 23.4, vous n'êtes pas en surpoids

masse = 88, taille = 1.6 Attention, votre IMC est égal à 34.4, vous êtes en surpoids

Exercice 3.6 (Note ECUE)

Soit `CT, CC, TP` respectivement les notes de contrôle terminal, de contrôle continue et de travaux pratiques d'un ECUE. La note finale est calculée selon la formule

$$0.3\text{TP} + \max\{0.7\text{CT}; 0.5\text{CT} + 0.2\text{CC}\}$$

Écrire un script qui calcule la note finale dans les cas suivants (vérifier les résultats !) **sans utiliser la fonction** `max` :

1. TP=10, CT=10, CC=10 ;
2. TP=10, CT=10, CC=20 ;
3. TP=10, CT=20, CC=10 ;
4. TP=20, CT=10, CC=10 ;
5. TP=20, CT=10, CC=20 ;
6. TP=20, CT=20, CC=10.

Correction

Notons que $\max\{0.7\text{CT}; 0.5\text{CT} + 0.2\text{CC}\} = \max\{0.5\text{CT} + 0.2\text{CT}; 0.5\text{CT} + 0.2\text{CC}\} = 0.5\text{CT} + 0.2\max\{\text{CT}; \text{CC}\}$. On peut alors écrire

```
if CT>=CC:
    note = 0.3*TP+0.7*CT
else:
    note = 0.3*TP+0.5*CT+0.2*CC
print(f"Note : {note}")
```

Cas 1 : (TP,CT,CC)= (10, 10, 10) Note : 10.0

Cas 2 : (TP,CT,CC)= (10, 10, 20) Note : 12.0

Cas 3 : (TP,CT,CC)= (10, 20, 10) Note : 17.0

Cas 4 : (TP,CT,CC)= (20, 10, 10) Note : 13.0

Cas 5 : (TP,CT,CC)= (20, 10, 20) Note : 15.0

Cas 6 : (TP,CT,CC)= (20, 20, 10) Note : 20.0

Version abrégée :

```
print( 0.3*TP + 0.5*CT + 0.2*( CT*(CT>=CC) + CC*(CT<CC) ) )
```

Exercice 3.7 (Triangles)

Écrire un script qui, étant donnés trois nombres réels positifs a, b, c correspondant aux longueurs des trois cotés d'un triangle, affiche le type de triangle dont il s'agit parmi équilatéral, isocèle et quelconque. Puis il affiche si le triangle est rectangle. Tester le script dans les cas suivants (dont on connaît la solution) :

1. $a = 1$, $b = 2$, $c = 3$, (quelconque)
2. $a = 1$, $b = 1$, $c = 2$, (isocèle)
3. $a = 1$, $b = 1$, $c = 1$, (équilatéral)
4. $a = 1$, $b = 0$, $c = -1$, (erreur de saisie)
5. $a = 3$, $b = 4$, $c = 5$, (quelconque, rectangle)
6. $a = 1$, $b = 1$, $c = 2^{1/2}$, (isocèle rectangle) ← attention à la comparaison entre `float`.

Nota bene : au lieu d'écrire `x==y` on utilisera `abs(x-y)<1.e-10` pour éviter des erreurs dues aux arrondis (*cf.* annexe A).

Correction

Attention à l'ordre dans lequel on écrit la condition : de la plus restrictive (non existence d'un triangle) à la moins restrictive (être équilatéral puis être isocèle). Si on inverse l'ordre, on n'obtiendra jamais de triangle équilatéral car, étant aussi isocèle, la première condition sera satisfaite et on n'entrera jamais dans la deuxième.

Pour éviter de considérer tous les sous cas, on ordonne les cotés dès le début.

```
A,B,C = sorted([a,b,c])

if A<=0:
    print("erreur de saisie")
    continue
```

```python
elif A==B==C : # mieux abs(A-B)<1.e-14 and abs(B-C)<1.e-14
    print("équilatéral")
elif A==B or B==C :  # mieux abs(A-B)<1.e-14 or abs(B-C)<1.e-14
    print("isocèle")
else :
    print("quelconque")

if A**2+B**2==C**2: # mieux abs(A**2+B**2-C**2)<1.e-14
    print(" rectangle")
```

Cas 1 quelconque

Cas 2 isocèle

Cas 3 équilatéral

Cas 4 erreur de saisie

Cas 5 quelconque rectangle

Cas 6 isocèle rectangle

Exercice 3.8 ($ax^2 + bx + c = 0$)

Écrire un script qui, étant donnés trois nombres réels a, b, c, détermine, stocke dans la variable `racines` et affiche la ou les solutions réelles (si elles existent) de l'équation du second degré $ax^2 + bx + c$. Cette variable est constituée de

- un tuple si les racines sont réelles et distinctes,
- un tuple avec un seul élément si la racine est unique,
- un tuple vide si les racines sont complexes conjuguées.

Tester le script dans les cas suivants (dont on connaît la solution) :

1. $a = 1$, $b = 0$, $c = -4$
2. $a = 1$, $b = 4$, $c = 4$
3. $a = 1$, $b = 0$, $c = 4$
4. $a = 0$, $b = 1$, $c = 2$
5. $a = 0$, $b = 0$, $c = 3$

Pour calculer la racine carrée d'un nombre p on utilisera la propriété $\sqrt{p} = p^{\frac{1}{2}}$, *e.g.* au lieu d'écrire `sqrt(p)` on écrira `p**0.5`.

Correction

```python
if a==0:
    if b!=0:
        racines = (-c/b,) # bien noter la virgule pour un tuple avec un singleton
    else:
        racines = ()
else:
    d = b**2-4*a*c
    if d>0 :
        racines = ( (-b-d**0.5)/(2*a), (-b+d**0.5)/(2*a) )
    elif d<0 :
        racines = ()
    else :
        racines = (-b/(2*a),)
print(racines)
```

Tests :

Si (a,b,c) = (1, 0, -4) alors racines = (-2.0, 2.0)

Si (a,b,c) = (1, 4, 4) alors racines = (-2.0,)

Si (a,b,c) = (1, 0, 4) alors racines = ()

Si (a,b,c) = (0, 1, 2) alors racines = (-2.0,)

Si (a,b,c) = (0, 0, 3) alors racines = ()

☆ Exercice Bonus 3.9 (Pierre Feuille Ciseaux)

Écrire un script où le joueur entre un mot parmi “pierre”, “feuille”, “ciseaux”, puis l’ordinateur choisit au hasard un de ces mots et il affiche le résultat (“Perdu”, “Gagné”, “Égalité”).

Pour que l’ordinateur choisisse aléatoirement on écrira

```
import random
valide = ["pierre", "feuille", "ciseaux"]
cpu = random.choice(valide)
```

Pour affecter à la variable "`user`" le mot que le joueur tape au clavier on écrira
`user = input("écrit ton choix: ")`

Correction

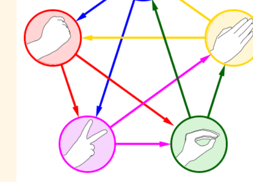
```
import random
valide = ["pierre", "feuille", "ciseaux"]
user = input("Écrit ton choix: ")
if user not in valide:
    print("input incorrect")
else:
    cpu = random.choice(valide)
    print (f"Choix cpu: {cpu}")
    if cpu == user:
        print ("Égalité")
    elif cpu == "pierre":
        print( "Tu as gagné") if user == "feuille" else print( "Tu as perdu")
    elif cpu == "ciseaux":
        print ("Tu as gagné") if user == "pierre" else print( "Tu as perdu")
    else:
        print ("Tu as gagné") if user =="ciseaux" else print( "Tu as perdu")
```

☆ Exercice Bonus 3.10 (Pierre, feuille, ciseaux, lézard, Spock)

Une variante de “Pierre feuille ciseaux” a été créée et révélée par la série américaine The Big Bang Theory et a obtenu pas mal de popularité. Créé à l’origine par Sam KASS et Karen BRYLA, elle est largement utilisé par Sheldon, Leonard, Howard et Raj. [a]

Voici les règles et toutes les combinaisons permettant de remporter (ou perdre) la partie :

- la pierre casse les ciseaux et écrase le lézard;
- les ciseaux décapitent le lézard et coupent la feuille;
- le lézard mange la feuille et empoisonne Spock;
- la feuille désapprouve Spock et recouvre la pierre;
- Spock vaporise la pierre et écrabouille les ciseaux.



Écrire un script qui joue à cette variante comme pour l’exercice 3.9.

a. Un article fort intéressant sur la construction d’autres variantes http://eljjdx.canalblog.com/archives/2015/10/21/32803533.html
Un autre article : https://info.blaisepascal.fr/nsi-chifoumi

Correction

```
import random
valide = ["pierre", "feuille", "ciseaux", "lezard", "Spock"]
user = input("Écrit ton choix: ")
if user not in valide:
```

```
    print("input incorrect")
else:
    cpu = random.choice(valide)
    print (f"Choix cpu: {cpu}")
    if user == cpu:
        print ("Égalité")
    elif user == "pierre":
        print( "Tu as gagné") if cpu == "ciseaux" or cpu=="Spock" else print( "Tu as perdu")
    elif user == "ciseaux":
        print( "Tu as gagné") if cpu == "lezard" or cpu=="feuille" else print( "Tu as perdu")
    elif user == "lezard":
        print( "Tu as gagné") if cpu == "feuille" or cpu=="Spock" else print( "Tu as perdu")
    elif user == "feuille":
        print( "Tu as gagné") if cpu == "Spock" or cpu=="pierre" else print( "Tu as perdu")
    else:
        print( "Tu as gagné") if cpu == "pierre" or cpu=="ciseaux" else print( "Tu as perdu")
```

CHAPITRE 4

Structures itératives

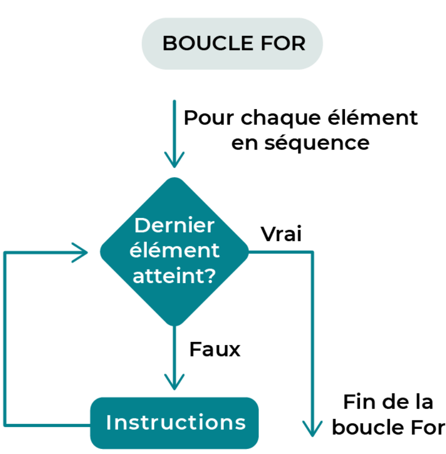
En programmation, il y a des ensembles d'instructions qu'il faut répéter plusieurs fois. Quand on a besoin de répéter un ensemble d'instructions, parfois on sait combien de fois on doit le répéter, d'autres fois on ne le sait pas à priori, on ait juste qu'il faut répéter le code jusqu'à ce qu'une certaine condition soit remplie. Pour tous ces usages, on va utiliser des boucles.

Les structures de répétition se classent en deux catégories : les *répétitions inconditionnelles* pour lesquelles le bloc d'instructions est à répéter un nombre donné de fois et les *répétitions conditionnelles* pour lesquelles le bloc d'instructions est à répéter autant de fois qu'une condition est vérifiée.

Pour comprendre l'exécution d'un code pas à pas on pourra utiliser : Visualize code and get live help
http://pythontutor.com/visualize.html

4.1. Répétition `for` : boucle inconditionnelle (parcourir)

Lorsque l'on souhaite répéter un bloc d'instructions un nombre déterminé de fois, on peut utiliser un *compteur actif*, c'est-à-dire une variable qui compte le nombre de répétitions et conditionne la sortie de la boucle.



C'est la structure introduite par le mot-clé `for` qui a la forme générale suivante (attention à l'**indentation** et aux **deux points**) :

```
for target in sequence:
    instruction_1
    instruction_2
    ...
```

où `target` est le *compteur actif* et `sequence` est un itérateur (souvent généré par la fonction `range` ou une liste, un tuple, une chaîne de caractères, un dictionnaire). Tant que `target` appartient à `sequence`, on exécute les instructions `instruction_i`.

Quelques exemples où `sequence` est une liste, une chaîne de caractères ou l'itérateur `range` :

```
L = ["Qui", "Quo", "Qua"]
for truc in L:
    print(truc)
```

```
Qui
Quo
Qua
```

```
L = "Minnie"
for c in L:
    print(c)
```

```
M
i
n
n
i
e
```

```
for n in range(5):
    print(n)
```

```
0
1
2
3
4
```

Il est possible d'imbriquer des boucles, c'est-à-dire que dans le bloc d'une boucle, on utilise une nouvelle boucle.

```
for x in [10,20,30,40]:
    for y in [3,7]:
        print(x+y,end=", ")
```

Dans ce petit programme x vaut d'abord 10, y prend la valeur 3 puis la valeur 7 (le programme affiche donc d'abord 13, puis 17). Ensuite $x = 20$ et y vaut de nouveau 3 puis 7 (le programme affiche donc ensuite 23, puis 27). Au final le programme affiche :

```
13, 17, 23, 27, 33, 37, 43, 47,
```

On peut utiliser une boucle pour ajouter des éléments à une liste avec la méthode `append`. Voici un exemple :

Création de la liste

$$\left[1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\right]$$

```
L = [] # liste vide
for n in range(4):
    L.append(1/2**n)
print(L)
```

```
[1.0, 0.5, 0.25, 0.125]
```

`enumerate` Pour **accéder en même temps à la position et au contenu** on utilisera `enumerate(liste)`.

```
L = ["Riri", "Fifi", "Loulou"]
for idx,item in enumerate(L):
    print(item, idx)
```

```
Riri 0
Fifi 1
Loulou 2
```

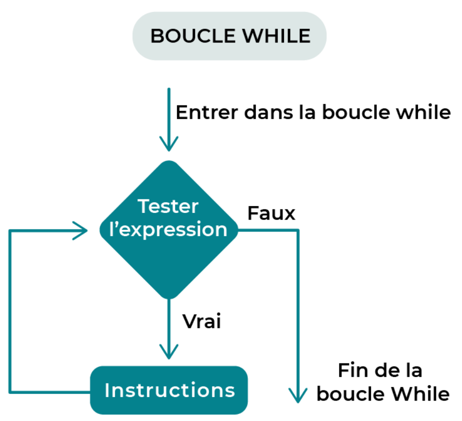
On peut modifier la valeur de départ de `enumerate` :

```
L = ["Riri", "Fifi", "Loulou"]
for idx,item in enumerate(L,start=101):
    print(item, idx)
```

```
Riri 101
Fifi 102
Loulou 103
```

4.2. Boucle `while` : répétition conditionnelle

La boucle `for` permet d'exécuter du code un nombre spécifique de fois, alors que la boucle `while` continue de s'exécuter jusqu'à ce qu'une certaine condition soit remplie.



`While` est la traduction de "tant que…". Concrètement, la boucle s'exécutera tant qu'une condition est remplie (donc tant qu'elle renverra la valeur `True`).

Le constructeur `while` a la forme générale suivante (attention à l'**indentation** et aux **deux points**) :

```
while condition:
    instruction_1
    instruction_2
    ...
```

où `condition` représente des ensembles d'instructions dont la valeur est `True` ou `False`. Tant que la condition `condition` a la valeur `True`, on exécute les instructions `instruction_i`.

Attention
Si la condition ne devient jamais fausse, le bloc d'instructions est répété indéfiniment et le programme ne se termine pas.

 © G. Faccanoni

- Affichage d'un compte à rebours. Tant que la condition $n > 0$ est vraie, on diminue n de 1. La dernière valeur affichée est $n = 1$ car ensuite $n = 0$ et la condition $n > 0$ devient fausse donc la boucle s'arrête.

```
n = 3
while n>0:
    print(n)
    n -= 1
print("Go!")
```

Output :

```
3
2
1
Go!
```

- Création de la liste

$$\left[1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\right]$$

```
n = 0
L = []
while n<4:
    L.append(1/2**n)
    n += 1
print(L)
```

```
[1.0, 0.5, 0.25, 0.125]
```

- On divise n par 2 tant qu'il est pair (cela revient à supprimer tous les facteurs 2 de l'entier n). Par exemple, si $n = 168 = 2^3 \times 3 \times 7$, le programme affichera 21.

```
n = 168
while n%2==0:
    n //= 2
print(n)
```

```
21
```

- On calcule la somme des n premiers entiers (et on vérifie qu'on a bien $n(n+1)/2$).

```
n = 100
s,i = 0,0
while i<n:
    i += 1
    s += i
print(s)
print(f'{n*(n+1)/2=}')
```

```
5050
n*(n+1)/2=5050.0
```

- Exemple classique de ce qu'on appelle une recherche de seuil : on recherche à partir de quelle valeur de n la somme $1+2+3+\cdots+n$ dépasse un million.

  Pour vérifier notre code, on peut afficher n, somme et aussi $n-1$ et somme-n.

```
n = 0
somme = 0
while somme < 10**6 :
    n += 1
    somme += n
print(f"{n=}, {somme=}")
print(f"{n-1=}, {somme-n=}")
```

```
n=1414, somme=1000405
n-1=1413, somme-n=998991
```

4.3. Ruptures de séquences

- Interrompre une boucle : `break`

  Sort immédiatement de la boucle `for` ou `while` en cours contenant l'instruction :

```
for i in [1,2,3,4,5]:
    if i ==4:
        break
    print(i, end=" ")
print("Boucle interrompue pour i =", i)
```

```
1 2 3 Boucle interrompue pour i = 4
```

  L'instruction `break` sort de la plus petite boucle `for` ou `while` englobante.

```
for lettre in 'AbcDefGhj':
    if lettre=='D':
        break
    print(lettre, end=" ")
```

```
A b c
```

- Court-circuiter une boucle : `continue`
  Passe immédiatement à l'itération suivante de la boucle `for` ou `while` en cours contenant l'instruction; reprend à la ligne de l'en-tête de la boucle :

```
for i in [1,2,3,4,5]:
    if i ==4:
        continue
    print(i, end=" ")
# la boucle a sauté la valeur 4
```

```
1 2 3 5
```

  L'instruction `continue` continue sur la prochaine itération de la boucle. Par exemple, supposons que nous voulions afficher $1/n$ pour n dans une liste. Si n vaut 0, le calcul $1/n$ sera impossible, il faudra donc sauter cette étape et passer au nombre suivant :

```
liste = [ -4, 2, 6, 0, 1, 3, 0, 10]
for n in liste:
    if n ==0:
        continue
    print(1/n, end=" ")
```

```
-0.25 0.5 0.16666666666666666 1.0
↪ 0.3333333333333333 0.1
```

  Autre exemple :

```
a=0
while a<=5:
    a+=1
    if a%2==0:
        continue
    print(a, end=" ")
print("Boucle terminée")
```

```
1 3 5 Boucle terminée
```

  Comparons les trois codes suivants :

```
for i in [1,2,3,4,5]:
    if i ==4:
        print("J'ai trouvé")
    print(i, end=" ")
```

```
1 2 3 J'ai trouvé
4 5
```

```
for i in [1,2,3,4,5]:
    if i ==4:
        print("J'ai trouvé")
        break
    print(i, end=" ")
```

```
1 2 3 J'ai trouvé
```

```
for i in [1,2,3,4,5]:
    if i ==4:
        print("J'ai trouvé")
        continue
    print(i, end=" ")
```

```
1 2 3 J'ai trouvé
5
```

- Utilisation avancée des boucles : la syntaxe complète des boucles autorise des utilisations plus rares.
  Les boucles `while` et `for` peuvent posséder une clause `else` qui ne s'exécute que si la boucle se termine normalement, c'est-à-dire sans interruption.
  Les instructions de boucle ont une clause `else` qui est exécutée lorsque la boucle se termine par épuisement de la liste (avec `for`) ou quand la condition devient fausse (avec `while`), mais pas quand la boucle est interrompue par une instruction `break` :

```
for item in items:
	if something(item):
		do_st(item)
		break
else: # else associé à for et non au if !!!
	print("Not found") # exécuté si la boucle se termine mais non exécuté si termine par
	↪ break
```

  - `while - else`

```
y = 10 # un entier positif
x = y // 2
```

```
while x > 1:
    if y % x == 0:
        print(y, "a pour facteur", x)
        break # voici l'interruption !
    x -= 1
else :
    print(y, "est premier.")
```

```
10 a pour facteur 5
```

- `for - else`
  Un exemple avec le parcours d'une liste :

```
L = [2, 5, 9, 7, 11]
cible = 5
for i in L :
    if i == cible :
        sauve = i
        break # voici l'interruption !
else : # else pour le for et non pas le if !
    print(cible, "n'est pas dans", L)
    sauve = None
# sauve vaut cible ou None
print("On obtient sauve =", sauve)
```

```
On obtient sauve = 5
```

Ceci est expliqué dans la boucle suivante, qui recherche des nombres premiers :

```
for n in range(2,10):
    for x in range(2,int(n/2)+1):
        if n%x==0:
            print(f"{n} est égale à {x}*{int(n/x)}")
            break
    else:
        print(f'{n} est un nombre premier')
```

```
2 est un nombre premier
3 est un nombre premier
4 est égale à 2*2
5 est un nombre premier
6 est égale à 2*3
7 est un nombre premier
8 est égale à 2*4
9 est égale à 3*3
```

4.4. Exercices

Exercice 4.1 (Devine le résultat - `for`)
Quel résultat donnent les codes suivants ? Après avoir écrit votre réponse, vérifiez-là avec l'ordinateur.

Cas 1 :
```
for n in range(5):
    print(n)
```

Cas 2 :
```
for n in range(2,8):
    print(n)
```

Cas 3 :
```
for n in range(2,8,2):
    print(n)
```

Cas 4 :
```
for n in range(10):
    if n%4==0:
        print(n)
```

Cas 5 :
```
for n in range(10):
    if n%4==0:
        print(n)
    elif n%2==0:
        print(2*n)
    else:
        None
```

Cas 6 :
```
for n in range(10):
    if n%2==0:
        print(2*n)
    elif n%4==0:
        print(n)
    else:
        None
```

Correction
Cas 1 : 0 1 2 3 4
Cas 2 : 2 3 4 5 6 7
Cas 3 : 2 4 6
Cas 4 : 0 4 8
Cas 5 : 0 4 4 12 8
Cas 6 : 0 4 8 12 16

Pour le cas 4, on aurait pu écrire simplement

```
for n in range(0,10,4):
    print(n)
```

Bien noter la différence entre le cas 5 et le cas 6 : dans ce dernier la partie correspondante à la condition `n%4==0` n'est jamais prise en compte car la condition `n%2==0` étant forcement vérifiée pour les mêmes cas, elle sera traitée avant. En règle générale, lors de l'écriture de conditions, toujours commencer par la condition la plus restrictive.

Exercice 4.2 (Les animaux sauvages - `for`)
Étant donné la liste

```
animaux_sauvages = ["aigle", "ours", "perroquet", "tigre", "pélican", "coyote"]
```

afficher

1. Tous les animaux d'abord sous forme de liste puis un par un à l'aide d'une boucle `for`,
2. Les mammifères en utilisant le découpage en tranches d'abord sous forme de liste puis un par un à l'aide d'une boucle `for`,
3. Oiseaux en utilisant le découpage en tranches d'abord sous forme de liste puis un par un à l'aide d'une boucle `for`,
4. Tous les animaux dans l'ordre inverse en utilisant le découpage en tranches d'abord sous forme de liste puis un par un à l'aide d'une boucle `for`.

Correction
```
animaux_sauvages = ["aigle", "ours", "perroquet", "tigre", "pélican", "coyote"]

print(animaux_sauvages)
for animal in animaux_sauvages:
    print(animal)
```

```
print(animaux_sauvages[1::2])
for mammifère in animaux_sauvages[1::2]:
    print(mammifère)

print(animaux_sauvages[::2])
for oiseau in animaux_sauvages[::2]:
    print(oiseau)

print(animaux_sauvages[::-1])
for animal in animaux_sauvages[::-1]:
    print(animal)
```

```
['aigle', 'ours', 'perroquet', 'tigre', 'pélican', 'coyote']
aigle
ours
perroquet
tigre
pélican
coyote
['ours', 'tigre', 'coyote']
ours
tigre
coyote
['aigle', 'perroquet', 'pélican']
aigle
perroquet
pélican
['coyote', 'pélican', 'tigre', 'perroquet', 'ours', 'aigle']
coyote
pélican
tigre
perroquet
ours
aigle
```

Attention

L'exercice 4.3 est le premier d'une longue liste d'exercices tirés des pydéfis, dont la liste complète est ici https://pydefis.callicode.fr/user/liste_defis. Il s'agit des énigmes des concours annuels c0d1ngUp http://codingup.fr/, un challenge de programmation, en temps limité et en présentiel, soutenu par l'Université de Poitiers. Durant une journée les participants font face à une série de problèmes qu'ils doivent résoudre par le moyen de leur choix. Les problèmes sont choisis pour que leur résolution nécessite de la programmation, de l'inventivité et de l'ingéniosité.

L'auteur aimerait que le site puisse continuer à être utilisé par des enseignants, avec leurs élèves. C'est impossible si les solutions toutes prêtes fleurissent. C'est pour cette raison que je ne publie pas les solutions même dans la version corrigée de ce polycopié. Pour vérifier vos réponse, vous pouvez me demander ou, encore mieux, vérifier directement sur le site des Pydéfis.

Exercice Bonus 4.3 (Pydéfis – L'algorithme du professeur Guique)

Pour dissimuler la combinaison à trois nombres de son coffre, le professeur Guique a eu l'idée de la cacher à l'intérieur d'un algorithme. Les trois nombres de la combinaison sont en effet les valeurs contenues dans les variables a, b et c après l'exécution de l'algorithme suivant :

```
Initialiser a, b, c, k et n respectivement a 1, 2, 5, 1 et 0
Répéter tant que k est strictement inférieur a 1000-n
    a = b
```

```
    b = c + a
    c = 3c + 4a - b
    n = a + b
    augmenter k de 1
fin répéter
```

Quelle est la séquence des trois nombres ouvrant le coffre du professeur Guique?

Source : https://pydefis.callicode.fr/defis/Algorithme/txt

Exercice 4.4 (Lower to Upper)

Considérons la liste d'acronymes

```
acronymes = ["asap", "faq", "fyi", "diy"]
```

Tous les acronymes sont en minuscules. Avec une boucle `for` les mettre en majuscules (en les remplaçant dans la même liste).

Correction

```
acronymes = ["asap", "faq", "diy"]
print(acronymes)
for i in range(len(acronymes)):
    acronymes[i] = acronymes[i].upper()
print(acronymes)
```

```
['asap', 'faq', 'diy']
['ASAP', 'FAQ', 'DIY']
```

Exercice 4.5 (Vente de voitures)

Vous travaillez pour un célèbre constructeur automobile et vous devez expédier les nouvelles voitures qui viennent d'arriver. Vos collègues ont indiqué les voitures destinées à la France, à l'Espagne et au Portugal, mais ils les ont mélangées :

```
cars = ["PT-754J", "ES-096L", "PT-536G", "FR-543H", "PT-653H"]
```

À l'aide d'une boucle `for`, dispatchez les trois groupes de voitures dans trois listes en fonction de leurs destinations.

Correction

```
cars = ["PT-754J", "ES-096L", "PT-536G", "FR-543H", "PT-653H"]
PT, ES, FR = [], [], []
for c in cars:
    if c[:2]=='PT':
        PT.append(c)
    elif c[:2]=='ES':
        ES.append(c)
    elif c[:2]=='FR':
        FR.append(c)
print(PT,ES,FR)
```

```
['PT-754J', 'PT-536G', 'PT-653H'] ['ES-096L'] ['FR-543H']
```

Exercice 4.6 (Triangles)

Créer des scripts qui dessinent des triangles comme les suivants :

```
x           xxxxxxxx           x    xxxxxxxx
xx          xxxxxxx           xx     xxxxxxx
xxx         xxxxxx           xxx      xxxxxx
xxxx        xxxxx           xxxx       xxxxx
xxxxx       xxxx           xxxxx        xxxx
xxxxxx      xxx           xxxxxx         xxx
xxxxxxx     xx           xxxxxxx          xx
xxxxxxxx    x           xxxxxxxx           x
```

Correction

```
for n in range(9):
    print('x'*n)

for n in range(8,0,-1):
    print('x'*n)
```

```
for n in range(9):
    print(' '*(8-n)+'x'*n)

for n in range(8,0,-1):
    print(' '*(8-n)+'x'*n)
```

Exercice 4.7 (Nombre de voyelles)

Pour un texte donné (sans accent), créer un programme qui affiche le nombre de voyelles et de consonnes.

Correction

Considérons l'exemple suivant :

```
texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
v,p,c = 0,0,0
for lettre in texte.lower():
    if lettre in "aeiouy":
        v += 1
    elif lettre in " ,:;.!?":
        p += 1
    else:
        c += 1

print(f"Voyelles={v}, Consonnes={c}")
```

```
Voyelles=19, Consonnes=28
```

Exercice 4.8 (Cartes de jeux)

On considère les deux listes suivantes :

- `couleurs = ['pique','coeur','carreau','trefle']`
- `valeurs = ['7','8','9','10','valet','reine','roi','as']`

Écrire un script qui affiche toutes les cartes d'un jeu de 32 cartes.

Correction

```
couleurs = ['pique','coeur','carreau','trefle']
valeurs = ['7','8','9','10','valet','reine','roi','as']
for c in couleurs:
    for v in valeurs:
        print(v, c)
```

7 pique	8 pique	9 pique	10 pique	valet pique	reine pique	roi pique	as pique
7 coeur	8 coeur	9 coeur	10 coeur	valet coeur	reine coeur	roi coeur	as coeur
7 carreau	8 carreau	9 carreau	10 carreau	valet carreau	reine carreau	roi carreau	as carreau
7 trefle	8 trefle	9 trefle	10 trefle	valet trefle	reine trefle	roi trefle	as trefle

 © G. Faccanoni

Exercice 4.9 (Tables de multiplication)

Afficher les tables de multiplication entre 1 et 9. Voici un exemple de ligne à afficher : `7 x 9 = 63`.

On verra à l'exercice 5.26 une autre présentation des tables de multiplication sous la forme du tableau de Pythagore.

Correction

```python
for b in range(1,10):
    for a in range(1,10):
        print(f'{a} x {b} = {a*b}')
    print('')
```

1 x 1 = 1	1 x 2 = 2	1 x 3 = 3	1 x 4 = 4	1 x 5 = 5	1 x 6 = 6	1 x 7 = 7	1 x 8 = 8	1 x 9 = 9
2 x 1 = 2	2 x 2 = 4	2 x 3 = 6	2 x 4 = 8	2 x 5 = 10	2 x 6 = 12	2 x 7 = 14	2 x 8 = 16	2 x 9 = 18
3 x 1 = 3	3 x 2 = 6	3 x 3 = 9	3 x 4 = 12	3 x 5 = 15	3 x 6 = 18	3 x 7 = 21	3 x 8 = 24	3 x 9 = 27
4 x 1 = 4	4 x 2 = 8	4 x 3 = 12	4 x 4 = 16	4 x 5 = 20	4 x 6 = 24	4 x 7 = 28	4 x 8 = 32	4 x 9 = 36
5 x 1 = 5	5 x 2 = 10	5 x 3 = 15	5 x 4 = 20	5 x 5 = 25	5 x 6 = 30	5 x 7 = 35	5 x 8 = 40	5 x 9 = 45
6 x 1 = 6	6 x 2 = 12	6 x 3 = 18	6 x 4 = 24	6 x 5 = 30	6 x 6 = 36	6 x 7 = 42	6 x 8 = 48	6 x 9 = 54
7 x 1 = 7	7 x 2 = 14	7 x 3 = 21	7 x 4 = 28	7 x 5 = 35	7 x 6 = 42	7 x 7 = 49	7 x 8 = 56	7 x 9 = 63
8 x 1 = 8	8 x 2 = 16	8 x 3 = 24	8 x 4 = 32	8 x 5 = 40	8 x 6 = 48	8 x 7 = 56	8 x 8 = 64	8 x 9 = 72
9 x 1 = 9	9 x 2 = 18	9 x 3 = 27	9 x 4 = 36	9 x 5 = 45	9 x 6 = 54	9 x 7 = 63	9 x 8 = 72	9 x 9 = 81

Exercice 4.10 (Parcourir deux listes)

On considère les deux listes suivantes (les deux listes ont la même taille) :

```python
a = [8468, 4560, 3941, 3328, 7, 9910, 9208, 8400, 6502, 1076, 5921, 6720, 948, 9561, 7391,
 ↪ 7745, 9007, 9707, 4370, 9636, 5265, 2638, 8919, 7814, 5142, 1060, 6971, 4065, 4629,
 ↪ 4490, 2480, 9180, 5623, 6600, 1764, 9846, 7605, 8271, 4681, 2818, 832, 5280, 3170,
 ↪ 8965, 4332, 3198, 9454, 2025, 1608, 4067]

b = [9093, 2559, 9664, 8075, 4525, 5847, 67, 8932, 5049, 5241, 5886, 1393, 9413, 8872,
 ↪ 2560, 4636, 9004, 7586, 1461, 350, 2627, 2187, 7778, 8933, 351, 7097, 356, 4110, 1393,
 ↪ 4864, 1088, 3904, 5623, 8040, 7273, 1114, 4394, 4108, 7123, 8001, 5715, 7215, 7460,
 ↪ 5829, 9513, 1256, 4052, 1585, 1608, 3941]
```

Trouvez le(s) nombre(s) qui sont exactement à la même place dans la liste `a` et dans la liste `b`.

Correction

```python
for i in range(len(a)):
    if a[i]==b[i]:
        print(f"En position {i} on a a[{i}]={a[i]} et b[{i}]={b[i]}.")
```

```
En position 32 on a a[32]=5623 et b[32]=5623.
En position 48 on a a[48]=1608 et b[48]=1608.
```

Remarque : il existe une fonction spécifique qui parcourt deux (ou plus) listes/tuples/string et génère un tuple : la fonction `zip(liste0,liste1,...)`.

```python
for i, (ai,bi) in enumerate(zip(a,b)):
    if ai==bi:
        print(f"En position {i} on a a[{i}]={ai} et b[{i}]={bi}.")
```

```
En position 32 on a a[32]=5623 et b[32]=5623.
En position 48 on a a[48]=1608 et b[48]=1608.
```

Exercice 4.11 (Parcourir deux chaînes de caractères)

On considère des mots à trous : ce sont des chaînes de caractères contenant uniquement des majuscules et des caractères "`*`". Par exemple "`INFO*MA*IQUE`", "`***I***E**`" et "`*S*`" sont des mots à trous.

Pour deux chaînes de caractères `mot` et `mot_a_trous` données, afficher `True` si on peut obtenir `mot` en remplaçant

convenablement les caractères “*” de `mot_a_trous`, `False` sinon.

Correction
Sujet 38.1 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

```
mot, mot_a_trou = "INFORMATIQUE", "INFO*MA*IQUE"
# mot, mot_a_trou = "AUTOMATIQUE", "INFO*MA*IQUE"
# mot, mot_a_trou = "BOL", "V*L"
# mot, mot_a_trou = "AUTO", "*UT*"

reponse = True
if len(mot) != len(mot_a_trou):
    reponse = False
else:
    for i in range(len(mot)):
        if mot_a_trou[i]!='*' and mot_a_trou[i]!=mot[i]:
            reponse = False

print(reponse)
```

```
True
```

Remarque : il existe une fonction spécifique qui parcourt deux (ou plus) listes/tuples/string et génère un tuple : la fonction `zip`.

```
reponse = True
for c,t in zip(mot,mot_a_trou):
    if len(mot)!=len(mot_a_trou) or (t!='*' and c!=t):
        reponse = False

print(reponse)
```

```
True
```

Exercice 4.12 (Devine le résultat)

```
for i in range(3):
    print(f"{i} - Je ne dois pas poser une question sans lever la main")
```

et

```
i = 0
while i < 3:
    print(f"{i} - Je ne dois pas poser une question sans lever la main")
    i = i +1
```

Correction
Les deux codes donnés affichent le même résultat :

```
0 - Je ne dois pas poser une question sans lever la main
1 - Je ne dois pas poser une question sans lever la main
2 - Je ne dois pas poser une question sans lever la main
```

En effet, les deux codes suivants donnent le même résultat :

```
for i in range(a,b,c):
    print(i)
```

```
i = a
while i<b:
    print(i)
    i += c
```

 © G. Faccanoni

Exercice 4.13 (Devine le résultat - `while`)
Quel résultat donnent les codes suivants ? Après avoir écrit votre réponse, vérifiez-là avec l'ordinateur.

Cas 1 :
```
n = 1
while n<5:
    print(n)
    n += 1
```

Cas 2 :
```
n = 1
while n<5:
    n += 1
    print(n)
```

Cas 3 :
```
n = 1
while (n<10) and
↪ (n<5):
    print(n)
    n += 1
```

Cas 4 :
```
n = 1
while (n<10) or (n<5):
    print(n)
    n += 1
```

Cas 5 :
```
n = 1
while 5<n<10:
    print(n)
    n += 1
```

Cas 6 :
```
n = 6
while 5<n<10:
    print(n)
    n += 1
```

Cas 7 :
```
n = 9
while 5<n<10:
    print(n)
    n-=1
```

Cas 8 :
```
i = 0
n = 0
while n<10:
    print(i,n)
    i += 1
    n = i*i
```

Cas 9 :
```
i = 0
n = 0
while n<10:
    print(i,n)
    n = i*i
    i += 1
```

Correction
Notons que "`(n<10) and (n<5)`" équivaut à "`(n<5)`" tandis que "`(n<10) or (n<5)`" équivaut à "`(n<10)`".

Cas 1 : 1 2 3 4
Cas 2 : 2 3 4 5
Cas 3 : 1 2 3 4
Cas 4 : 1 2 3 4 5 6 7 8 9
Cas 5 :
Cas 6 : 6 7 8 9
Cas 7 : 9 8 7 6
Cas 8 : 0 0 1 1 2 4 3 9
Cas 9 : 0 0 1 0 2 1 3 4 4 9

Exercice 4.14 (Suites type ① : $u_n = f(n)$, recherche de seuil (plus petit n tel que...))

1. Écrire un script qui affiche le plus petit entier n tel que $(n+1)(n+3) \geq 12\,345$.
2. Écrire un script qui affiche le plus petit entier n tel que $4+5+6+\cdots+n \geq 12\,345$.
3. Écrire un script qui affiche le plus petit entier n tel que $1^2+2^2+3^2+\cdots+n^2 \geq 12\,345$.

Dans ce type d'exercice on ne peut pas utiliser une boucle `for` car on ne sait pas à priori combien de répétitions il faudra faire. C'est donc une boucle `while` qu'il faut utiliser : ne pas oublier d'initialiser le compteur avant la boucle et de le modifier à chaque répétition. On continuera la boucle **tant que le critère n'est pas satisfait** ; par exemple, si on cherche le plus petit n tel que $f(n) > s$, il faudra répéter les calculs tant que $f(n) \leq s$ car la négation de $>$ est $\leq$ etc.

Correction

1.
```
n = 0
while (n+1)*(n+3)<12345:
    n += 1
print(f"Le plus petit entier tel que (n+1)*(n+3)>=12345 est n = {n}. En effet:")
# VALIDATION
print(f"si n = {n} alors (n+1)*(n+3)={(n+1)*(n+3)}")
m = n-1
print(f"si n = {m} alors (n+1)*(n+3)={(m+1)*(m+3)}")
```

```
Le plus petit entier tel que (n+1)*(n+3)>=12345 est n = 110. En effet:
si n = 110 alors (n+1)*(n+3)=12543
si n = 109 alors (n+1)*(n+3)=12320
```

Dans ce cas, on peut calculer $n \in \mathbb{N}$ analytiquement :

$$n(n+2) \le 12345 < (n+1)(n+3) \iff n^2+2n \le 12345 < n^2+4n+3 \iff \begin{cases} n^2+2n-12345 \le 0, \\ n^2+4n-12342 > 0 \end{cases}$$
$$\iff \begin{cases} n \le \frac{-2+\sqrt{4+4\times 12345}}{2} = \frac{-2+\sqrt{49384}}{2} \approx 110.11255, \\ n > \frac{-4+\sqrt{16+4\times 12342}}{2} = \frac{-4+\sqrt{49384}}{2} \approx 109.11255. \end{cases}$$

2.
```
n = 4
somme = n
while somme < 12345 :
    n += 1
    somme += n
print(f"n = {n}")
print(f"""En effet, si n = {n} alors somme={somme} \
tandis que, si n = {n-1} alors somme={somme-n}""")
```

```
n = 157
En effet, si n = 157 alors somme=12397 tandis que, si n = 156 alors somme=12240
```

Dans ce cas aussi on peut calculer $n \in \mathbb{N}$ analytiquement car $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$:

$$\sum_{i=4}^{n-1} i \le 12345 < \sum_{i=4}^{n} i \iff 0 \le 12345 - \sum_{i=4}^{n-1} i < n \iff 0 \le 12345 - \left(\frac{(n-1)n}{2} - 1 - 2 - 3\right) < n \iff 0 \le 12351 - \frac{(n-1)n}{2} < n$$
$$\iff \begin{cases} n^2+n-2\times 12351 > 0, \\ n^2-n-2\times 12351 \le 0 \end{cases} \iff \begin{cases} n > \frac{-1+\sqrt{1+8\times 12351}}{2} = \frac{-1+\sqrt{98809}}{2} \approx 156.6694, \\ n \le \frac{1+\sqrt{1+8\times 12351}}{2} = \frac{1+\sqrt{98809}}{2} \approx 157.6694. \end{cases}$$

3.
```
n = 1
somme = n**2
while somme < 12345 :
    n += 1
    somme += n**2
print(f"n = {n}")
print(f"""En effet, si n = {n} alors somme={somme} \
tandis que, si n = {n-1} alors somme={somme-n**2}""")
```

```
n = 33
En effet, si n = 33 alors somme=12529 tandis que, si n = 32 alors somme=11440
```

Sachant que $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$, on peut en théorie calculer $n \in \mathbb{N}$ analytiquement. Cependant, il s'agit cette fois-ci de calculer les racines d'un polynôme de degré 3 et les formules sont beaucoup plus compliquées que celle pour un polynôme de degré 2 comme précédemment.

Exercice 4.15 (Suites type ① : $u_n = f(n)$ (suite géométrique))

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_n = (0.7)^{3n}$. Quel est le plus petit n tel que $u_n < 10^{-4}$?

Hint : on vérifiera par exemple que $u_7 \approx 0.00056$.

Correction

On peut calculer n analytiquement : $u_n = \left(\left(\frac{7}{10}\right)^3\right)^n = \left(\frac{343}{1000}\right)^n$. Il s'agit d'une suite géométrique de raison $0 < q < 1$: elle est donc décroissante et $\lim_n u_n = 0$. On a

$$u_n < 10^{-4} \iff \left(\frac{7}{10}\right)^{3n} < 10^{-4} \iff \log_{10}\left(\frac{7}{10}\right)^{3n} < -4 \iff 3n\log_{10}\left(\frac{7}{10}\right) < -4 \overset{\log_{10}\left(\frac{7}{10}\right)<0}{\iff} n > -\frac{4}{3\log_{10}\left(\frac{7}{10}\right)} \simeq 8.6.$$

La valeur cherchée est donc $n = 9$.

Vérifions nos calculs :

Idée 1 :

```
n = 0
while (0.7)**(3*n)>=1.e-4:
    n += 1
print(n)
```

```
9
```

Idée 2 : on ajoute l'affichage de u_n et u_{n-1}

```
n = 0
u = (0.7)**(3*n)
while u>=1.e-4:
    n += 1
    u = (0.7)**(3*n)
print(f"u_{n}={u}")
print(f"u_{n-1}={(0.7)**(3*(n-1))}")
```

```
u_9=6.571236236353417e-05
u_8=0.00019158123138056612
```

Idée 3 : pour bien voir ce qu'on fait on affiche tous les u_i pour $i = 0,\dots,n$:

```
n = 0
u=1
print(f"u_{n}={u:1.5f}")
while u>=1.e-4:
    n += 1
    u=(0.7)**(3*n)
    print(f"u_{n}={u:1.5f}")
```

```
u_0=1.00000    u_2=0.11765    u_4=0.01384    u_6=0.00163    u_8=0.00019
u_1=0.34300    u_3=0.04035    u_5=0.00475    u_7=0.00056    u_9=0.00007
```

Idée 4 : on a une suite définie par une fonction explicite de n : $u_n = f(n)$. Lorsqu'on aura vu le chapitre 6 sur les fonctions, on pourra alors écrire :

```
f = lambda n : (0.7)**(3*n)    # f: n → (0.7)^{3n}
n = 0
u = f(n)
print(f"u_{n}={u:1.5f}")
while u>=1.e-4:
    n += 1
    u = f(n)
    print(f"u_{n}={u:1.5f}")
```

```
u_0=1.00000    u_2=0.11765    u_4=0.01384    u_6=0.00163    u_8=0.00019
u_1=0.34300    u_3=0.04035    u_5=0.00475    u_7=0.00056    u_9=0.00007
```

Exercice 4.16 (Suites type ② : récurrence $u_{n+1} = f(u_n)$)

Soit $(u_n)_{n\in\mathbb{N}}$ une suite définie par récurrence. Écrire un script qui affiche $u_0,\dots u_6$ dans les cas suivants en utilisant d'abord une boucle `while` puis une boucle `for` :

$$\begin{cases} u_0 = 1, \\ u_{n+1} = 2u_n + 1; \end{cases} \qquad \begin{cases} u_0 = -1, \\ u_{n+1} = -u_n + 5. \end{cases}$$

Hint : on vérifiera par exemple que dans le premier cas on a $u_3 = 15$ et dans le deuxième $u_3 = 6$.

Correction

Méthode sans stockage de tous les éléments :

```
n = 0
u = 1
print(f"u_{n}={u}")
while n<6:
    n += 1
    u = 2*u+1
    print(f"u_{n}={u}")
```

```
u = 1
print(f"u_0={u}")
for n in range(1,7):
    u=2*u+1
    print(f"u_{n}={u}")
```

```
u_0=1
u_1=3
u_2=7
u_3=15
u_4=31
u_5=63
u_6=127
```

```
n = 0
u = -1
print(f"u_{n}={u}")
while n<6:
    n += 1
    u = -u+5
    print(f"u_{n}={u}")
```

```
u = -1
print(f"u_0={u}")
for n in range(1,7):
    u = -u+5
    print(f"u_{n}={u}")
```

```
u_0=-1
u_1=6
u_2=-1
u_3=6
u_4=-1
u_5=6
u_6=-1
```

Méthode avec stockage de tous les éléments :

```
n = 0
u = [1]
print(f"u_{n}={u[-1]}")
while n<6:
    n += 1
    u.append(2*u[-1]+1)
    print(f"u_{n}={u[-1]}")
```

```
u = [1]
print(f"u_0={u[-1]}")
for n in range(1,7):
    u.append(2*u[-1]+1)
    print(f"u_{n}={u[-1]}")
```

```
u_0=1
u_1=3
u_2=7
u_3=15
u_4=31
u_5=63
u_6=127
```

```
n = 0
u = [-1]
print(f"u_{n}={u[-1]}")
while n<6:
    n += 1
    u.append(-u[-1]+5)
    print(f"u_{n}={u[-1]}")
```

```
u = [-1]
print(f"u_0={u[-1]}")
for n in range(1,7):
    u.append(-u[-1]+5)
    print(f"u_{n}={u[-1]}")
```

```
u_0=-1
u_1=6
u_2=-1
u_3=6
u_4=-1
u_5=6
u_6=-1
```

Notons qu'ici nous n'avons pas $u_n = f(n)$ mais $u_n = f(u_{n-1})$. Lorsqu'on aura vu le chapitre 6 sur les fonctions, on pourra alors écrire :

```
#f = lambda a : 2*a+1 # f: a→2a+1
f = lambda a : -a+5    # f: a→5−a
u = [-1]
for n in range(1,7):
    u.append(f(u[-1]))
for i,ui in enumerate(u):
    print(f"u_{i}={ui}")
```

```
u_0=-1    u_1=6    u_2=-1    u_3=6    u_4=-1    u_5=6    u_6=-1
```

Exercice 4.17 (Suites type ③ : récurrence à deux pas $u_{n+1} = f(u_n, u_{n-1})$)
Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par récurrence

$$\begin{cases} u_0 = 0, \\ u_1 = 1, \\ u_{n+1} = 3u_n + 2u_{n-1}. \end{cases}$$

Calculer la valeur u_N et le rang N du premier terme de cette suite supérieur ou égal à 10 000.

Correction
On ne connaît pas combien d'itérations seront nécessaire, on utilisera donc une boucle `while`.

Méthode avec stockage de toutes les valeurs :

```
u = [0,1]
n = 1
while u[-1]<=10000:
    n += 1
    u.append(3*u[-1]+2*u[-2])
for i,ui in enumerate(u):
    print(f"u_{i}={ui}")
```

```
u_0=0
u_1=1
u_2=3
u_3=11
u_4=39
u_5=139
u_6=495
u_7=1763
u_8=6279
u_9=22363
```

Méthode avec stockage des seules valeurs nécessaires :

```
u = [0,1]
n = 1
print(f"u_{0}={u[0]}")
print(f"u_{1}={u[1]}")
while u[-1]<=10000:
    n += 1
    u=[u[-1],3*u[-1]+2*u[-2]]
    print(f"u_{n}={u[-1]}")
```

```
u_0=0
u_1=1
u_2=3
u_3=11
u_4=39
u_5=139
u_6=495
u_7=1763
u_8=6279
u_9=22363
```

Lorsqu'on aura vu le chapitre 6 sur les fonctions, on pourra alors écrire :

```
f = lambda a,b : 3*a+2*b # f: a,b → a+b
u = [0,1]
while u[-1]<=10000:
    u.append(f(u[-1],u[-2]))
for i,ui in enumerate(u):
    print(f"u_{i}={ui}")
```

```
u_0=0
u_1=1
u_2=3
u_3=11
u_4=39
u_5=139
u_6=495
u_7=1763
u_8=6279
u_9=22363
```

Exercice 4.18 (Suites type ③ : récurrence à deux pas $u_{n+1} = f(u_n, u_{n-1})$ (Fibonacci))
Soit $(u_n)_{n\in\mathbb{N}}$ une suite définie par récurrence. Écrire un script qui affiche $u_0, \dots u_6$ en utilisant d'abord une boucle `while` puis une boucle `for` :

$$\begin{cases} u_0 = 0, \\ u_1 = 1, \\ u_{n+2} = u_{n+1} + u_n. \end{cases}$$

Hint : on vérifiera par exemple qu'on a $u_4 = 3$.

Correction
Méthode avec stockage de toutes les valeurs :

```
n = 1
u = [0,1]
while n<6:
    n += 1
    u.append(u[-1]+u[-2])
for i,ui in enumerate(u):
    print(f"u_{i}={ui}")
```

```
u = [0,1]
for n in range(2,7):
    u.append(u[-1]+u[-2])
    #u.append(u[n-1]+u[n-2])
for i,ui in enumerate(u):
    print(f"u_{i}={ui}")
```

```
u_0=0
u_1=1
u_2=1
u_3=2
u_4=3
u_5=5
u_6=8
```

Méthode avec stockage des seules valeurs nécessaires :

```
n = 1
u = [0,1]
print(f"u_{0}={u[0]}")
print(f"u_{1}={u[1]}")
while n<6:
    n += 1
    u=[u[-1],u[-1]+u[-2]]
    print(f"u_{n}={u[-1]}")
```

```
u = [0,1]
print(f"u_{0}={u[0]}")
print(f"u_{1}={u[1]}")
for n in range(2,7):
    u = [u[-1],u[-1]+u[-2]]
    print(f"u_{n}={u[-1]}")
```

```
u_0=0
u_1=1
u_2=1
u_3=2
u_4=3
u_5=5
u_6=8
```

Notons qu'ici nous n'avons pas $u_n = f(n)$ mais $u_n = f(u_{n-1}, u_{n-2})$. Lorsqu'on aura vu le chapitre 6 sur les fonctions, on pourra alors écrire :

```
f = lambda a,b : a+b
u = [0,1]
for n in range(2,7):
    u.append(f(u[-1],u[-2]))
for i,ui in enumerate(u):
    print(f"u_{i}={ui}")
```

```
u_0=0    u_1=1    u_2=1    u_3=2    u_4=3    u_5=5    u_6=8
```

Exercice 4.19 (Suites type ③ : récurrence à deux pas $u_{n+1} = f(u_n, u_{n-1})$ (Fibonacci – bis))
Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par récurrence :

$$\begin{cases} u_0 = 0, \\ u_1 = 1, \\ u_{n+2} = u_{n+1} + u_n. \end{cases}$$

Soit $(v_n)_{n>1}$ la suite définie par $v_n = \frac{u_n}{u_{n-1}}$. On peut montrer que $v_n \xrightarrow[n\to\infty]{} \varphi$ (le nombre d'or).

Calculer le plus petit N pour lequel $|v_N - v_{N-1}| < 10^{-10}$.

Correction

```
u = [0,1,1]
v = [0,0,u[-1]/u[-2]]
while abs(v[-1]-v[-2])>=1.e-10:
    u.append(u[-1]+u[-2])
    v.append(u[-1]/u[-2])
N = len(u)-1
print(f"N={N}")
print(f"u_{N-2}={u[N-2]:4d}, v_{N-2}={v[N-2]:.6f}")
print(f"u_{N-1}={u[N-1]:4d}, v_{N-1}={v[N-1]:.6f}")
print(f"u_{N}={u[N]:4d}, v_{N}={v[N]:.6f}")
print(f"|v_{N-1}-v_{N-2}|={abs(v[N-1]-v[N-2])}")
print(f"|v_{N}-v_{N-1}|={abs(v[N]-v[N-1])}")
```

```
N=28
u_26=121393, v_26=1.618034
u_27=196418, v_27=1.618034
u_28=317811, v_28=1.618034
|v_27-v_26|=1.0979950282319351e-10
|v_28-v_27|=4.193956293363499e-11
```

Exercice Bonus 4.20 (Pydéfis – Fibonacci)

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent

$$\begin{cases} F_0 = 0, \\ F_1 = 1, \\ F_{n+2} = F_{n+1} + F_n \quad \text{pour } n = 1, 2, \dots \end{cases}$$

Défi : trouvez le premier terme de la suite de Fibonacci dont la somme des chiffres est égale à $v = 120$.

Testez votre code : si $v = 24$ alors la solution est 987 (le dix-septième terme) puisque $9 + 8 + 7 = 24$.

Source : https://pydefis.callicode.fr/defis/FiboChiffres/txt

Exercice 4.21 (Défi Turing n°2 – Fibonacci)

Chaque nouveau terme de la suite de Fibonacci est généré en ajoutant les deux termes précédents. En commençant avec 1 et 1, les 10 premiers termes sont 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

En prenant en compte les termes de la suite de Fibonacci dont les valeurs ne dépassent pas 4 millions, trouver la somme des termes pairs.

Correction
Chaque terme de cette suite est la somme des deux termes précédents :

$$\begin{cases} F_1 = 1, \\ F_2 = 1, \\ F_{n+2} = F_{n+1} + F_n \quad \text{pour } n = 2, 3, \dots \end{cases}$$

On peut réécrire cette suite comme suit :

$$\begin{cases} F_0 = 0, \\ F_1 = 1, \\ F_{n+2} = F_{n+1} + F_n \quad \text{pour } n = 1, 2, \dots \end{cases}$$

ainsi on a les mêmes indices qu'en Python.

Dans cet exercice on doit trouver la somme des termes de la suite de Fibonacci dont la valeur est paire et ne dépasse pas 4 millions. Un programme brute force est suffisant. Pour savoir si un nombre est pair, il faut que ce nombre soit divisible par 2, c'est à dire que le reste de la division de ce nombre avec 2 soit égal à 0. En Python, il faut utiliser le signe % pour obtenir le reste d'une division.

Dans ce premier code on garde tous les termes de la suite :

```
fib = [0,1]
s = 0
while (fib[-1]<=4.e6):
	fib.append(fib[-1]+fib[-2])
	if (fib[-1]%2)==0:
		s += fib[-1]
print(s)
```

Dans ce nouveau code, on ne garde que les termes qui nous servent à construire le terme suivant :

```
a,b = 0,1
s = 0
while (b<=4.e6):
	if (b%2)==0:
		s += b
	a,b = b,a+b
print(s)
```

Dans les deux cas le résultat est

```
4613732
```

Exercice 4.22 (Suites type ③ : récurrence à deux pas $u_{n+1} = f(u_n, u_{n-1})$ (Euclide))
L'algorithme d'Euclide pour le calcul du **PGCD** de deux entiers u, v avec $u \ge v > 0$ peut être ainsi décrit. On définit, par récurrence à deux pas, une suite $(x_n)_{n\in\mathbb{N}}$ en posant $x_0 = u$, $x_1 = v$ et, tant que $x_i > 0$, on pose x_{i+1} = reste de la division euclidienne de x_{i-1} par x_i. Le dernier terme non nul de la suite est le PGCD de u et v.

Coder cet algorithme et le valider (voir aussi l'exercice 6.25).

Correction
On construit la suite

$$\begin{cases} x_0 = u, \\ x_1 = v, \\ x_{i+1} = x_{i-1} \pmod{x_i} \end{cases}$$

tant que $x_{i+1} > 0$. Dès que $x_{i+1} = 0$ on s'arrête et le PGCD de u et v sera x_i.

```
for u,v in [(3*5,2*3),(5*7,2*3),(2*2*3,2*2)]:
	x = [u,v]
	while x[-1]!=0:
		x.append(x[-2]%x[-1])
	print(f"PGCD({u},{v})={x[-2]}")
```

```
PGCD(15,6)=3
PGCD(35,6)=1
PGCD(12,4)=4
```

Si on ne veut pas stocker toute la suite, nous pouvons alors juste écrire :

```
for u,v in [(3*5,2*3),(5*7,2*3),(2*2*3,2*2)]:
	a,b = u,v
	while b>0:
		a,b = b, a%b
	print(f"PGCD({u},{v})={a}")
```

```
PGCD(15,6)=3
PGCD(35,6)=1
PGCD(12,4)=4
```

Exercice 4.23 (Suites type ④ : récurrence $(u,v)_{n+1} = (f_1(u_n, v_n), f_2(u_n, v_n))$ et affectations //)
Calculer u_4 et v_4 avec $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ deux suites définies par

$$\begin{cases} u_0 = 1, \\ v_0 = -1, \\ u_{n+1} = 3v_n + 1, \\ v_{n+1} = u_n^2. \end{cases}$$

Hint : on vérifiera par exemple qu'on a $u_3 = 13$ et $v_3 = 16$.

Correction

L'instruction "`u,v = 3*v+1, u*u`" permet de définir les nouveaux u et v en même temps (il s'agit de deux affectations en parallèle) :

```
u,v = 1,-1
print(f"u_0={u},\t v_0={v}")
for n in range(1,5):
	u,v = 3*v+1, u*u
	print(f"u_{n}={u},\t v_{n}={v}")
```

```
u_0=1,	 v_0=-1
u_1=-2,	 v_1=1
u_2=4,	 v_2=4
u_3=13,	 v_3=16
u_4=49,	 v_4=169
```

Sinon, il faut utiliser des variables supplémentaires `uold` et `vold` pour sauvegarder les anciennes valeurs :

```
u,v = 1,-1
print(f"u_0={u},\t v_0={v}")
for n in range(1,5):
    uold = u
    u = 3*v+1
    v = uold*uold
    print(f"u_{n}={u},\t v_{n}={v}")
```

```
u_0=1,	 v_0=-1
u_1=-2,	 v_1=1
u_2=4,	 v_2=4
u_3=13,	 v_3=16
u_4=49,	 v_4=169
```

Attention à ne pas écrire

```
u,v=1,-1
print(f"u_0={u}, v_0={v}")
for n in range(1,5):
    u=3*v+1
    v=u*u
    print(f"u_{n}={u}, v_{n}={v}")
```

```
u_0=1, v_0=-1
u_1=-2, v_1=4
u_2=13, v_2=169
u_3=508, v_3=258064
u_4=774193, v_4=599374801249
```

qui correspond à la suite

$$\begin{cases} u_0 = 1, \\ v_0 = -1, \\ u_{n+1} = 3v_n + 1, \\ v_{n+1} = u_{\mathbf{n+1}}^2. \end{cases}$$

Méthode avec stockage de toutes les valeurs :

```
u = [1]
v = [-1]
print(f"u_0={u[-1]},\t v_0={v[-1]}")
for n in range(1,5):
    u.append( 3*v[n-1]+1 )
    v.append( u[n-1]**2 )
    print(f"u_{n}={u[-1]},\t v_{n}={v[-1]}")
```

```
u_0=1,	 v_0=-1
u_1=-2,	 v_1=1
u_2=4,	 v_2=4
u_3=13,	 v_3=16
u_4=49,	 v_4=169
```

Attention à ne pas écrire

```
u = [1]
v = [-1]
print(f"u_0={u[-1]},\t v_0={v[-1]}")
for n in range(1,5):
    u.append( 3*v[-1]+1 )
    v.append( u[-1]**2 )
    print(f"u_{n}={u[-1]},\t v_{n}={v[-1]}")
```

```
u_0=1,	 v_0=-1
u_1=-2,	 v_1=4
u_2=13,	 v_2=169
u_3=508,	 v_3=258064
u_4=774193,	 v_4=599374801249
```

qui correspond à la suite

$$\begin{cases} u_0 = 1, \\ v_0 = -1, \\ u_{n+1} = 3v_n + 1, \\ v_{n+1} = u_{\mathbf{n+1}}^2. \end{cases}$$

Exercice 4.24 (Suites type ④ : récurrence $(u,v)_{n+1} = (f_1(u_n,v_n), f_2(u_n,v_n))$ et affectations //)

Écrire une boucle `while` pour calculer la moyenne arithmétique-géométrique de deux nombres réels positifs x et y. Elle est définie comme la limite des deux suites $(a_n)_{n\in\mathbb{N}}$, $(b_n)_{n\in\mathbb{N}}$:

$$\begin{cases} a_0 = x, \\ b_0 = y, \\ a_{n+1} = \dfrac{a_n + b_n}{2}, \\ b_{n+1} = \sqrt{a_n b_n}. \end{cases}$$

Les deux suites convergent vers le même nombre $\ell(x,y)$. Utiliser la boucle pour déterminer la constante de Gauss $G = 1/(1,\sqrt{2})$.

Source : https://scipython.com/book/chapter-2-the-core-python-language-i/questions/the-arithmetic-geometric-mean/

Correction

```
tol = 1.e-14
an, bn = 1, 2**0.5
while abs(an-bn) > tol :
    an, bn = (an + bn) / 2, (an*bn)**0.5

print(f'G = {1/an:.14f} = {1/bn:.14f}')
```

```
G = 0.83462684167407 = 0.83462684167407
```

Exercice 4.25 (Suites type ⑤ : récurrence $(u,v)_{n+1} = (f_1(n,u_n,v_n), f_2(n,u_n,v_n))$ et affectations //)

Calculer u_{100} et v_{100} où $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ sont deux suites définies par

$$\begin{cases} u_0 = v_0 = 1, \\ u_{n+1} = u_n - \frac{v_n}{n+1}, \\ v_{n+1} = (n+1)u_n + v_n, \qquad n = 0,\dots \end{cases}$$

Hint : on vérifiera par exemple qu'on a $u_5 \approx 0.1166$ et $v_5 \approx -14.75$.

Bien noter la différence avec l'exercice 4.23 car, cette fois-ci, n intervient explicitement dans la définition de u et v.

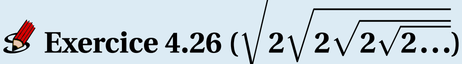
Correction

Calculons les premiers terms "à la main" pour vérifier que notre boucle a été bien écrite : $(u,v)_{n=0} = (1,1)$ puis $(u,v)_1 = \left(u_0 - \frac{v_0}{0+1}, (0+1)u_0 + v_0\right) = (0,2)$ et $(u,v)_2 = \left(u_1 - \frac{v_1}{1+1}, (1+1)u_1 + v_1\right) = (-1,2)$

Calcule explicite du 100-ème terme :

```
u,v = 1,1 # n=0, u_0, v_0
for n in range(0,100):
    # NB ici !=range(1,101)
    # car on a "n" dans l'affectation
    (u,v) = (u-v/(n+1),(n+1)*u+v)
print("u_100 =",u,"\nv_100 =",v)
```

```
u_100 = -82506365759177.17
v_100 = -9476942524736076.0
```



Exercice 4.26 ($\sqrt{2\sqrt{2\sqrt{2\sqrt{2\dots}}}}$)

On peut donner un sens au nombre b qui s'écrit

$$b = \sqrt{2\sqrt{2\sqrt{2\sqrt{2\dots}}}}.$$

Lequel et que vaut b ?

Correction

Introduisons la suite récurrente de premier terme $u_0 > 1$ telle que $u_{n+1} = \sqrt{2u_n}$.

Suite majorée. On montre par récurrence que $u_n < 2$ pour tout $n \in \mathbb{N}$: on a bien $u_0 < 2$; montrons que si $u_k < 2$ alors $u_{k+1} < 2$. On a

$$u_k < 2 \implies 2u_k < 4 \implies u_{k+1} = \sqrt{2u_k} < \sqrt{4} = 2.$$

Monotonie. Pour tout $n \in \mathbb{N}$ on a

$$\frac{u_{n+1}}{u_n} = \sqrt{\frac{2}{u_n}} \geq 1$$

pour tout $n \in \mathbb{N}$.

Convergence. (u_n) est une suite monotone croissante et majorée. Elle est donc convergente et b est sa limite.

Limite. La suite (u_{n+1}) tend vers b et la suite $f(u_n)$ tend vers $f(b)$. La limite vérifie donc

$$b = \sqrt{2b}.$$

On a donc $b(b-2) = 0$ ainsi $b = 2$.

```
uu = [0,1]
while uu[-1] != uu[-2] :
    uu.append( (2*uu[-1])**0.5 )
print(len(uu),uu[-1])
```

```
56 1.9999999999999998
```

Exercice 4.27 (4n)

Soit $m \in \mathbb{N}$ un nombre terminant par 4 (*i.e.* le chiffre des unités est 4). On pourra écrire ce nombre sous la forme $\underline{s4}$ ayant noté avec une barre la suite des chiffres qui le composent dans l'écriture décimale. Par exemple, si $m = 125674$, on a $s = 12567$.

Parmi ces nombres se terminant par 4, il en existe un tel que, lorsqu'on déplace ce 4 tout à gauche, on obtient le nombre $4 \times m$. Autrement dit, $\underline{4s}$ vaut $4 \times m$. Quel est ce nombre ?

Correction

Le nombre m se termine par le chiffre 4, on peut donc le construire comme $10n+4$ avec $n \in \mathbb{N}$, soit encore `int(str(n)+'4')`.

Le nombre obtenu en déplaçant le dernier 4 tout à gauche peut être construit comme `int('4'+str(n))`.

```
n = 0 # m = 10*n+4
while int(str(4)+str(n)) != 4*int(str(n)+str(4)):
    n += 1

m = 10*n+4
print(f'm = {m}')
print(f'En effet, si n = {n}, alors m = {m} et 4m = {4*m}')
```

```
m = 102564
En effet, si n = 10256, alors m = 102564 et 4m = 410256
```

☆ Exercice Bonus 4.28 (Défi Turing n°70 – Permutation circulaire)

Prenons le nombre $n = 102564$. En faisant passer le dernier chiffre complètement à gauche (*i.e.* on enlève le chiffre des unités et on le place toute à gauche), on obtient un nouveau nombre : $m = 410256$. Ce nouveau nombre a une particularité : il est un multiple du nombre de départ (mais différent du nombre de départ). En effet, $m = 410256 = 102564 \times 4 = 4n$.

Additionner tous les nombres de 6 chiffres ayant cette propriété.

Remarque : dans l'exercice 4.27 on cherchait que les multiples de 4, ici n'importe quel multiple.

Correction

```
L = []
for n in range(10**5,10**6):
    m = int(str(n)[-1]+str(n)[:-1])
    if m!=n and m%n==0 :
        L.append(n)
        print(f"n={n} car {n}x{m//n}={m}")
print("Les nombres ayant cette propriété sont", *L)
print("Leur somme vaut", sum(L))
```

```
n=102564 car 102564x4=410256
n=128205 car 128205x4=512820
n=142857 car 142857x5=714285
n=153846 car 153846x4=615384
n=179487 car 179487x4=717948
n=205128 car 205128x4=820512
n=230769 car 230769x4=923076
Les nombres ayant cette propriété sont 102564 128205 142857 153846 179487 205128 230769
Leur somme vaut 1142856
```

Exercice 4.29 (Plus petit diviseur)

Pour $n \in \mathbb{N}$ donné ≥ 2, afficher le plus petit diviseur $d \geq 2$ de n.

Correction

On rappelle que d divise n si et seulement si `n%d==0`. Utiliser une boucle `for d in range(2,n+1)` est une mauvaise idée car dès qu'on trouve un diviseur, on sait qu'il sera le plus petit et il est inutile de continuer dans la boucle. Avec une boucle `while`, on continue à chercher tant qu'on n'a pas trouvé de diviseur (dès qu'on l'a trouvé, la boucle s'arrête).

Voici un exemple pour $n = 49$:

```
n = 49
d = 2
while n%d!=0 and d<=n:
    d += 1
print(f"n = {n} et d = {d}")
```

```
n = 49 et d = 7
```

Exercice 4.30 (Puissance)

Soit $n \in \mathbb{N}$ donné et cherchons la première puissance de 2 plus grande que n en utilisant une boucle `while`.

Correction

On cherche p tel que $2^{p-1} \leq n < 2^p$. On affichera 2^p.

Méthode 1 :

```
n = 100
p = 0
while 2**p<=n:
    p += 1 # p = p+1
print(2**p)
```

```
128
```

Méthode 2 :

```
n = 100
p2 = 1
while p2<=n:
    p2 *= 2 # p2 = p2*2
print(p2)
```

```
128
```

Méthode 3 : $2^p = q \iff p = \log_2 q$

```
from math import log
n = 100
p = int(log(n,2))+1
print(2**p)
```

```
128
```

☆ Exercice Bonus 4.31 (Défis Turing n°72)

Parmi tous les entiers inférieurs à 1 milliard, combien sont des carrés se terminant par exactement 3 chiffres identiques (mais pas 4 ou plus) ? Par exemple, $213444 = 462^2$.

Correction

On peut calculer le carrée des nombres de 10 à 10^5 et vérifier qu'il a au moins 3 chiffres et au plus 9 chiffres (*i.e.* $10^3 \le n < 10^9$), que les derniers trois chiffres sont identiques mais pas celui qui les précède.

```
L = {}
for i in range(4,10**5):
    n = i*i
    sn = str(n)
    if n<=10**9 and sn[-1]==sn[-2]==sn[-3] and sn[-4]!=sn[-3]:
        L[i] = n
#print(L)
print(len(L))
```

```
127
```

La condition `sn[-1]==sn[-2]==sn[-3]` équivaut à `len(set(sn[-3:]))==1`.

☆ Exercice Bonus 4.32 (Dictionnaire)

Soit `L` une liste de listes. Créer un dictionnaire qui contient comme clés la longueur des listes et comme valeur la liste des listes de telle longueur. Autrement dit, on regroupe chaque liste de la liste `L` selon leur longueur.

Correction

```
L = [ [1], [4,5], [-1,8,9], [11,12,13] , [0] , [1,2] , [3,17] ]

dp = {}
for l in L:
    k = len(l)
    if k not in dp: dp[k] = []
    dp[k].append(l)

print(dp)
```

```
{1: [[1], [0]], 2: [[4, 5], [1, 2], [3, 17]], 3: [[-1, 8, 9], [11, 12, 13]]}
```

☆ Exercice Bonus 4.33 (Dictionnaires : construction d'un histogramme)

Supposons que nous voulions créer un histogramme de la fréquence d'utilisation de chaque lettre de l'alphabet dans un texte donné. Il est possible de réaliser cette tâche avec un algorithme extrêmement simple basé sur un dictionnaire.

Tout d'abord, nous créons un dictionnaire vide nommé `lettres`. Ensuite, nous remplissons le dictionnaire en utilisant les caractères de l'alphabet comme clés. Les valeurs stockées pour chacune de ces clés sont les fréquences des caractères correspondants dans le texte. Pour calculer ces fréquences, nous parcourons la chaîne de caractères `texte`. Pour chaque caractère, nous utilisons la méthode `get()` pour interroger le dictionnaire en utilisant le caractère comme clé. Ainsi, nous pouvons lire la fréquence déjà stockée pour ce caractère. Si cette valeur n'existe

pas encore, la méthode `get()` renvoie une valeur nulle. Dans tous les cas, nous incrémentons la valeur trouvée et la stockons dans le dictionnaire à l'emplacement correspondant à la clé (c'est-à-dire au caractère en cours de traitement).

Enfin, si nous souhaitons afficher l'histogramme dans l'ordre alphabétique, nous pouvons convertir le dictionnaire en une liste de tuples. Nous pouvons alors utiliser la méthode `sort()`, qui ne s'applique qu'aux listes, pour trier cette liste.

Source : https://python.developpez.com/cours/apprendre-python3/?page=page_12

Correction

```
texte = "les saucisses et saucissons secs sont dans le saloir"
lettres = {}
for c in texte:
    lettres[c] = lettres.get(c, 0) + 1
print(lettres)
```

```
{'l': 3, 'e': 5, 's': 14, ' ': 8, 'a': 4, 'u': 2, 'c': 3, 'i': 3, 't': 2, 'o': 3, 'n': 3, 'd':
↪ 1, 'r': 1}
```

```
lettres_triees = list(lettres.items())
lettres_triees.sort()
print(lettres_triees)
```

```
[(' ', 8), ('a', 4), ('c', 3), ('d', 1), ('e', 5), ('i', 3), ('l', 3), ('n', 3), ('o', 3),
↪ ('r', 1), ('s', 14), ('t', 2), ('u', 2)]
```

☆ Exercice Bonus 4.34 (Dictionnaires : The Most Frequent)

Vous avez une séquence de chaînes et vous souhaitez déterminer la chaîne la plus fréquente dans la séquence (il n'y en a qu'une).

Entrée : une liste non vide de chaînes de caractères.

Sortie : une chaîne de caractères.

Source : https://py.checkio.org/en/mission/the-most-frequent/

Correction

```
def most_frequent(data: list[str]) -> str:
    dico = {}
    for d in data:
        dico[d] = dico.get(d,0)+1
    return max(dico,key=dico.get)

# TESTS
print(most_frequent(["a", "b", "c", "a", "b", "a"]))
print(most_frequent(["a", "a", "bi", "bi", "bi", "c"]))
print(most_frequent(["a"]))
print(most_frequent(["a","a","z"]))
```

```
a
bi
a
a
```

⚠ Exercice Bonus 4.35 (Pydéfis – Vous parlez Fourchelangue ?)

En ce moment Harry fait des rêves inquiétants et il arrive même qu'il parle Fourchelangue pendant son sommeil.

Le Fourchelangue n'est finalement pas très difficile à comprendre : chaque « syllabe » Fourchelangue correspond à un caractère. La correspondance est donnée dans la table suivante (on remarque que I et J d'une part, et U et V d'autre part se prononcent de la même manière en Fourchelangue) :

HFH	FFH	SHS	SHH	SSH	FHF	FSS	HFF	HHH	SFS	FFS	FHS
A	B	C	D	E	F	G	H	IJ	K	L	M
SSF	FHH	HHF	SFF	FSF	FSH	HHS	FFF	SSS	HFS	SHF	SFH
N	O	P	Q	R	S	T	UV	W	X	Y	Z

La syllabe HS permet de séparer les mots.

À titre d'exemple, BONJOUR HARRY se dit, en Fourchelangue, FFHFHHSSFHHHFHHFFFFSFHSHFFHFHFSFFSFSHF

Ron a réussi à noter ce que Harry a dit la nuit dernière. Il demande votre aide pour comprendre le contenu du message car il craint que certaines personnes ne soient en danger.

Le texte à traduire est disponible ici : https://pydefis.callicode.fr/defis/Fourchelangue/txt

Exercice Bonus 4.36 (Pydéfis - Code Konami)

Dans le monde dédié à Gradius, vous affronterez de multiples ennemis aux commandes du Vic Viper. Un message vous a été remis au début de votre mission, qui devrait vous révéler les épreuves qui vous attendent. Ce message semble directement inspiré du Code Konami.

Des membres de votre équipe ont pu mettre la main sur le moyen de traduire ces messages. Le code est composé de 6 symboles : ←, →, ↓, ↑, A et B. À chaque couple de symboles (il y a 36 possibilités) est associé un caractère alphabétique ou de ponctuation. Par exemple : BA correspond à e, ←→ correspond à m, ↓↓ correspond à o, ↑↑ correspond à p. Ainsi, la séquence ↑↑↓↓←→←→ BA en Code Konami correspond à "pomme".

Le code complet ainsi que l'intégralité du message à déchiffrer sont disponibles ici : https://pydefis.callicode.fr/defis/C22_KonamiCode/input

Ce message vous révèle, entre autres, quelle sera la sixième épreuve à affronter. Indiquez-la pour valider le défi (entrez moins de 50 caractères).

Source : https://pydefis.callicode.fr/defis/C22_KonamiCode/txt

Correction

```python
dico = {'←←' : 'h','←↑' : '!','←→' : 'm','←↓' : 'l','←A' : 's','←B' : ',','↑←' :
↪  'r','↑↑' : 'p','↑→' : 'x','↑↓' : 'b','↑A' : 'j','↑B' : 'v','→←' : 'a','→↑' : 'i','→→' : '
↪  ','→↓' : 'w','→A' : 'g','→B' : 'é','↓←' : 'è','↓↑' : 't','↓→' : '.','↓↓' : 'o','↓A' :
↪  'n','↓B' : 'u','A←' : 'à','A↑' : '?','A→' : 'y','A↓' : 'c','AA' : 'f','AB' : 'd','B←' :
↪  'q','B↑' : 'k','B→' : "",'B↓' : 'z','BA' : 'e','BB' : 'ê'}

msg =
↪  "←↓↓↓↑←←AB←↓BBA→→↓↑↓B→→→←↓B↑←→←←A→→↑B→←→↑↓AA↓↓B→→↑↓→↑→A→→A↓↓↓↑←BA←B→→→←AAAA↑←↓↓↓A↓↑→B→→←↓BA→

decoded = ""
for i in range(len(msg)//2):
    decoded += dico[msg[2*i:2*i+2]]
print(decoded)
```

```
lorsque tu auras vaincu big core, affronté le regard impitoyable des statues  moai et survécu
↪  au ballet des soucoupes, après avoir tenu tête aux amibes  tentaculaires, tu entreras
↪  enfin dans la base des bactérians peuplée de robots bipèdes. puis, tu découvriras le
↪  cerveau, quil te faudra détruire. et ne tavise pas de commencer à déchiffrer ce message à
↪  la main en partant de la fin, car mon objectif est de te décourager dagir de la sorte.
```

Exercice Bonus 4.37 (Pydéfis - Difficile de comprendre un lapin crétin)

Vous ne vouliez pas y aller, mais vous venez de vous retrouver dans le monde des lapins crétins. Ça se termine généralement en explosion et catastrophe, et vous ne comptez pas perdre votre avatar si bêtement.

Tic tac... Ils jouent encore avec une bombe.

Vous devez la désamorcer avant qu'il ne soit trop tard, mais elle est remplie de leviers, manettes... et vous n'avez pas d'autre instruction qu'un lapin qui semble vouloir votre bien et vous informe de ce qu'il faut faire... à sa façon.

BWAXA BWAWA ?

Vous savez qu'ils ne disent pas n'importe quoi. Les lettres utiles sont simplement insérées entre des syllabes sans signification, de type : `BWA.A` où `.` peut être une lettre arbitraire.

Pour dire HELLO, un lapin pourrait «simplement» vous dire :

```
BWAYABWANAHBWAIAEBWAPABWAMABWAZALBWAPABWALALBWAGABWAQABWAEAOBWAEABWAYA
          ^     ^               ^          ^               ^
```

Si on découpe les syllabes, la structure est très visible :

```
BWAYA BWANA H BWAIA E BWAPA BWAMA BWAZA L BWAPA BWALA L BWAGA BWAQA BWAEA O BWAEA BWAYA
            ^       ^                   ^             ^                   ^
```

La transcription de votre ami le lapin est disponible ici : https://pydefis.callicode.fr/defis/C22_BwaCode01/input

Sur quoi devez-vous appuyer pour désamorcer la bombe ?

Source : https://pydefis.callicode.fr/defis/C22_BwaCode01/txt

Correction

```python
# msg = "BWAYABWANAHBWAIAEBWAPABWAMABWAZALBWAPABWALALBWAGABWAQABWAEAOBWAEABWAYA"

msg = """BWAZABWAKABWAJABWAPABWAIABWANAIBWAKABWAFABWAIABWAZABWAMALBWAMABWACABWATABWAWABWA
PABWAKABWADABWARABWALABWACABWAOABWABABWAEABWATABWAJAFBWAXAABWAAABWATABWAHABWACAU
BWAJABWANABWASABWAAABWARABWAXABWACABWAWABWAHABWAIATBWAGABWADABWAGABWASABWACABWAU
ABWAAABWAPABWAIABWAFABWAUABWATABWACABWAPABWATABWARABWAWABWATATBWASABWAKABWATABWA
NABWAXABWAMABWARAIBWAPABWAUABWAGABWAZABWAPABWAKARBWAMABWAXABWAPABWAEABWAYABWAVAB
WAVABWANABWAQABWAYABWAHABWAUABWACABWAEABWAOABWALABWASABWAVAEBWAIABWAJABWAJABWAIA
BWAMABWAJABWAPABWANABWATABWAPARBWAMABWAOABWAZABWARABWAXABWAHABWADABWAOABWAGABWAI
ABWACABWAEABWAQABWAFABWABABWATALBWAQABWACABWATABWAGABWAWABWAAABWASABWALABWAJABWA
TABWAGABWAFABWAKABWAUAEBWAGABWAOABWAZABWAMABWAMABWAIABWASABWANABWAOABWABABWANABW
ANABWAKABWADABWAJABWAFABWAMABWAPABWACALBWASABWAUABWASABWAOABWAFABWABABWAVABWAJAB
WADABWADABWAVABWAHABWARABWAZABWADAEBWAKABWARABWAAABWANABWADABWAVABWAHABWAEABWANA
BWAXAVBWAZABWASABWAWABWATABWAIABWAFAIBWAIABWAAABWATABWAMABWATABWAGABWAWABWAZABWA
EABWATABWAZABWAIABWAAABWAFABWANAEBWATABWAUABWASARBWAUABWAQABWAFABWAZABWAAABWAUAB
WABABWAKABWAIABWAVABWAGABWAKABWAZABWARABWARABWAKABWACABWAPABWALAEBWAMABWAMABWAHA
BWAQABWAKABWAHABWAAABWADABWAOABWAIABWAWABWAYATBWASABWAYABWACABWAYABWAAABWAIAABWA
YAPBWAJABWANABWAVABWAOABWAQABWASABWACAPBWAVABWALABWAQABWACABWAJABWASABWADABWAWAB
WARABWANABWAGABWADABWAYABWAFABWAJABWAXABWAHABWACABWAKAUBWAHABWATABWARAYBWADABWAF
ABWAXABWAQABWAOABWAHABWAZABWAFABWACABWAMABWAFABWATABWACAEBWAYABWAMABWAEABWAMABWA
LABWABABWAXABWAZABWAAABWAMABWAXABWARABWATABWABABWAGARBWAGABWAIABWAQABWAZABWAXABW
AWABWADASBWAPABWAPABWATABWAJABWAVABWAVABWASABWALABWAJABWASABWAZABWAKABWAKABWARAB
WAMABWALABWALAUBWAPABWAZABWAZABWAPABWAXABWAVABWASABWAAABWAAABWAKABWAEABWAWARBWAM
ABWARABWAAABWASABWAVABWAGABWALABWAYABWACABWAJABWASALBWAAABWARABWAYABWAZABWAAABWA
UABWAQABWADABWABABWAOABWAOABWAEABWAFABWAYABWAUABWAJAEBWAWABBWAJABWABABWAAABWAIAB
WAYABWAJABWAEABWASABWATABWAWABWAXAOBWABABWACABWAJAUBWAYABWANABWATABWAEABWAIABWAV
ABWAMABWANABWAAABWACABWAUABWAEABWAEABWAZABWAZABWAMABWAGABWAOATBWAZABWAUABWAGABWA
LABWAOABWAFABWAPABWAQABWAJABWAGABWAYABWAFABWAZABWAPAOBWARANBWANABWAUABWABAJBWAXA
BWAMABWAOABWADABWAZABWABABWAVABWAAABWAEABWACABWAKABWAKAABWAZABWAZABWAJABWAJABWAB
ABWAGABWADABWALABWAMABWAEABWAIABWAGABWAPABWAWABWAIABWAFABWALABWACAUBWADABWAQABWA
YABWADANBWANABWAYABWAFAEBWALABWAZABWAJABWATABWAXABWAMABWACABWAUABWAQABWACABWALAB
WATABWAXABWATA"""

msg = msg.replace("\n", "")

i = 0
while i <len(msg)-5:
    if msg[i+5:i+8] == "BWA" :
        i += 5
    else:
        print(msg[i+5],end='')
```

```
        i += 6
```

```
ILFAUTTIRERLELEVIERETAPPUYERSURLEBOUTONJAUNE
```

Exercice Bonus 4.38 (Pydéfis – Le retourneur de temps)

Le professeur Dumbledore a confié à Hermione son retourneur de temps pour l'aider à suivre tous les cours qu'elle a choisis pour sa troisième année.

Le retourneur de temps est un objet très complexe. Il permet de remonter dans le temps d'un certain nombre de minutes. Pour voyager dans le passé, on fait faire des tours au retourneur de temps (il ressemble à un sablier), et à chaque tour, une quantité de minutes à remonter est calculée. Lorsqu'on arrête de l'actionner, le voyage commence.

- En faisant un seul tour, le retourneur nous enverra 2 minutes dans le passé.
- En faisant deux tours, il nous enverra 4 minutes dans le passé
- Avec trois tours, nous ferons un voyage de 6 minutes dans le passé
- ...

On a l'impression qu'à chaque tour, le voyage nous fera reculer de 2 minutes supplémentaires. Mais le principe est en fait un peu plus complexe.

Si à un moment le nombre de minutes du voyage a la somme de ses chiffres divisible par 7 (c'est-à-dire si la somme de ses chiffres vaut 7, ou 14, ou 21...), alors au prochain tour, plutôt que de nous faire voyager de 2 minutes supplémentaires dans le passé, le retourneur modifiera la durée pour que notre voyage nous fasse remonter de 7 minutes de moins (au lieu de 2 de plus) !

Le tableau suivant récapitule la durée du voyage (en minutes) en fonction du nombre de tours effectués sur le retourneur :

nombre de tours	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
durée voyage	2	4	6	8	10	12	14	**16**	9	11	13	15	17	19	21	23	**25**	18

Comme la somme des chiffres de 16 est divisible par 7, le 9ème tour, au lieu de nous faire remonter le temps de 16 + 2 minutes, nous le fera remonter de 16-7 minutes.

De même, comme la somme des chiffres de 25 est divisible par 7, le 18ème tour, au lieu de nous faire remonter le temps de 25 + 2 minutes, nous le fera remonter de 25-7 minutes.

Le retourneur de temps est ainsi conçu pour qu'on ne puisse pas voyager trop longtemps dans le passé. Cette manière de calculer impose en effet une limite pour le voyage le plus long qu'il est possible de faire.

Combien de minutes on peut reculer dans le temps au maximum ?

Source : https://pydefis.callicode.fr/defis/RetourneurTemps/txt

Exercice Bonus 4.39 (Pydéfis – Entrée au Ministère)

Il n'est pas si facile d'entrer au ministère de la magie. Un des moyens d'accès, outre la poudre de cheminette (et les cuvettes des toilettes), est de passer par une des cabines téléphoniques Londoniennes. Une fois dans la cabine, on entre un code et après un rapide voyage, on se retrouve dans l'atrium du ministère.

Le code de la cabine est changé régulièrement, mais il a toujours la même caractéristique. Si on note les chiffres utilisés pour écrire le carré du code, on a très exactement besoin des chiffres 1, 2, 4, 6 et 7. Ainsi, récemment, le code à entrer dans la cabine était 64224. En effet, $64224^2 = 4124722176$. On a besoin des chiffres 1, 2, 4, 6 et 7 pour l'écrire. Le prochain code est le prochain nombre, plus grand que 64224, qui a la même propriété, et ainsi de suite. Notez que 64631 ne convient pas, car une fois mis au carré, il ne contient pas de 2. Or le carré doit utiliser exactement les chiffres 1, 2, 4, 6 et 7 (tous!). Quels seront les trois prochains codes à utiliser dans la cabine pour entrer au ministère de la magie ?

Source : https://pydefis.callicode.fr/defis/CodeCabine/txt

Exercice Bonus 4.40 (Pydéfis – Créatures nocturnes pas si sympathiques que cela…)

Antonio a eu l'idée saugrenue d'aller se promener dans une forêt inconnue une nuit de pleine lune. Bien entendu, il ne savait pas que celle-ci était peuplée de créatures nocturnes pour le moins antipathiques. En effet, il est amené à rencontrer des meutes de chauves-souris enragées, des skellingtons et des zombies tout droit sortis de Minecraft et Call of Duty ainsi que des fantômes baveux.

Heureusement, Antonio a toujours une faux dans son sac. Oui, c'est étrange, mais qu'est-ce qui ne l'est pas dans un jeu de survie ! Cette faux va lui permettre de faucher plus ou moins rapidement ces créatures maléfiques de la nuit et même, qui sait, de lui sauver la vie…

Sachant que :

- au démarrage du jeu, il n'y a aucune créature ;
- 10 chauves-souris apparaissent toutes les 2 secondes ;
- 5 skellingtons apparaissent toutes les 5 secondes ;
- 4 zombies apparaissent toutes les 6 secondes ;
- 3 fantômes baveux apparaissent toutes les 10 secondes.

Durant les quatre premières minutes de jeu il faut à Antonio :

- 6 secondes pour tuer 2 chauves-souris ;
- 20 secondes pour tuer 1 skellington ;
- 30 secondes pour tuer 1 zombie ;
- 40 secondes pour tuer 1 fantôme baveux.

Toutes les quatre minutes Antonio va s'améliorer, et dans le même temps, il pourra tuer 2 chauves-souris, 1 skellington, 1 zombie et 1 fantôme baveux supplémentaires.

Voici comment évoluent le nombre de chauves-souris, de skellingtons, de zombies et de fantômes baveux au fil des secondes :

```
au départ              :  0, 0, 0, 0
au bout de   1 seconde  :  0, 0, 0, 0
au bout de   2 secondes : 10, 0, 0, 0   <= chauves-souris +10
au bout de   3 secondes : 10, 0, 0, 0
au bout de   4 secondes : 20, 0, 0, 0   <= chauves-souris +10
au bout de   5 secondes : 20, 5, 0, 0   <= skellingtons +5
au bout de   6 secondes : 28, 5, 4, 0   <= chauves-souris +10 -2
                                           zombies +4
...
au bout de 238 secondes : 1112, 224, 149, 64   <= chauves-souris +10
au bout de 239 secondes : 1112, 224, 149, 64
au bout de 240 secondes : 1120, 228, 152, 66   <= chauves-souris +10 -2
                                                   skellingtons +5 -1
                                                   zombies +4 -1
                                                   fantômes +3 -1

À partir de maintenant, Antonio tuera 4 chauves-souris en 6 secondes, 2 skellingtons
en 20 secondes, 2 zombies en 30 secondes et 2 fantômes baveux en 40 secondes.
```

Au bout de 50 minutes de jeu, combien restera-t-il de chauves-souris, de skellingtons, de zombies et de fantômes baveux ?

Source : https://pydefis.callicode.fr/defis/C22_VampireSurvivors/txt

Exercice Bonus 4.41 (Pydéfis – SW I : À l'assaut de Gunray. Découpage de la porte blindée)

La visite de Qui-Gon Jinn et Obi-Wan Kenobi sur le vaisseau de la fédération qui commande le blocus de la planète Naboo tourne mal et Qui-Gon, armé de son sabre laser doit percer la porte blindée derrière laquelle Nute Gunray s'est terré.

On modélise la découpe au sabre laser de la manière suivante : chaque seconde, le sabre perce le blindage sur une épaisseur E (exprimée en centimètres). Durant cette seconde, il produit un volume de métal en fusion égal à environ

8E (exprimé en centimètres-cubes).

À mesure que le métal fond le sabre perce moins vite. Si le volume de métal en fusion autour du sabre est V (exprimé en centimètres-cubes) à un instant donné, dans le seconde qui suit, le sabre pénétrera dans le blindage d'une épaisseur $E = 3 - 0.005V$

Exemple

- Lorsque le sabre commence à percer (à l'instant $t = 0$), il n'y a pas de métal en fusion, donc $V = 0$.
- En une seconde le sabre perce sur une profondeur de $E = 3 - 0.005V = 3$ cm. Le sabre a donc percé 3 cm en une seconde. Mais il a aussi produit une quantité de métal en fusion égale à 8E, c'est-à-dire 24 cm^3.
- Durant la deuxième seconde, puisque le volume de métal en fusion est maintenant 24 cm^3, le sabre va pénétrer de $3 - 0.005 \times 24 = 2.88$ cm supplémentaires. Au total il aura donc percé en 2 secondes $3 + 2.88 = 5.88$ cm. Mais durant cette deuxième seconde, il aura fait fondre $8 \times 2.88 = 23.04$ cm^3. Il y a donc maintenant (au bout de 2 secondes) $24 + 23.04 = 47.04$ cm^3 de métal en fusion.
- Durant la troisième seconde, il percera donc de $3 - 0.005 \times 47.04 = 2.7648$ cm. Au bout de trois secondes, le sabre aura pénétré au total de $5.88 + 2.7648 = 8.6448$ cm.

Défi : la porte blindée fait 70 cm d'épaisseur. Au bout de combien de secondes Qui Gon aura-t-il percé la moitié de la porte ? Et au bout de combien de secondes aura-t-il percé toute la porte ?

Source : https://pydefis.pydefis.callicode.fr/defis/PorteBlindeeSabre/txt

Exercice Bonus 4.42 (Pydéfis – L'hydre de Lerne)

Histoire Pour son deuxième travail, Eurysthée demanda à Hercule de tuer l'Hydre, une sorte de dragon possédant plusieurs têtes et qui hantait les marais de Lerne. Pour mener à bien sa mission, Hercule, muni de sa seule épée, décida de trancher les têtes de l'Hydre. La tâche n'était pas si facile et les têtes repoussaient parfois lorsqu'il les coupait. Toutefois, la repousse des têtes de l'Hydre suivait une règle simple, ainsi que la stratégie d'Hercule :

- À chaque coup d'épée, Hercule coupait la moitié des têtes restantes.
- Si après une coupe, il restait un nombre impair de têtes, alors le nombre de têtes restantes triplait instantanément, et une tête supplémentaire repoussait encore.
- Si à un moment l'Hydre ne possédait plus qu'une seule tête, Hercule pouvait l'achever d'un coup d'épée supplémentaire.

Exemple

- Si l'Hydre a 6 têtes, Hercule en coupe la moitié au premier coup d'épée. Il en reste 3. Instantanément, le nombre de têtes triple et il en pousse une autre. Il y a maintenant 10 têtes.
- Au second coup d'épée, Hercule en coupe 5. Des têtes repoussent, il y en a maintenant 16.
- Au troisième coup d'épée, Hercule coupe 8 têtes. Il en reste 8. Rien ne repousse.
- Au quatrième coup d'épée, Hercule coupe 4 têtes, il en reste 4. Rien ne repousse.
- Au cinquième coup d'épée, Hercule coupe 2 têtes, il en reste 2. Rien ne repousse.
- Au sixième coup d'épée, Hercule coupe une tête. Il n'en reste plus qu'une.
- Le septième et dernier coup d'épée permet d'achever l'Hydre.

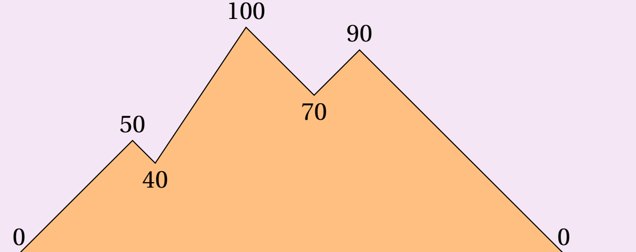
Si l'Hydre a 6 têtes, Hercule doit donc donner 7 coups d'épée pour la vaincre.

Défi : le nombre réel de têtes de l'Hydre est donné 8542. Combien de coups d'épée seront nécessaires pour venir à bout du monstre ?

Source : https://pydefis.callicode.fr/defis/Herculito02Hydre/txt

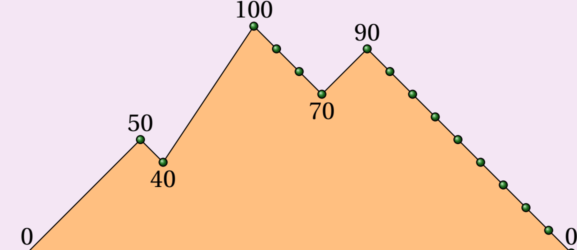
Exercice Bonus 4.43 (Pydéfis – Le sanglier d'Érymanthe)

Pour son quatrième travail, Eurysthée demanda à Hercule de capturer vivant le sanglier d'Érymanthe. Gigantesque, celui-ci dévastait avec rage le nord-ouest de l'Arcadie. Après avoir débusqué le sanglier, Hercule le poursuivit dans les montagnes en lui jetant des pierres. Le profil montagneux était assez accidenté et ressemblait à ceci :



Hercule, pour économiser ses forces, ne jetait des pierres que dans les descentes. Précisément, il jetait une pierre tous les 10 mètres (changement d'altitude). Ainsi, dans une descente de 30 mètres, il jetait 4 pierres.

On peut produire un relevé du profil des montagnes, en donnant les altitudes de chaque sommet et chaque col. Dans l'exemple qui précède, le relevé donnerait $0, 50, 40, 100, 70, 90, 0$.
À partir de ce relevé uniquement, on peut voir qu'il y a 3 descentes, de 10, 30 et 90 mètres.
Hercule jettera donc 16 pierres sur le sanglier.



Défi : un relevé du profil réel vous est donné en entrée :

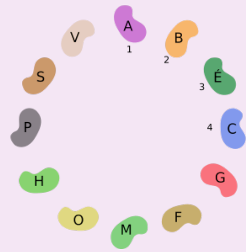
```
[0, 80, 60, 140, 100, 120, 50, 150, 30, 140, 120, 180, 10, 90, 80, 150, 20, 160, 50, 90,
↪ 80, 150, 140, 150, 20, 30, 10, 80, 70, 150, 50, 150, 130, 180, 60, 170, 60, 150, 120,
↪ 170, 80, 100, 70, 170, 140, 150, 110, 190, 10, 20, 10, 110, 20, 100, 50, 190, 120,
↪ 200, 30, 200, 160, 190, 20, 90, 30, 80, 60, 150, 110, 200, 80, 170, 140, 170, 40, 110,
↪ 20, 200, 60, 90, 80, 200, 10, 170, 40, 190, 60, 180, 70, 140, 90, 130, 110, 200, 90,
↪ 170, 40, 70, 20, 110, 70, 170, 90, 160, 80, 110, 100, 150, 130, 190, 50, 180, 70, 190,
↪ 150, 190, 110, 130, 110, 140, 60, 140, 20, 90, 20, 130, 40, 110, 90, 180, 120, 130,
↪ 90, 120, 100, 120, 40, 120, 40, 100, 10, 80, 60, 80, 30, 100, 80, 90, 40, 110, 20, 90,
↪ 20, 150, 70, 180, 70, 170, 60, 130, 90, 150, 20, 100, 90, 190, 170, 200, 160, 180, 20,
↪ 80, 30, 80, 50, 90, 80, 150, 130, 200, 140, 150, 110, 190, 20, 150, 100, 120, 40, 140,
↪ 80, 100, 60, 100, 50, 160, 60, 120, 70, 110, 50, 190, 0]
```

Vous devez indiquer à Hercule combien de pierre il aura à jeter sur le sanglier.

Source : https://pydefis.callicode.fr/defis/Herculito04Sanglier/txt

Exercice Bonus 4.44 (Pydéfis – Les dragées surprises)

Harry, Ron et Hermione sont dans le Poudlard express et Harry est heureux de partager ses dragées surprises de Bertie Crochue, achetées l'an passé chez Honeydukes. Il lui reste exactement 12 dragées : Aubergine, Bouillabaisse, Épinards, Chaussettes, Glace, Foie, Morve de Troll, Œuf Pourri, Herbe, Poubelle, Saucisse, et Vomi. Les partager va être difficile. Hermione propose de les disposer en cercle :



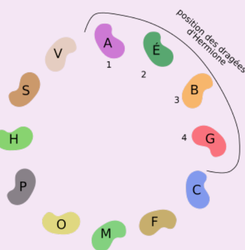
Puis, elle commence à compter : 1 sur Aubergine, 2 sur Bouillabaisse…

> Je vous propose de mélanger un peu tout ça. Je vais tourner en comptant, et toutes les 5 dragées, j'inverserai la dragée avec celle située juste avant :
>
> - 1, 2, 3, 4, 5 : j'échange Glace et Chaussette.
> - 1, 2, 3, 4, 5 : j'échange Poubelle et Herbe.
> - 1, 2, 3, 4, 5 : j'échange Épinards et Bouillabaisse.
> - etc.

 © G. Faccanoni

Je vais faire ça pendant un moment, ce qui va mélanger les dragées. Ensuite, je prendrai les 4 premières (positions 1, 2, 3 et 4), Ron les 4 suivantes, et Harry les 4 dernières.

Voici la position des dragées après ces 3 premiers échanges :



Ron a l'air soupçonneux. . . : "Combien d'échanges tu vas faire exactement ?"

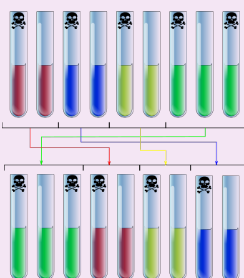
Hermione : "Oh, peu importe, je vais choisir un nombre au hasard, assez grand pour que ce soit bien mélangé."

Sachant qu'Hermione ne raffole pas des parfums qui restent, exceptés Aubergine, Épinards, Glace et Herbe, qui semblent un peu moins mauvais que les autres, et qu'elle souhaite terminer cette histoire au plus vite, combien d'échanges doit-elle faire (inclus les 3 qu'elle a déjà faits lors de son explication) ?

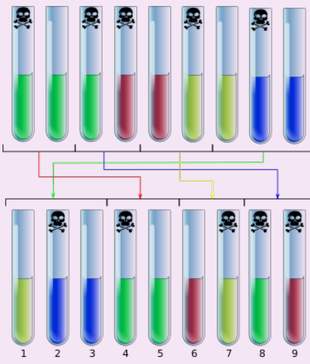
Source : https://pydefis.callicode.fr/defis/DrageesBertie/txt

Exercice Bonus 4.45 (Pydéfis – Les tubes à essai d'Octopus. Méli-mélo de poison)

Toujours prêt à mettre au défi son ennemi Spiderman, le Dr Octopus lui présente une série de 9 tubes à essais dont ceux occupant les positions impaires sont empoisonnés.
« Je vais mélanger les tubes en suivant le protocole suivant : je sépare la série en 4 tas, contenant respectivement 2 (en rouge), 2 (en bleu), 2 (en jaune) et 3 (en vert) tubes; les couleurs sont uniquement là pour mieux visualiser le fonctionnement. Puis, je place le dernier tas en premier (les verts), le premier tas en second (les rouges), le troisième tas (les jaunes) et enfin le second (les bleus). Ci-contre ce que cela donne. »



« Et je recommence. Je fais 4 tas de 2, 2, 2 et 3, puis je place le dernier tas en premier, etc. : »



« Je vais faire mon opération de mélange plusieurs fois aussi vite que mes bras mécaniques le permettent et tu vas devoir sélectionner les 4 tubes que tu boiras. Mouahahaha. Dans le cas qui précède, il faudrait que tu boives les tubes en position 1, 3, 5, et 6. »

« Mais attention. . . on ne va pas jouer avec 9 tubes de couleur marqués par une tête de mort. À la place, nous allons prendre 65 tubes incolores. Le premier, le troisième, le cinquième etc... seront empoisonnés. Au lieu de faire 4 tas de 2, 2, 2, et 3 tubes, je vais faire 4 tas de 14, 14, 14, et 23 tubes. Ainsi, si les tubes sont rangés dans l'ordre : 1, 2, 3,. . . 65, après une étape de mélange, les tubes seront dans l'ordre :

« 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28

« Je vais refaire cette opération de mélange plusieurs fois. À toi de me donner la liste des positions des 32 tubes que tu boiras. »

Exemple

- Si Octopus n'avait réalisé qu'un seul mélange, Spidey aurait dû boire les tubes occupant les positions : 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65
- Si Octopus avait réalisé en tout deux mélanges, Spidey aurait dû boire les tubes occupant les positions : 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 38, 40, 42, 44, 46, 48, 50, 53, 55, 57, 59, 62, 64

Défi : le Dr Octopus opère 48 mélanges. Aidez Spidey en lui indiquant la liste des tubes qu'il doit boire.

Source : https://pydefis.callicode.fr/defis/MelangeTubes/txt

Exercice Bonus 4.46 (Pydéfis – Désamorçage d'un explosif (I))

Une bombe dévastatrice a été placée à Los Angeles par un membre des Maîtres du mal. Black Widow a réussi à la trouver et doit maintenant tenter de la désamorcer.

Une fois la bombe ouverte, c'est un véritable sac de nœuds. Il y a 1000 fils numérotés de 0 à 999, et il faut en couper un seul, le bon, pour arrêter le compte à rebours.

Heureusement, les Avengers ont pu fournir un manuel de désamorçage à Black Widow. Celui-ci indique (il est en russe, nous avons traduit pour vous) :

> Le numéro du fil à couper peut être déduit du numéro de série de la bombe ainsi :
> 1. commencez par relever le numéro de série
> 2. coupez le numéro de série en 2. Les trois premiers chiffres forment le nombre U et les 3 derniers le nombre N
> 3. répétez N fois les opérations 4 et 5 suivantes en partant du nombre U
> 4. multipliez ce nombre par 13
> 5. ne conservez que les 3 derniers chiffres.
>
> Une fois cet algorithme terminé, le nombre obtenu est le numéro du fil à couper.

Testez votre code : par exemple, si le numéro de série avait été 317010, il aurait fallu couper le fil 133.

Défi : indiquez à Black Widow le numéro du fil à couper pour valider le défi si le numéro de série est 449149.

Source : https://pydefis.callicode.fr/defis/Desamorcage01/txt

Exercice Bonus 4.47 (Pydéfis – Méli Mélo de nombres)

Soit $a = 195$ et $b = 117$. À partir d'un nombre de 4 chiffres, comme $u = 9697$, on fabrique un nouveau nombre avec la méthode suivante :

- tout d'abord, on sépare les 2 derniers chiffres des deux premiers, ce qui donne deux nombres : 96 et 97, qu'on ajoute; nous obtenons 193.
- Puis, on multiplie ce résultat par a, et on ajoute b, ce qui donne 37752.
- Enfin, on calcule le reste de la division entière de 37752 par 9973, ce qui donne 7833.

Ainsi, à partir du nombre 9697, nous avons fabriqué le nouveau nombre 7833. On recommence cette opération n fois, ce qui construit une suite de nombres.

Éventuellement, un des nombres de la suite peut ne compter que 1, 2 ou 3 chiffres. L'opération est quand même possible. Pour calculer le nombre qui vient après 137, on sépare les deux derniers chiffres du nombre et on obtient les deux nombres 1 et 37 (attention, pas 13 et 7, mais 1 et 37), qu'on ajoute, etc. De même, si le nombre à transformer est 8, les deux nombres à ajouter seront 0 (le nombre de centaines), et 8 (le reste de la division par 100), etc.

Testez votre code : si $u = 3456$ et $n = 5$, il faut répondre 8641 car la suite de nombres calculés vaut 3456, 7694, 3348, 5939, 9254, 8641.

Défi : si $u = 3773$ et $n = 194$, quel nombre obtient-on si on applique la transformation ci-dessus n fois, en partant de u ?

Source : https://pydefis.callicode.fr/defis/Melange/txt

Exercice Bonus 4.48 (Pydéfis – Suite Tordue)

L'objet de cet exercice est de construire une suite de nombres en suivant certaines règles. Pour passer d'un nombre u au suivant il faut, après avoir écrit u sur 4 chiffres (en complétant éventuellement avec des 0 à gauche), ajouter le nombre formé des deux premiers chiffres de u avec le nombre formé des deux derniers chiffres de u, multiplier ce résultat par 191, et ajouter 161, prendre le reste de la division entière de ce nouveau résultat par 9973. Le nouveau nombre obtenu est le suivant dans la suite.

Testez votre code : par exemple, si u vaut 4267, on commence par calculer $42+67=109$. Puis on multiplie par 191 et on ajoute 161 pour trouver $109 \times 191+161=20980$. Enfin, on prend le reste de la division entière par 9973, ce qui nous donne 1034.

Testez votre code : autre exemple, si u vaut 112, on commence par ajouter 01 et 12, pour trouver 13. Puis on multiplie par 191 et on ajoute 161 pour trouver 2644. Enfin, on prend le reste de la division entière par 9973, ce qui nous donne 2644.

Défi : l'entrée du problème est constituée de 2 valeurs : u_1 (premier terme de la suite) et n. Que vaut u_n ?

Source : https://pydefis.callicode.fr/defis/SuiteTordue/txt

Exercice Bonus 4.49 (Pydéfis – Bombe à désamorcer)

Afin de pouvoir enfin opérer sur le terrain, il vous reste à passer un examen pratique : le désamorçage de bombe.

Vous pouvez manipuler 5 fils : un noir, un rouge, un vert, un jaune, et un bleu. Sur ce type de bombe, le désamorçage consiste à débrancher les 5 fils, dans le bon ordre. L'essentiel du problème est donc de déterminer dans quel ordre il faut débrancher les fils. Chaque couleur correspond à un numéro : 1 pour le noir, 2 pour le rouge, 3 pour le vert, 4 pour le jaune et 5 pour le bleu.

La donnée de 5 chiffres indique l'ordre de coupure des fils. **Par défaut, il s'agit de 34125**, ce qui signifie qu'il faut couper en premier le vert (3), en deuxième le jaune (4), en troisième le noir (1), en quatrième le rouge (2) et enfin le bleu (5).

Avant que la bombe ne soit amorcée, la combinaison de désamorçage (34125) a toutefois été modifiée. On lui a fait subir des permutations. Une permutation consiste à échanger 2 chiffres de la combinaison. On décrit la permutation en donnant les positions des 2 chiffres à échanger. Par exemple, la permutation 14 signifie qu'il faut échanger le premier chiffre et le quatrième.

Voici un exemple (pour tester votre code) : la combinaison de départ est le code sortie d'usine 34125. Supposons qu'on ait appliqué les permutations 14, 25, 13. Le code devient alors

$$34125 \xrightarrow[\text{permutation 14}]{} 2413524135 \xrightarrow[\text{permutation 25}]{} 2513425134 \xrightarrow[\text{permutation 13}]{} 15234$$

Le résultat après les 3 permutations est 15234. Cela signifie qu'il faudra couper en premier le fil noir (1), en deuxième le fil bleu (5), en troisième le fil rouge (2), en quatrième le fil vert (3) et en dernier le fil jaune (4).

Dans quel ordre il faut couper les fils si la liste des permutations effectuées avant amorçage est la suivante ?

41, 35, 13, 51, 34, 42, 23, 31, 13, 51, 32, 32, 43, 24, 54, 34, 34, 41, 35, 52, 15, 12, 43, 52, 14,
24, 35, 13, 12, 31, 51, 31, 51, 35, 45, 15, 21, 42, 25, 32, 34, 21, 13, 12, 51, 13, 45, 52, 14, 54, 34,
34, 42, 34, 21, 51, 54, 34, 51, 43, 31, 24, 31, 23, 51, 25, 23, 53, 12, 35, 41, 31, 15, 35, 45, 24, 45,
12, 34, 45, 12, 12, 12, 15, 35, 51, 34, 12, 54, 32, 12, 25, 41, 45, 32, 53, 35, 45, 41, 3

Source : https://pydefis.callicode.fr/defis/Desamorcage03/txt

Exercice 4.50 (Devine le résultat)

Quel résultat donne le code suivant ? Après avoir écrit la réponse, vérifier avec l'ordinateur.

```
L_1 = list(range(0,11,2))
```

```
L_2 = list(range(1,12,2))
L = []
for i in range(len(L_1)):
    L = L + [L_1[i]] + [L_2[i]] # idem que L.append(L_1[i]).append(L_2[i])
print(L_1)
print(L_2)
print(L)
```

Correction

```
[0, 2, 4, 6, 8, 10]
[1, 3, 5, 7, 9, 11]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Exercice 4.51 (Cadeaux)

Vous recevez un cadeau de 1 centime aujourd'hui (0.01 €). Demain, vous recevrez le double (2 centimes). Le lendemain, vous recevrez à nouveau le double (4 centimes). Etc. Une fois que vous aurez reçu plus de 1 million d'euros, vos cadeaux cesseront. Écrivez le code pour déterminer combien de jours vous recevrez des cadeaux, combien sera le dernier cadeau et combien vous recevrez au total.

Correction

```
cadeau_total, cadeau_du_jour, jour = .01, .01, 1
while cadeau_total < 1.e6:
    jour += 1
    cadeau_du_jour *= 2
    cadeau_total += cadeau_du_jour

print(f'''Le jour {jour} mon cadeau est de {cadeau_du_jour}€ \
ainsi la somme totale est {cadeau_total:.2f}€.''')
```

```
Le jour 27 mon cadeau est de 671088.64€ ainsi la somme totale est 1342177.27€.
```

Exercice 4.52 (Affichage)

Pour $n \in \mathbb{N}$ donné, afficher $1+2+\cdots+n=\mathrm{N}$ où N est à calculer.

Correction

```
n = 10
for i in range(1,n):
    print(i," + ", end=" ")
print(n,"=",int(n*(n+1)/2))
```

$1+2+3+4+5+6+7+8+9+10=55$

Exercice Bonus 4.53 (Pydéfis – Insaisissable matrice)

On considère la matrice suivante

$$\begin{pmatrix} 36 & 19 & 27 & 36 & 7 & 10 \\ 2 & 18 & 3 & 33 & 2 & 21 \\ 26 & 27 & 4 & 22 & 30 & 31 \\ 29 & 36 & 7 & 20 & 6 & 30 \\ 30 & 6 & 14 & 23 & 15 & 13 \\ 22 & 10 & 10 & 35 & 15 & 22 \end{pmatrix}$$

Cette matrice va évoluer au cours du temps, et le contenu m_{ij} d'une case est transformé, à chaque étape, en $(11m_{ij}+4)\%37$ où $a\%b$ donne le reste de la division entière de a par b. À chaque étape de calcul, tous les nombres de la matrice sont simultanément modifiés.

Défi : que vaut la somme des valeurs contenues dans la matrice après application de 23 étapes ?

Source : https://pydefis.callicode.fr/defis/AlgoMat/txt

Exercice 4.54 (Maximum d'une liste de nombres sans la fonction `max`)

Soit une liste de nombres. Trouver le plus grand et le plus petit élément de cette liste **sans utiliser les fonctions prédéfinies** `max`, `min`, `sorted` ni la méthode `sort`.

Correction

```
L = [10, 5, 15, -2, 17, -22]
mymax = L[0]
mymin = L[0]
for ell in L:
	if ell>mymax:
		mymax = ell
	if ell<mymin:
		mymin = ell
print(f"L={L}, mymax(L)={mymax}, mymin(L)={mymin}")
```

```
L=[10, 5, 15, -2, 17, -22], mymax(L)=17, mymin(L)=-22
```

Exercice 4.55 (Chaîne la plus longue d'une liste de strings)

Soit une liste de chaînes de caractères (toutes de longueurs différentes). Trouver la plus longue et la plus courte chaîne de cette liste **sans utiliser les fonctions prédéfinies** `max`, `min`, `sorted` ni la méthode `sort`.

Correction

```
L = ["Je", "Vous", "Bonjour", "Adieu"]
mymax = L[0]
mymin = L[0]
for ell in L:
	if len(ell)>len(mymax):
		mymax = ell
	if len(ell)<len(mymin):
		mymin = ell
print(f"L={L}, mymax(L)={mymax}, mymin(L)={mymin}")
```

```
L=['Je', 'Vous', 'Bonjour', 'Adieu'], mymax(L)=Bonjour, mymin(L)=Je
```

☆ Exercice Bonus 4.56 (Défi Turing n°9 – triplets pythagoriciens)

Le triplet d'entiers naturels non nuls (a, b, c) est pythagoricien si $a^2 + b^2 = c^2$. Par exemple, $(3, 4, 5)$ est un triplet pythagoricien.

Parmi les triplets pythagoriciens (a, b, c) tels que $a + b + c = 3600$, donner le produit $a \times b \times c$ le plus grand.

Correction

Nous n'avons pas besoin de faire varier les 3 valeurs, en faire varier 2 suffit car si nous connaissons, par exemple, la valeur de c et a, nous pouvons en déduire celle de b en résolvant l'équation $a + b + c = p$ où p est le périmètre (ici $p = 3600$).

De plus, nous savons que $a < c$, $b < c$ et nous pouvons étudier seulement les cas $a < b$. Dans ce cas, comme $a < b < c$ et $a + b + c = p$, alors $3a < p < 3c$.

```
l = []
p = 3600
for a in range(1,int(p/3)+1): # au lieu de range(1,p-1) car a<b<c implique 3a<a+b+c=p
	for c in range(int(p/3),p): # au lieu de range(a,p-a) car a<b<c implique p=a+b+c<3c
		b = p-a-c
		if (a*a+b*b)==(c*c):
			l.append(a*b*c)
print(max(l))
```

```
1654329600
```

Exercice Bonus 4.57 (Défi Turing n°11 - nombre miroir)

On appellera “miroir d’un nombre n” le nombre n écrit de droite à gauche. Par exemple, le miroir de 7423 est 3247. Quel est le plus grand nombre inférieur à 10 millions ayant la propriété : “miroir de $n = 4n$” ?

Correction

```python
l = []
for n in range(10**7):
    miroir = int(str(n)[::-1])
    if miroir==4*n:
        l.append(n)
print(max(l))
```

```
2199978
```

Exercice Bonus 4.58 (Défi Turing n°13 – Carré palindrome)

Un nombre palindrome se lit de la même façon de gauche à droite et de droite à gauche (voir exercice 1.19). Un nombre à un chiffre est palindrome. Le plus petit carré palindrome ayant un nombre pair de chiffres est $698896 = 836^2$. Quel est le carré palindrome suivant ?

Correction

```python
for n in range (10**7):
    sc = str(n*n)
    if len(sc)%2==0 and sc[::-1]==sc:
        print(f"n = {n} car n^2 = {n*n}")
```

```
n = 836 car n^2 = 698896
n = 798644 car n^2 = 637832238736
```

Exercice Bonus 4.59 (Défi Turing n°43 – Carré palindrome)

Un nombre palindrome se lit de la même façon de gauche à droite et de droite à gauche. Un nombre à un chiffre est palindrome. Donner la somme des nombres dont le carré est un palindrome d’au plus 13 chiffres.

Correction

```python
s = 0
for n in range (10**7):
    sc = str(n*n)
    if len(sc)<=13 and sc[::-1]==sc:
        s += n
print(s)
```

```
27974694
```

Exercice 4.60 (Entier palindrome dans une base b)

Déterminer si le nombre entier donné est un palindrome ou non en base B. Par exemple,

- 6 s’écrit 110 en base 2 ($6 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2$) donc la réponse est `False`
- 34 s’écrit 114 en base 5 ($34 = 4 \times 5^0 + 1 \times 5^1 + 1 \times 5^2$) donc la réponse est `False`
- 455 s’écrit 111000111 en base 2 donc la réponse est `True`

Correction

```
# d en base 10 -> b en base B
for d,B in [ (455,2) , (3148,16) , (1442,10) ]:
    n = d
    b = []
    while n > 0:
        n, res = divmod(n,B)
        b.append(res)
    b = ''.join([str(c) for c in b] )[::-1]
    print( f"Le nombre d'écriture décimale {d}, s'écrit {b} en base {B}. Est-il palindrome?
    ↪ {b==b[::-1]}" )
```

```
Le nombre d'écriture décimale 455, s'écrit 111000111 en base 2. Est-il palindrome? True
Le nombre d'écriture décimale 3148, s'écrit 21421 en base 16. Est-il palindrome? False
Le nombre d'écriture décimale 1442, s'écrit 1442 en base 10. Est-il palindrome? False
```

☆ Exercice Bonus 4.61 (Défi Turing 22 – Les anagrammes octuples)

Mathilde a trouvé deux nombres de six chiffres étonnants. Lorsqu'on les multiplie par 8, on obtient un nombre de six chiffres qui s'écrit avec les mêmes chiffres rangés dans un ordre différent. Quels sont les nombres de Mathilde ?

Correction

```
for n in range(100000,int(1000000/8)+1):
    if sorted(str(n))==sorted(str(8*n)):
        print(n)
```

```
113967
116397
```

☆ Exercice Bonus 4.62 (Défi Turing n°21 – Bonne année 2013 !)

2013 a une particularité intéressante : c'est la première année depuis 1987 à être composée de chiffres tous différents. Une période de 26 ans sépare ces deux dates.

Entre l'an 1 et 2013 (compris) :

1) Combien y a-t-il eu d'années composées de chiffres tous différents ? (les années de l'an 1 à l'an 9 seront comptées dans ce nombre).
2) Quelle a été la durée (en années) de la plus longue période séparant deux dates ayant des chiffres tous différents ?

Donner le produit des résultats de 1) et 2).

Correction

```
l = []
for n in range(1,2014):
    stringa = str(n)
    if len(stringa)==len(set(stringa)):
        l.append(n)

Q1 = len(l)
ecarts = [ l[i]-l[i-1] for i in range(1,len(l)) ]
Q2 = max(ecarts)
print(f"{Q1 = }, {Q2 = }, {Q1*Q2 = }")
```

```
Q1 = 1243, Q2 = 105, Q1*Q2 = 130515
```

Exercice Bonus 4.63 (Pydéfis – La suite Q de Hofstadter)

La Q-Suite est définie ainsi :

$$\begin{cases} Q_1 = 1, \\ Q_2 = 1, \\ Q_n = Q_{n-Q_{n-1}} + Q_{n-Q_{n-2}} & \text{pour } n > 2. \end{cases}$$

Que vaut $\sum_{i=2313}^{i=2375} Q_i$?

Source : https://pydefis.callicode.fr/defis/QSuite/txt

Exercice Bonus 4.64 (Pydéfis – L'escargot courageux)

Un escargot veut gravir une tour de 324 mètres. Le premier jour, il monte de x centimètres. La première nuit, il glisse (vers le bas) de y centimètres. Chaque jour supplémentaire, il monte de 1 centimètre(s) de moins que la journée précédente. En revanche, la nuit il glisse toujours de y centimètres.

Dans la région où est située cette tour, il pleut toutes les 8 nuits. Lorsque ça se produit, au bout de la nuit, l'escargot se retrouve au même endroit que 48 heures auparavant.

En appelant le jour 0 celui de son départ (il part le matin), et sachant qu'il vient de pleuvoir la nuit dernière, quel jour l'escargot arrive-t-il en haut de la tour si $x = 1370$ et $y = 280$?

Pour vérifier la compréhension de l'énoncé, voici le début du parcours de l'escargot, pour $x = 1330$ et $y = 300$

```
Fin du jour     0 : altitude=1330 cm
Fin de la nuit 0 : altitude=1030 cm
Fin du jour     1 : altitude=2359 cm
Fin de la nuit 1 : altitude=2059 cm
Fin du jour     2 : altitude=3387 cm
Fin de la nuit 2 : altitude=3087 cm
Fin du jour     3 : altitude=4414 cm
Fin de la nuit 3 : altitude=4114 cm
Fin du jour     4 : altitude=5440 cm
Fin de la nuit 4 : altitude=5140 cm
Fin du jour     5 : altitude=6465 cm
Fin de la nuit 5 : altitude=6165 cm
Fin du jour     6 : altitude=7489 cm
Fin de la nuit 6 : altitude=7189 cm
Fin du jour     7 : altitude=8512 cm
Fin de la nuit 7 : altitude=6165 cm
Fin du jour     8 : altitude=7487 cm
Fin de la nuit 8 : altitude=7187 cm
Fin du jour     9 : altitude=8508 cm
Fin de la nuit 9 : altitude=8208 cm
```

Source : https://pydefis.callicode.fr/defis/Escargot/txt

Exercice Bonus 4.65 (Pydéfis – Mon beau miroir…)

On appelle l'image miroir d'un nombre, le nombre lu à l'envers. Par exemple, l'image miroir de 324 est 423. Un nombre est un palindrome s'il est égal à son image miroir. Par exemple, 52325, ou 6446 sont des palindromes. À partir d'un nombre de départ, nous pouvons l'ajouter à son image miroir, afin d'obtenir un nouveau nombre, puis recommencer avec ce nouveau nombre jusqu'à obtenir un palindrome. À ce nombre de départ correspondent ainsi 2 valeurs : le palindrome obtenu, ainsi que le nombre d'addition qu'il a fallu faire pour l'obtenir. Par exemple, pour le nombre de départ 475, nous obtenons :

```
475 + 574 = 1049
1049 + 9401 = 10450
10450 + 5401 = 15851
```

Le denier nombre, 15851, est un palindrome. Pour le nombre de départ 475, nous atteignons donc le palindrome 15851 en 3 étapes.

Dans cet exercice, l'entrée est une séquence de nombres. Vous devez répondre en donnant une séquence de couples : le palindrôme obtenu, et le nombre d'étapes. Par exemple, si l'entrée est `(844, 970, 395, 287)` vous devrez obtenir `[[7337, 3] , [15851, 3] , [881188, 7] , [233332, 7]]`

Qu'obtient-on avec la séquence d'entrée : `746, 157, 382, 461, 885, 638, 462, 390, 581, 692`?

Source : https://pydefis.callicode.fr/defis/MiroirAjout/txt

Exercice Bonus 4.66 (Pydéfis – Persistance)

Il s'agit ici d'étudier les «suites de persistance». Ces suites sont obtenues, à partir de n'importe quel nombre entier, en calculant le produit de ses chiffres, et en recommençant. La suite de persistance de l'entier 347, par exemple est

$$347 \to 3 \times 4 \times 7 = 84 \to 8 \times 4 = 32 \to 3 \times 2 = 6$$

On s'arrête lorsqu'il ne reste plus qu'un chiffre.

On cherche à savoir quels sont les chiffres sur lesquels on tombera le plus souvent (ici, nous sommes tombés sur le chiffre 6). 0 est exclu de cette étude, car il est obtenu en écrasante majorité (dès qu'il y a un 0 dans un nombre, la suite s'arrête sur 0).

Indiquer combien de fois chaque chiffre entre 1 et 9 a été obtenu comme terminaison de la suite de persistance, pour tous les entiers entre $L = 1701$ et $R = 4581$.

Par exemple, si les nombres donnés étaient $L = 371$ et $R = 379$, il faudrait répondre avec la liste $[0,2,0,1,0,3,0,2,0]$. En effet, si on construit toutes les suites de persistance

$$\begin{aligned}
&371 \to 21 \to 2\\
&372 \to 42 \to 8\\
&373 \to 63 \to 18 \to 8\\
&374 \to 84 \to 32 \to 6\\
&375 \to 105 \to 0\\
&376 \to 126 \to 12 \to 2\\
&377 \to 147 \to 28 \to 16 \to 6\\
&378 \to 168 \to 48 \to 32 \to 6\\
&379 \to 189 \to 72 \to 14 \to 4
\end{aligned}$$

On obtient donc le chiffre 1 zéro fois, le chiffre 2 deux fois, le chiffre 3 zéro fois, le chiffre 4 une fois... le chiffre 8 deux fois et le chiffre 9 zéro fois. La réponse est en conséquence $0,2,0,1,0,3,0,2,0$

Source : https://pydefis.callicode.fr/defis/Persistance/txt

Exercice Bonus 4.67 (Pydéfis – Toc Boum)

Défi : dans cet exercice, un nombre entier n vous est donné en entrée. Ce nombre peut s'écrire : $n = 13a + 7b$ où a et b sont des entiers strictement positifs. Si plusieurs couples a, b conviennent, il faut trouver le couple tel que a et b soient des nombres les plus proches possibles.

Testez votre code : si l'entrée fournie est 287, les couples a, b possibles sont $(7,28)$ $(14,15)$ et $(21,2)$. Le couple a, b solution (celui pour lequel a et b sont les plus proche) est donc $(14,15)$.

Source : https://pydefis.callicode.fr/defis/TocBoum/txt

Exercice Bonus 4.68 (Pydéfis – Les juments de Diomède)

Histoire : pour son huitième travail, Eurysthée demanda à Hercule de lui ramener les juments de Diomède. Ces quatre féroces animaux se nourrissaient de chair humaine et Diomède, un des fils d'Arès, les nourrissait avec les voyageurs de passage.

Hercule se rendit donc en Thrace et entreprit de calmer la faim des juments afin de les capturer. N'ayant jamais eu l'intention de sacrifier ses amis ou les innocents de passage, il avait pris soin de faire embarquer un grand nombre de paquets de croquettes pour chat sur son bateau (Hercule voyageait à pied, car il souffrait du mal de mer, mais son équipe voyageait en bateau).

Défi : sachant que chacun des quatre animaux, pour être repu, consommait 131 kg de croquettes et qu'Hercule possédait à son bord 20 sacs de 7 kg, 20 sacs de 11 kg et 20 sacs de 13 kg, combien de sacs de 7, 11 et 13 kg devrait-il débarquer pour apporter très exactement la quantité de nourriture nécessaire aux juments, ni moins (elle ne seraient pas repues), ni plus (ne pas pouvoir finir leur assiette mettait les juments particulièrement en colère) ? Parmi toutes les solutions possibles, Hercule voulait débarquer le moins de sacs. Et parmi les solutions qui satisfaisaient ce critère, il devait essayer, pour épargner ses compagnons, de débarquer le moins de sacs de 13 kg.

Testez votre code : si Hercule avait eu à son bord 7 sacs de 5 et 7 sacs de 9 kg, et si les juments avaient chacune consommé 23 kg, alors Hercule aurait dû débarquer 12 sacs : 1 sac de 5 kg, 6 sacs de 7 kg et 5 sacs de 9 kg. En effet, le total fait bien $1 \times 5 + 6 \times 7 + 5 \times 9 = 92$ kg $= 4 \times 23$ kg. La solution à ce problème serait donc 1, 6, 5. Remarquez que la solution 5, 7, 2 ne convient pas car elle nécessite plus de sacs (14 sacs au lieu de 12). La solution 2, 4, 6 ne convient pas non plus, car même si elle ne nécessite aussi que 12 sacs, il faut débarquer 6 gros sacs (au lieu de 5 pour la solution valide). Enfin, la solution 0, 8, 4 ne convient pas non plus, car on n'a que 7 sacs de chaque sorte, et non 8.

Source : https://pydefis.callicode.fr/defis/Herculito08Juments/txt

Exercice Bonus 4.69 (Pydéfis – Produit et somme palindromiques)

Nous cherchons ici les nombres entiers a, b, c, d tels que le produit $abcd$ et la somme $a+b+c+d$ soient des palindromes.

Par exemple :

- si $a = 15$, $b = 71$, $c = 59$, $d = 87$, le produit $abcd = 5466645$ est un palindrome ainsi que la somme $a+b+c+d = 232$;
- si $a = 13$, $b = 47$, $c = 98$, $d = 68$, le produit $abcd = 4071704$ est un palindrome, ce qui n'est pas le cas de la somme $a+b+c+d = 226$;
- si $a = 12$, $b = 4$, $c = 68$, $d = 37$, le produit $abcd = 120768$ n'est pas un palindrome, alors que la somme $a+b+c+d = 121$ l'est.

Défi : on donne en entrée les bornes `mini = 25` et `maxi = 95` (incluses) de a, b, c et d. Trouvez combien de quadruplets a, b, c, d sont tels que $abcd$ et $a+b+c+d$ soient tous les deux des palindromes.

Testez votre code : si `mini = 10` et `maxi = 28`, alors il y a 5 solutions :

$$(11,11,11,11) \to 11 \times 11 \times 11 \times 11 = 14641 \text{ et } 11+11+11+11 = 44$$
$$(11,11,19,25) \to 11 \times 11 \times 19 \times 25 = 57457 \text{ et } 11+11+19+25 = 66$$
$$(13,13,14,26) \to 13 \times 13 \times 14 \times 26 = 61516 \text{ et } 13+13+14+26 = 66$$
$$(14,14,14,24) \to 14 \times 14 \times 14 \times 24 = 65856 \text{ et } 14+14+14+24 = 66$$
$$(17,21,22,28) \to 17 \times 21 \times 22 \times 28 = 219912 \text{ et } 17+21+22+28 = 88$$

Source : https://pydefis.callicode.fr/defis/Palindromes/txt

 © G. Faccanoni

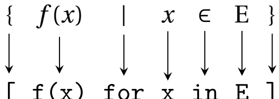
CHAPITRE 5

Définitions en compréhension

L'idée derrière l'utilisation des expressions en compréhension est de vous permettre d'écrire et de raisonner dans le code de la même manière que vous feriez les mathématiques à la main.

5.1. Listes en compréhension

Les listes définies par compréhension permettent de générer des listes de manière très concise sans avoir à utiliser des boucles. La syntaxe pour définir une liste par compréhension est très proche de celle utilisée en mathématiques pour définir un ensemble :



Syntaxe :

```
[ function(item) for item in list if condition(item) ]
```

La boucle	La liste en compréhension

`L=[]`
`for i in range(5):`
`    L.append(i**2)`
`print(L)`
	`L=[i**2 for i in range(5)]`
	`print(L)`

affiche [0, 1, 4, 9, 16]	affiche la même liste [0, 1, 4, 9, 16]

Si on compare le temps d'exécution, la deuxième méthode est plus performante.

Voici quelques exemples :

- Après avoir définie une liste E, on affiche d'abord les triples des éléments de la liste liste donnée, ensuite des listes avec les éléments de E et leurs cubes, puis les triples des éléments de la liste liste donnée si l'élément de E est > 5 ou si le carré est < 50 :

```
>>> E = [2, 4, 6, 8, 10] # E=list(range(2,11,2))

>>> [ 3*x for x in E ]
[6, 12, 18, 24, 30]

>>> [ (x,x**3) for x in E ]
[(2, 8), (4, 64), (6, 216), (8, 512), (10, 1000)]

>>> [ 3*x for x in E if x>5 ]
[18, 24, 30]

>>> [ 3*x for x in E if x**2<50 ]
[6, 12, 18]
```

- On peut utiliser des boucles imbriquées :

```
>>> E = list(range(2,11,2))
>>> F = list(range(3))
>>> L = [x*y for x in E for y in F]
>>> print(L)
[0, 2, 4, 0, 4, 8, 0, 6, 12, 0, 8, 16, 0,
 10, 20]
```

```
>>> # IDEM que
>>> L = []
>>> for x in E:
...     for y in F:
...         L.append(x*y)
...
>>> print(L)
[0, 2, 4, 0, 4, 8, 0, 6, 12, 0, 8, 16, 0,
 10, 20]
```

- Après avoir définie une liste, on affiche d'abord les carrés des éléments de la liste liste donnée, ensuite les nombres paires, enfin les carrés pairs. On montre l'équivalent sans l'écriture en compréhensions :

```
>>> # E = list(range(1,8))
>>> E = [1, 2, 3, 4, 5, 6, 7]
>>> L = [x**2 for x in E] # {x^2 | x∈E}
>>> print(L)
[1, 4, 9, 16, 25, 36, 49]
```

```
>>> L = []
>>> for x in E:
...     L.append(x**2)
...
>>> print(L)
[1, 4, 9, 16, 25, 36, 49]
```

```
>>> E = [1, 2, 3, 4, 5, 6, 7]
>>> L = [x for x in E if x%2 == 0] # pairs
>>> print(L)
[2, 4, 6]
```

```
>>> L = []
>>> for x in E:
...     if x%2 == 0:
...         L.append(x**2)
...
>>> print(L)
[4, 16, 36]
```

```
>>> E = [1, 2, 3, 4, 5, 6, 7]
>>> L = [x**2 for x in E if x**2%2 == 0] #
 carres pairs
>>> print(L)
[4, 16, 36]
```

```
>>> L = []
>>> for x in E:
...     if x**2%2 == 0:
...         L.append(x**2)
...
>>> print(L)
[4, 16, 36]
```

```
>>> E = [1, 2, 3, 4, 5, 6, 7]
>>> L = [x for x in [a**2 for a in E] if
 x%2 == 0]
>>> print(L)
[4, 16, 36]
```

```
>>> A = []
>>> for a in E:
...     A.append(a**2)
...
>>> L = []
>>> for x in A:
...     if x%2 == 0:
...         L.append(x)
...
>>> print(L)
[4, 16, 36]
```

- Une autre façon de créer la liste $[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}]$ avec un itérateur généré par la fonction `range` :

```
>>> [1/n for n in range(1,5)]
[1.0, 0.5, 0.3333333333333333, 0.25]
```

- On peut même utiliser des conditions `if...else` :

```
>>> [x+1 if x >= 3 else x+5 for x in range(6)]
[5, 6, 7, 4, 5, 6]
```

- Listes en compréhension imbriquées : transposée d'une matrice.

```
>>> M = [[1,2,3],[4,5,6]]
>>> M_transpose = [ [row[i] for row in M] for i in range(len(M)) ]
>>> print(f'{M}\n{M_transpose}')
```

```
[[1, 2, 3], [4, 5, 6]]
[[1, 4], [2, 5]]
```

Notons qu'on obtient (presque) le même résultat avec [1]

```
>>> M = [[1,2,3],[4,5,6]]
>>> M_transpose = list(zip(*M))
>>> print(f'{M}\n{M_transpose}')
[[1, 2, 3], [4, 5, 6]]
[(1, 4), (2, 5), (3, 6)]
```

5.2. ☆ Dictionnaires en compréhension

La syntaxe générale est

```
dico = {key:value for (key,value) in dictonary.items()}
```

Exemples :

1. on crée un dictionnaire `dico1` et on génère par compréhensions un `dico2` dans lequel chaque valeur est doublée :

```
dico1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
print(dico1)

dico2 = {k:v*2 for (k,v) in dico1.items()}
print(dico2)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
{'a': 2, 'b': 4, 'c': 6, 'd': 8, 'e': 10}
```

2. on crée un dictionnaire `dico1` et on génère par compréhensions un `dico3` dans lequel chaque clé est "doublée" (au sens des chaînes de caractères) :

```
dico1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
print(dico1)

dico3 = {k*2:v for (k,v) in dico1.items()}
print(dico3)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
{'aa': 1, 'bb': 2, 'cc': 3, 'dd': 4, 'ee': 5}
```

5.3. ☆ Ensembles en compréhension

De la même manière que les listes en compréhension, on peut définir un ensemble en compréhension :

```
>>> {skill for skill in ['SQL', 'SQL', 'PYTHON', 'PYTHON']}
{'SQL', 'PYTHON'}
```

La sortie ci-dessus est un ensemble de 2 valeurs car les ensembles ne peuvent pas avoir plusieurs occurrences du même élément. On peut bien-sûr ajouter des conditions dans la construction :

```
>>> {skill for skill in ['GIT', 'PYTHON', 'SQL'] if skill not in {'GIT', 'PYTHON', 'JAVA'}}
{'SQL'}
```

1. Il s'agit cette fois-ci d'une liste de tuples et non plus d'une liste de listes.

5.4. Exercices

Exercice 5.1 (Sous-listes)
Soit L une liste de nombres. Construire **en compréhensions** les sous-listes suivantes

- `P_val` qui ne contient que les éléments pairs de la liste L,
- `P_idx` qui ne contient que les éléments de la liste L qui sont en position paire dans la liste L.

Correction

```
L = [1,3,6,2,3,4,9,19,21] # exemple
P_val = [x for x in L if x%2==0]
P_idx = [L[i] for i in range(len(L)) if i%2==0] # idem que L[::2]
print(f"{L = }, {P_val = }, {P_idx = }")
```

```
L = [1, 3, 6, 2, 3, 4, 9, 19, 21], P_val = [6, 2, 4], P_idx = [1, 6, 3, 9, 21]
```

Exercice 5.2 (Somme des carrés)
Soit L une liste de nombres. Construire en compréhensions la liste des carrés des éléments de L. En calculer ensuite la somme. Par exemple, si L=`[0,1,2]`, on doit obtenir s=5.

Correction

```
L = list(range(3)) # exemple
C = [x**2 for x in L]
s = sum(C)
print(f"{L = }, {C = }, {s = }")
```

```
L = [0, 1, 2], C = [0, 1, 4], s = 5
```

Exercice 5.3 (Conversion)
Soit S une liste de chaînes de caractères; chaque chaîne étant constitué que de chiffres. Construire par compréhension la liste L qui contient les nombres entiers associés à chaque chaîne. Par exemple, si `S = ["5", "10", "15"]`, alors `L = [5, 10, 15]`.

Correction

```
S = ["5", "10", "15"]
L = [int(x) for x in S]
print(f"{L = }, {S = }")
```

```
L = [5, 10, 15], S = ['5', '10', '15']
```

Exercice 5.4 (Chaînes de caractères)
Soit s une chaîne de caractères. Construire en compréhension la liste qui contient chaque caractère. Exemple : si s=`"Ciao"`, on doit obtenir la liste `["C", "i", "a", "o"]`.

Correction

```
>>> s = "Ciao"
>>> L = [c for c in s]
>>> print(L)
['C', 'i', 'a', 'o']
```

ou, plus simplement

```
>>> s = "Ciao"
>>> L = list(s)
>>> print(L)
['C', 'i', 'a', 'o']
```

Exercice 5.5 (Somme des chiffres d'un nombre)

Soit $n \in \mathbb{N}$. Générer par compréhension la liste qui contient chaque chiffre de n. Attention, chaque élément de cette liste doit être un `int`. Calculer ensuite la somme de ces éléments. Par exemple, si `n=30071966`, on doit obtenir 32.

Correction
On converti ce nombre en une chaîne de caractères (`str(n)`). On lit les chiffres un par un (`for c in str(n)`) et on les converti en entier (`int(c)`). Enfin on additionne les éléments de cette liste :

```
n = 30071966
s = sum([int(c) for c in str(n)])
print(f"La somme des chiffres de {n} est {s}")
```

```
La somme des chiffres de 30071966 est 32
```

Exercice 5.6 (Défi Turing n°5 – somme des chiffres d'un nombre)

$2^{15} = 32768$ et la somme de ses chiffres vaut $3+2+7+6+8=26$. Que vaut la somme des chiffres composant le nombre 2^{2222} ?

Correction
Pour résoudre ce problème, on trouve d'abord la valeur de 2^{2222}, on convertit ce nombre en chaîne de caractères, on lis les chiffres un par un, on les convertit en entier, enfin on les additionne :

```
print(sum([int(x) for x in str(2**15)]))
print(sum([int(x) for x in str(2**2222)]))
```

```
26
2830
```

Exercice 5.7 (Liste de Moyennes)

Soit `P` une liste de listes de nombres. Définir par compréhension une liste qui contient les moyennes arithmétiques des sous-listes de `P`.

Par exemple, si `P = [ [1,2,3], [4,5,6,7], [5,-1,8], [10,11] ]`, on doit obtenir [2.0, 5.5, 4.0, 10.5].

Correction

```
>>> P = [ [1,2,3], [4,5,6,7], [5,-1,8], [10,11] ]
>>> [ sum(l)/len(l) for l in P ]
[2.0, 5.5, 4.0, 10.5]
```

Exercice 5.8 (Morceau)

Diviser une liste en plusieurs listes plus petites d'une taille spécifiée. Par exemple, si `L = [ 1,2,3,4,5,6,7,8,9,10 ]` et `size = 4`, on doit obtenir [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10]].

Correction

```
>>> L = [ 1,2,3,4,5,6,7,8,9,10 ]
>>> size = 4
>>> [ L[i:i+size] for i in range(0,len(L), size) ]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10]]
```

Exercice 5.9 (Position du minimum d'une liste de nombres)

Soit une liste de nombres. Trouver la position du plus petit élément de cette liste (s'il y en a plusieurs, renvoyer le premier indice).

Suggestion : utiliser la fonction `enumerate()` pour créer une liste de tuple. Comment agit la fonction `min()` sur une liste de tuple ?

Correction
La fonction *built-in* `enumerate()` renvoie un tuple `(index, element)` pour chaque élément de la liste.

Lorsque la fonction `min()` est utilisée sur une liste de tuples, elle compare les tuples en utilisant leur première valeur par défaut.

```
L = [10, -22, 5, 15, -2, 17, -22]

tmp = [ (v,i) for i,v in enumerate(L) ]
min_tmp = min(tmp)

print(f"{L = },\n{tmp = },\n{min_tmp = }, le minimum vaut {min_tmp[0] = } et sa position est
 ↪  {min_tmp[1] = }")
```

```
L = [10, -22, 5, 15, -2, 17, -22],
tmp = [(10, 0), (-22, 1), (5, 2), (15, 3), (-2, 4), (17, 5), (-22, 6)],
min_tmp = (-22, 1), le minimum vaut min_tmp[0] = -22 et sa position est min_tmp[1] = 1
```

Variante : il est possible de spécifier une clé de comparaison pour utiliser une autre valeur. La fonction lambda `key` est utilisée pour indiquer à `min()` de comparer les éléments de la liste en fonction de leur valeur et non de leur index.

```
L = [10, -22, 5, 15, -2, 17, -22]

min_tmp = min(enumerate(L), key=lambda x: x[1])
print(f"{L = },\n{min_tmp = }, le minimum vaut {min_tmp[0] = }")
```

```
L = [10, -22, 5, 15, -2, 17, -22],
min_tmp = (1, -22), le minimum vaut min_tmp[0] = 1
```

Exercice 5.10 (Chaturanga)

Selon la légende, l'ancêtre des échecs, le Chaturanga, aurait été créé en Inde par le sage Sissa. Le roi de l'époque fut tellement séduit par ce nouveau jeu qu'il offrit au sage de choisir tout ce qu'il désirait en récompense. Cependant, Sissa ne demanda rien et resta silencieux. Le roi, offensé par son attitude, le pressa de s'exprimer, mais le sage, blessé, décida de se venger. Il demanda au roi de déposer un grain de riz sur la première case de l'échiquier, deux grains sur la deuxième case, quatre grains sur la troisième case et ainsi de suite jusqu'à la dernière case. Le roi accepta, mais le lendemain matin, il fut alerté par son intendant que la quantité de riz requise était beaucoup trop grande pour être satisfaite. Pourquoi une telle affirmation ?

1. Générer par compréhension la liste du nombre de grains sur chacune des cases.
2. Déterminer le nombre total de grains sur l'échiquier.
3. Chercher la masse d'un grain de riz et déterminer la masse totale de riz sur l'échiquier.

Correction

```
L = [2**i for i in range(64)]

# Quelque prints
for idx in [0,1,2,10,63]:
	print(f"Grains de riz sur la case {idx+1} = {L[idx]}")

nb_grain = sum(L)
print(f"Grains de riz sur l'échiquier = {nb_grain} i.e. {nb_grain:g} ")

un_grain = 0.02 # en gramme
print(f"Un grain de Riz a une masse de {un_grain}g")

tot_gr = un_grain*nb_grain
tot_kg = tot_gr*10**(-3)
tot_Tonne = tot_gr*10**(-6)
print(f"Masse sur l'échiquier {tot_gr:g}g = {tot_kg:g}Kg = {tot_Tonne:g}t ")
print(f"La masse de la Terre est d'environ {5.972e21}t \n")
```

```
Grains de riz sur la case 1 = 1
Grains de riz sur la case 2 = 2
Grains de riz sur la case 3 = 4
Grains de riz sur la case 11 = 1024
Grains de riz sur la case 64 = 9223372036854775808
Grains de riz sur l'échiquier = 18446744073709551615 i.e. 1.84467e+19
Un grain de Riz a une masse de 0.02g
Masse sur l'échiquier 3.68935e+17g = 3.68935e+14Kg = 3.68935e+11t
La masse de la Terre est d'environ 5.972e+21t
```

Nous pouvons calculer directement le nombre de grains sur l'échiquier. En effet, notons b_n le nombre de grains de blé sur la case n, n allant de 0 à 63. La suite (g_n) est géométrique de raison 2 car $g_{n+1} = 2g_n$ donc $g_n = 2^n g_0 = 2^n$. Ainsi la somme totale des grains de blé sera

$$\sum_{n=0}^{63} g_n = \sum_{n=0}^{63} 2^n = \frac{1-2^{64}}{1-2} = 2^{64} - 1 = 18446744073709551615 \simeq 18 \cdot 10^{18}.$$

Au lieu de comparer la masse de riz avec la masse de la Terre, essayons d'avoir une idée du volume occupé par cette quantité. Un grain de riz est, grosso modo, un cylindre de diamètre 1 mm et de hauteur 5 mm. Ainsi on pourrait faire tenir 200 grains de riz dans un centimètre cube (= 1000 mm^3). On peut commencer nos calculs. Si l'on peut faire tenir 200 grains de riz dans un centimètre cube, alors il nous en faut $200 \cdot 100^3$ pour un mètre cube et $200 \cdot 100^3 \cdot 1000^3$ pour un kilomètre cube. Si on divise cette quantité de grains par $2 \cdot 10^{17}$ on obtient le volume de total de notre montagne rizière : 92 kilomètres cubes. Ce volume est tout aussi difficile à imaginer. En remarquant que la France a une superficie d'environ 375000 km^2, on peut se représenter la quantité de riz demandée par l'inventeur du jeu d'échecs de la manière suivante : avec elle on pourrait couvrir toute la France d'une couche de riz de 13 centimètres de haut (car 13 centimètres correspondent à peu près à 7.5 millièmes de kilomètre et $675000/7500 = 90$).

Exercice 5.11 (Filtrer une liste)

Soit la liste

["maths", "info", "python", "exposants", "alpha", "fonctions", "parabole", "equilateral", "orthogonal", "cercles", "isocèle"]

1. Créer et afficher une liste qui ne contient que les mots commençant par une voyelle.
2. Créer et afficher une liste qui ne contient que les mots qui se terminent par un `"s"`.

Correction

`mot[0]` récupère la première lettre de `mot`. Pour vérifier que c'est une voyelle, on vérifie qu'elle appartient à `"aàeéèiouùy"`. `mot[-1]` récupère la dernière lettre de `mot`.

```
>>> ma_liste_de_mots = ["maths", "info", "python", "exposants", "alpha", "fonctions",
→  "parabole", "equilateral", "orthogonal", "cercles", "isocèle" ]
>>> [ mot for mot in ma_liste_de_mots if mot[0] in "aàeéèiouùy"]
['info', 'exposants', 'alpha', 'equilateral', 'orthogonal', 'isocèle']
>>> [ mot for mot in ma_liste_de_mots if mot[-1]=="s"]
['maths', 'exposants', 'fonctions', 'cercles']
```

Exercice 5.12 (Liste des diviseurs)

Pour un entier $n \in \mathbb{N}$ donné, calculer la liste de ses **diviseurs propres** (diviseurs de n strictement inférieurs à n). On rappelle que d divise n si et seulement si `n%d==0`.

Correction

```
>>> n = 100
>>> [d for d in range(1,n) if (n%d==0)]
[1, 2, 4, 5, 10, 20, 25, 50]
```

Mieux (on ne teste que la moitié des cas) :

```
>>> n = 100
>>> [d for d in range(1,int(n/2+1)) if (n%d==0)]
[1, 2, 4, 5, 10, 20, 25, 50]
```

Exercice 5.13 (Nombres parfaits)

Pour un entier $n \in \mathbb{N}$ donné, on note $d(n)$ la somme des **diviseurs propres** de n.

- Si $d(n) = n$ on dit que n est parfait,
- si $d(n) < n$ on dit que n est déficient,
- si $d(n) > n$ on dit que n est abondant.

Par exemple,

$$\left.\begin{aligned} a &= 220 \text{ est divisible par } 1,2,4,5,10,11,20,22,44,55,110 \\ d(a) &= 1+2+4+5+10+11+20+22+44+55+110 = 284 > a \end{aligned}\right] \implies a \text{ est abondant}$$

$$\left.\begin{aligned} b &= 284 \text{ est divisible par } 1,2,4,71,142 \\ d(b) &= 1+2+4+71+142 = 220 < b \end{aligned}\right] \implies b \text{ est déficient}$$

$$\left.\begin{aligned} c &= 28 \text{ est divisible par } 1,2,4,7,14 \\ d(c) &= 1+2+4+7+14 = 28 = c \end{aligned}\right] \implies c \text{ est parfait}$$

Classer tous les nombres $n \le 100$ en trois listes.

Correction

```
A, D, P = [], [], []
for n in range(1,100+1):
	diviseurs = [x for x in range(1,int(n/2)+1) if (n%x==0)]
	s=sum(diviseurs)
	# print(f"n={n:3d}, s={s:3d}, diviseurs={diviseurs}") # Phase de debug
	if s==n :
		P.append(n)
	elif s<n:
		D.append(n)
	else :
		A.append(n)

print("Abondants\n",A)
print("Defectueux\n",D)
print("Parfaits\n",P)
```

Exercice Bonus 5.14 (Défi Turing n°18 – Somme de nombres non abondants)

Un nombre parfait est un nombre dont la somme de ses diviseurs propres est exactement égal au nombre. Par exemple, la somme des diviseurs propres de 28 serait $1+2+4+7+14 = 28$, ce qui signifie que 28 est un nombre parfait. Un nombre n est appelé déficient si la somme de ses diviseurs propres est inférieur à n et on l'appelle abondant si cette somme est supérieure à n. Comme 12 est le plus petit nombre abondant ($1+2+3+4+6 = 16$), le plus petit nombre qui peut être écrit comme la somme de deux nombres abondants est 24.

Trouver la somme de tous les entiers positifs inférieurs ou égaux à 2013 qui **ne peuvent pas** être écrits comme la somme de deux nombres abondants.

Correction

```
# (dictionnaire) nb : liste diviseurs
div = { x : [ i for i in range(1,x) if x%i==0 ]  for x in range(2,2014) }
# (dictionnaire) nb : somme ses diviseurs propres
s = { k : sum(v)  for k,v in div.items() }
# (liste) nb abondant
a = [ k for k,v in s.items() if v>k ]
```

```
# (ensemble) nb somme de 2 nb abondnats (sans doublons)
L = set([a1+a2 for a1 in a for a2 in a])
print(sum([ x for x in range(2014) if x not in L]))
```

```
577167
```

Exercice Bonus 5.15 (Défis Turing n°71 – ensembles)
On remarque que $567^2 = 321489$ utilise tous les chiffres de 1 à 9, une fois chacun (si on excepte le carré). Quel est le seul autre nombre qui, élevé au carré, présente la même propriété ?

Correction

```
>>> [i  for i in range(100,1000) if len( set(str(i)+str(i*i)).difference(set("0")) )==9]
[567, 854]
```

Autre version sans utiliser des ensembles :

```
>>> [n for n in range(100,1000) if (sorted(str(n)+str(n**2))==list('123456789'))]
[567, 854]
```

Exercice 5.16 (Liste de nombres)
Écrire une liste qui contient tous les entiers compris entre 0 et 999 qui vérifient toutes les propriétés suivantes : l'entier se termine par 3, la somme des chiffres est supérieure à 15, le chiffre des dizaines est pair.

Correction
Première méthode : on considère tous les entiers compris entre 0 et 999 et on vérifie s'ils satisfont les propriétés.

```
print([n for n in range(1,1000) if n%10==3 and  sum([int(c) for c in str(n)])>15 and
 ↪  int(str(n)[-2])%2==0 ])
```

```
[583, 683, 763, 783, 863, 883, 943, 963, 983]
```

Notons que pour savoir si l'entier se termine par 3, on peut écrire `int(str(n)[-1])==3` ou `str(n)[-1]=='3'` ou `n%10==3`.

Deuxième méthode : on construit les nombres $n = c \times 10^2 + d \times 10 + u$:

$$n = \boxed{c \mid d \mid u}$$

L'entier se termine par 3 donc $u = 3$, le chiffre des dizaines est pair donc $d = 0, 2, 4, 6, 8 =$`range(0,10,2)`, la somme des chiffres est supérieure ou égale à 15 ainsi $c > 12 - d$:

```
L = [c*10**2+d*10+3 for d in range(0,10,2) for c in range(12-d+1,10)]
L.sort()
print(L)
```

```
[583, 683, 763, 783, 863, 883, 943, 963, 983]
```

qui s'écrit aussi comme

```
L = []
for d in range(0,10,2):
	for c in range(12-d+1,10):
		L.append(c*10**2+d*10+3)
L.sort()
print(L)
```

```
[583, 683, 763, 783, 863, 883, 943, 963, 983]
```

Exercice Bonus 5.17 (Pydéfi – SW III : L'ordre 66 ne vaut pas 66...)

Lorsque Palpatine prend le pouvoir, il entame la destruction des Jedis en envoyant un ordre à tous les clones : l'ordre 66. Les clones se retournent alors contre les Jedis. C'est du moins ce que prétend la légende cinématographique.

La réalité est un peu différente et s'il y a bien un ordre particulier à donner aux clones, ce n'est pas l'ordre 66. Pour le rendre difficile à découvrir, les Kaminoans ont utilisé les talents de calcul mental de Jango Fett en apprenant aux clones une liste de propriétés. Si le numéro satisfait ces propriétés, alors les clones se retourneront contre les Jedis.

Voici la liste des propriétés enseignées aux clones :

- l'ordre est un nombre à 4 chiffres, tous impairs
- chaque chiffre du nombre est strictement plus petit que le suivant (par exemple 3579 convient, mais pas 3557 ou 3157)
- le produit des chiffres du nombre est un nouveau nombre qui ne contient que des chiffres impairs
- la somme des chiffres du nombre est un nouveau nombre qui ne contient que des chiffres pairs

Trouvez le seul numéro d'ordre qui convient et provoque l'attaque des clones contre les Jedis.

Source : https://pydefis.callicode.fr/defis/Ordre66/txt

Exercice 5.18 (Soustraire deux listes)

Soit deux listes de nombres $A = [a_i]_{i=0}^n$ et $B = [b_i]_{i=0}^n$ de même cardinalité. Construire la liste $D = [d_i]_{i=0}^n$ telle que $d_i = a_i - b_i$.

Correction

```
>>> A = [1,4,6,19,125]
>>> B = [78,2,46,19,56]
>>> D = [ A[i]-B[i] for i in range(len(A)) ]
>>> D
[-77, 2, -40, 0, 69]
```

Il existe une fonction spécifique qui parcourt deux (ou plus) listes de même longueur et génère un tuple : la fonction `zip(liste0,liste1,...)` :

```
>>> A = [1,4,6,19,125]
>>> B = [78,2,46,19,56]
>>> D = [ a_i - b_i for a_i, b_i in zip(A, B) ]
>>> D
[-77, 2, -40, 0, 69]
```

Exercice 5.19 (Jours du mois)

Soient les listes suivantes :

```
J = [31,28,31,30,31,30,31,31,30,31,30,31]
M = ['Janvier','Fevrier','Mars','Avril','Mai','Juin',
   'Juillet','Aout','Septembre','Octobre','Novembre','Decembre']
```

Générer en compréhension la **liste de tuples** suivante :

```
[('Janvier',31),('Fevrier',28)...
```

Correction

Le deux écritures suivantes construisent la même liste :

```
[ x for x in L ]
[ x[i] for i in range(len(L)) ]
```

En adaptant la deuxième approche on peut parcourir deux listes en même temps :

```
J = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
M = ['Janvier', 'Fevrier', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Aout', 'Septembre',
↪ 'Octobre', 'Novembre', 'Decembre']
F = [ (M[i],J[i]) for i in range(len(J)) ]
print(F)
```

[('Janvier', 31), ('Fevrier', 28), ('Mars', 31), ('Avril', 30), ('Mai', 31), ('Juin', 30), ('Juillet', 31), ('Aout', 31), ('Septembre', 30), ('Octobre', 31), ('Novembre', 30), ('Decembre', 31)]

☆ Bonus : il existe une fonction spécifique qui parcourt deux (ou plus) listes de même longueur et génère un tuple : la fonction `zip(liste0,liste1,...)` :

```
J = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
M = ['Janvier', 'Fevrier', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Aout', 'Septembre',
↪ 'Octobre', 'Novembre', 'Decembre']
F = list(zip(M,J))
print(F)
```

[('Janvier', 31), ('Fevrier', 28), ('Mars', 31), ('Avril', 30), ('Mai', 31), ('Juin', 30), ('Juillet', 31), ('Aout', 31), ('Septembre', 30), ('Octobre', 31), ('Novembre', 30), ('Decembre', 31)]

Un dictionnaire est peut-être une structure un peu plus adaptée qu'une liste de tuples :

```
J=[31,28,31,30,31,30,31,31,30,31,30,31]
M = ['Janvier','Fevrier','Mars','Avril','Mai','Juin','Juillet',
   'Aout','Septembre','Octobre','Novembre','Decembre']
F={ M[i] : J[i]  for i in range(len(J)) }
print(F)
```

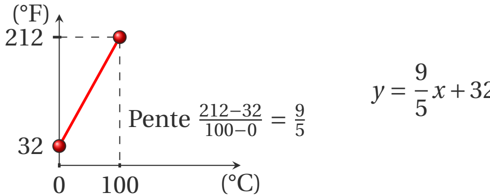
'Janvier' : 31, 'Fevrier' : 28, 'Mars' : 31, 'Avril' : 30, 'Mai' : 31, 'Juin' : 30, 'Juillet' : 31, 'Aout' : 31, 'Septembre' : 30, 'Octobre' : 31, 'Novembre' : 30, 'Decembre' : 31

Exercice 5.20 (Conversion de températures)

Conversion des dégrées Celsius en dégrées Fahrenheit : une température de 0°C correspond à 32°F tandis que 100°C correspondent à 212°F. Sachant que la formule permettant la conversion d'une valeur numérique x de la température en (°C) vers l'unité (°F) est affine, définir une liste de tuples dont la première composante contient la valeur en Celsius (on considère des températures allant de 0°C à 100°C par paliers de 10°C) et la deuxième l'équivalente en Fahrenheit.

Correction

La formule permettant la conversion d'une valeur numérique x de la température en (°C) vers l'unité (°F) est affine :



$$y = \frac{9}{5}x + 32$$

ainsi

```
print([ (x, 9*x/5+32) for x in range(0,101,10) ])
```

[(0, 32.0), (10, 50.0), (20, 68.0), (30, 86.0), (40, 104.0), (50, 122.0), (60, 140.0), (70, 158.0), (80, 176.0), (90, 194.0), (100, 212.0)]

Exercice 5.21 (Années bissextiles)

Depuis l'ajustement du calendrier grégorien, l'année sera bissextile si l'année est

$$\Big(\text{divisible par 4} \quad \text{et} \quad \text{non divisible par 100}\Big) \qquad \text{ou} \qquad \Big(\text{divisible par 400.}\Big)$$

Ainsi,

- 2019 n'est pas bissextile car non divisible par 4 ni par 400
- 2008 était bissextile suivant la première règle (divisible par 4 et non divisible par 100)

- 1900 n'était pas bissextile car divisible par 4, mais aussi par 100 (première règle non respectée) et non divisible par 400 (seconde règle non respectée).
- 2000 était bissextile car divisible par 400.

Écrire la liste des années bissextiles entre l'année 1800 et l'année 2099 et la liste des années non bissextiles entre l'année 1800 et l'année 2000.

Correction
Ce programme traduit la définition des années bissextiles : "b est bissextile si et seulement si ($r = 0$ et $s \neq 0$) ou ($t = 0$)" où r, s et t désignent successivement les restes dans la division euclidienne de b par 4, 100 et 400. Attention aux parenthèses : "($r = 0$ et $s \neq 0$) ou ($t = 0$)" équivaut à "($r = 0$ ou $t = 0$) et ($r = 0$ ou $s \neq 0$)".

```
print([b for b in range(1800,2100) if (b%4==0 and b%100!=0) or (b%400==0)])
```

[1804, 1808, 1812, 1816, 1820, 1824, 1828, 1832, 1836, 1840, 1844, 1848, 1852, 1856, 1860, 1864, 1868, 1872, 1876, 1880, 1884, 1888, 1892, 1896, 1904, 1908, 1912, 1916, 1920, 1924, 1928, 1932, 1936, 1940, 1944, 1948, 1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016, 2020, 2024, 2028, 2032, 2036, 2040, 2044, 2048, 2052, 2056, 2060, 2064, 2068, 2072, 2076, 2080, 2084, 2088, 2092, 2096]

La négation s'écrit : "n n'est pas bissextile si et seulement si ($r \neq 0$ ou $s = 0$) et ($t \neq 0$)"

```
print([n for n in range(1800,2000) if (n%4!=0 or n%100==0) and (n%400!=0)])
```

[1800, 1801, 1802, 1803, 1805, 1806, 1807, 1809, 1810, 1811, 1813, 1814, 1815, 1817, 1818, 1819, 1821, 1822, 1823, 1825, 1826, 1827, 1829, 1830, 1831, 1833, 1834, 1835, 1837, 1838, 1839, 1841, 1842, 1843, 1845, 1846, 1847, 1849, 1850, 1851, 1853, 1854, 1855, 1857, 1858, 1859, 1861, 1862, 1863, 1865, 1866, 1867, 1869, 1870, 1871, 1873, 1874, 1875, 1877, 1878, 1879, 1881, 1882, 1883, 1885, 1886, 1887, 1889, 1890, 1891, 1893, 1894, 1895, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1905, 1906, 1907, 1909, 1910, 1911, 1913, 1914, 1915, 1917, 1918, 1919, 1921, 1922, 1923, 1925, 1926, 1927, 1929, 1930, 1931, 1933, 1934, 1935, 1937, 1938, 1939, 1941, 1942, 1943, 1945, 1946, 1947, 1949, 1950, 1951, 1953, 1954, 1955, 1957, 1958, 1959, 1961, 1962, 1963, 1965, 1966, 1967, 1969, 1970, 1971, 1973, 1974, 1975, 1977, 1978, 1979, 1981, 1982, 1983, 1985, 1986, 1987, 1989, 1990, 1991, 1993, 1994, 1995, 1997, 1998, 1999]

Si on a un doute pour la négation, on peut tout simplement écrire `if !((b%4==0 and b%100!=0) or (b%400==0))`.

Exercice 5.22

On effectue une enquête auprès de N français, afin de savoir où ils ont passé leurs dernières vacances et la durée de leur séjour (on ne considère qu'un seul séjour par personne). Pour chaque personne ayant répondu à l'enquête, on enregistre la réponse dans un tuple qui contient deux valeurs entières : la première est la durée du séjour (exprimée en jour), la seconde est le code du pays selon les conventions suivantes :

- personne non partie en vacances : code 0,
- personne partie en France : code 1,
- personne partie à l'étranger : code > 1 (exemple : Italie code 2, Espagne code 3, ...)

Ces tuples sont regroupés dans une liste.

Écrire un script qui crée les deux sous-liste "français partis en vacances en France", "français partis en vacances à l'étranger". Puis calculer et afficher :

1. le nombre de français ayant répondu à l'enquête,
2. le nombre de français partis en vacances en France et la durée moyenne de leur séjour,
3. le nombre de français partis en vacances à l'étranger et la durée moyenne de leur séjour.

Pour valider le script, créer un jeu de données bien choisi.

Correction

```
L = [ (0,0) , (7,1) , (7,2) , (28,2) , (3,1) ]

# Sous-listes
no_vac = [ (j,p) for j,p in L if p==0]
oui_vac_fr = [ (j,p) for j,p in L if p==1]
oui_vac_et = [ (j,p) for j,p in L if p>1]
```

```
# Affichage
print(f"Nombre de français ayant répondu à l'enquête: {len(L)}")
print(f"Nombre de français partis en vacances en France: {len(oui_vac_fr)}. Durée moyenne de
↪ leur séjour: { sum([j for j,p in oui_vac_fr])/len(oui_vac_fr) }")
print(f"Nombre de français partis en vacances à l'étranger: {len(oui_vac_et)}. Durée moyenne
↪ de leur séjour: { sum([j for j,p in oui_vac_et])/len(oui_vac_et) }")
```

```
Nombre de français ayant répondu à l'enquête: 5
Nombre de français partis en vacances en France: 2. Durée moyenne de leur séjour: 5.0
Nombre de français partis en vacances à l'étranger: 2. Durée moyenne de leur séjour: 17.5
```

Exercice 5.23 (Produit matriciel)

En calcul matriciel, si $\mathbb{A}$ est une matrice de n lignes et p colonnes et $\mathbb{B}$ une matrice de p lignes et q colonnes, la matrice $\mathbb{C} = \mathbb{A}\mathbb{B}$ est une matrice de n lignes et q colonnes dont les éléments sont définis par

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

Écrire une boucle qui, pour $\mathbb{A}$ et $\mathbb{B}$ données, calcule $\mathbb{C}$.

Testez votre code sur les exemples suivants :

$$\overbrace{\begin{pmatrix} 3 & 1 & 5 \\ 2 & 7 & 0 \end{pmatrix}}^{2\times 3} \overbrace{\begin{pmatrix} 2 & 1 & -1 & 0 \\ 3 & 0 & 1 & 8 \\ 0 & -5 & 3 & 4 \end{pmatrix}}^{3\times 4} = \overbrace{\begin{pmatrix} 9 & -22 & 13 & 28 \\ 25 & 2 & 5 & 56 \end{pmatrix}}^{2\times 4}$$

$$\overbrace{\begin{pmatrix} -3 & 0 & 5 \end{pmatrix}}^{1\times 3} \overbrace{\begin{pmatrix} 2 \\ -4 \\ -3 \end{pmatrix}}^{3\times 1} = -21$$

$$\overbrace{\begin{pmatrix} -3 \\ 0 \\ 5 \end{pmatrix}}^{3\times 1} \overbrace{\begin{pmatrix} 2 & -4 & -3 \end{pmatrix}}^{1\times 3} = \overbrace{\begin{pmatrix} -6 & 12 & 9 \\ 0 & 0 & 0 \\ 10 & -20 & -15 \end{pmatrix}}^{3\times 3}$$

Correction

```
TESTS = [ (  [ [3,1,5] , [2,7,0] ] , [ [2,1,-1,0] , [3,0,1,8] , [0,-5,3,4] ] ),
        (  [ [-3,0,5 ] ]  ,  [ [2], [-4], [-3] ]  ),
        (  [ [-3],[0],[5] ] , [ [2,-4,-3] ] ) ]

for A,B in TESTS:
    print(f"A = {A}")
    print(f"B = {B}")
    n,p1 = len(A), len(A[0])
    p2,q = len(B), len(B[0])
    print(f"A est un matrice {n}x{p1}")
    print(f"B est un matrice {p2}x{q}")
    if p1==p2:
        print(f"C est un matrice {n}x{q}")
        C = [[ sum(A[i][k]*B[k][j] for k in range(p1)) for j in range(q)] for i in range(n)]
    print(f"C = {C}\n")
```

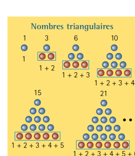
```
A = [[3, 1, 5], [2, 7, 0]]
B = [[2, 1, -1, 0], [3, 0, 1, 8], [0, -5, 3, 4]]
A est un matrice 2x3
B est un matrice 3x4
```

```
C est un matrice 2x4
C = [[9, -22, 13, 28], [25, 2, 5, 56]]

A = [[-3, 0, 5]]
B = [[2], [-4], [-3]]
A est un matrice 1x3
B est un matrice 3x1
C est un matrice 1x1
C = [[-21]]

A = [[-3], [0], [5]]
B = [[2, -4, -3]]
A est un matrice 3x1
B est un matrice 1x3
C est un matrice 3x3
C = [[-6, 12, 9], [0, 0, 0], [10, -20, -15]]
```

Exercice 5.24 (Nombres triangulaires)

La suite des nombres triangulaires est générée en additionnant les nombres naturels. Ainsi, le 7-ème nombre triangulaire est 1+2+3+4+5+6+7 = 28. Les dix premiers nombres triangulaires sont les suivants : [1, 3, 6, 10, 15, 21, 28, 36, 45, 55]

Écrire la suite des premiers 100 nombres triangulaires.



Correction

- Version naïve : $L_n = \sum_{i=1}^{n} i = 1+2+3+\cdots+n =$ `sum(range(1,n+1))`

```
L = [ sum(range(1,n+1)) for n in range(1,101)]
print(L)
```

- Mieux : $\begin{cases} L_1 = 1 \\ L_n = L_{n-1} + n \end{cases}$

```
L = [1]
for n in range(2,101):
    L.append(L[-1]+n)
print(L)
```

Vu que la division par 2 se fait uniquement sur des nombres pairs (car soit n est pair soit $n+1$ est pair), la suite est une suite d'entiers. Pour garder cette propriété il faut utiliser la division entière.

- Encore mieux : $L_n = \sum_{i=1}^{n} i = 1+2+3+\cdots+n = \dfrac{n(n+1)}{2}$

```
mylist = [n*(n+1)//2 for n in range(1,101)]
print(mylist)
```

[1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351, 378, 406, 435, 465, 496, 528, 561, 595, 630, 666, 703, 741, 780, 820, 861, 903, 946, 990, 1035, 1081, 1128, 1176, 1225, 1275, 1326, 1378, 1431, 1485, 1540, 1596, 1653, 1711, 1770, 1830, 1891, 1953, 2016, 2080, 2145, 2211, 2278, 2346, 2415, 2485, 2556, 2628, 2701, 2775, 2850, 2926, 3003, 3081, 3160, 3240, 3321, 3403, 3486, 3570, 3655, 3741, 3828, 3916, 4005, 4095, 4186, 4278, 4371, 4465, 4560, 4656, 4753, 4851, 4950, 5050]

Exercice Bonus 5.25 (Nombres pentagonaux)

Un nombre pentagonal est un nombre qui peut être représenté par un pentagone https://fr.wikipedia.org/wiki/Nombre_pentagonal).

Pour tout entier $n \geq 1$, considérons la suite arithmétique $(P_n)_{n\in\mathbb{N}^*}$ de premier terme 1 et de raison 3 :

$$\begin{cases} P_1 = 1, \\ P_n = P_{n-1} + 3. \end{cases}$$

Le n-ième nombre pentagonal est la somme des n premiers termes de cette suite :

$$L_n = \sum_{i=1}^{n} P_i = 1+4+7+\cdots+(3n-2).$$

Les dix premiers nombres pentagonaux sont les suivants : $[1,5,12,22,35,51,70,92,117,145]$.

Écrire la suite des premiers 100 nombres pentagonaux.

Correction

```
P = [1]
for n in range(2,101):
    P.append(P[-1]+3)

L = [sum(P[:n]) for n in range(1,len(P))]
print(L)
```

On peut expliciter la suite définie par récurrence en écrivant P_n directement en fonction n :

$$P_n = 3n-2$$

ainsi

$$L_n = \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} 3i-2 = 3\sum_{i=1}^{n} i - 2n = 3\frac{n(n+1)}{2} - 2n = \frac{n(3n-1)}{2}.$$

```
mylist = [ n*(3*n-1)//2 for n in range(1,101)]
print(mylist)
```

[1, 5, 12, 22, 35, 51, 70, 92, 117, 145, 176, 210, 247, 287, 330, 376, 425, 477, 532, 590, 651, 715, 782, 852, 925, 1001, 1080, 1162, 1247, 1335, 1426, 1520, 1617, 1717, 1820, 1926, 2035, 2147, 2262, 2380, 2501, 2625, 2752, 2882, 3015, 3151, 3290, 3432, 3577, 3725, 3876, 4030, 4187, 4347, 4510, 4676, 4845, 5017, 5192, 5370, 5551, 5735, 5922, 6112, 6305, 6501, 6700, 6902, 7107, 7315, 7526, 7740, 7957, 8177, 8400, 8626, 8855, 9087, 9322, 9560, 9801, 10045, 10292, 10542, 10795, 11051, 11310, 11572, 11837, 12105, 12376, 12650, 12927, 13207, 13490, 13776, 14065, 14357, 14652, 14950]

Exercice 5.26 (Table de multiplication)

Afficher la table de multiplication par $1,\dots,10$ suivante (l'élément en position (i,j) est égal au produit $i\,j$ lorsque les indices commencent à 1) :

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Correction

```
n = 10
M = []
for i in range(1,n):
	M.append([(j+1)*(i+1) for j in range(n)])
print(M)
```

soit encore

```
n = 10
M = [ [(j+1)*(i+1) for j in range(n)] for i in range(n) ]
print(M)
```

[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [2, 4, 6, 8, 10, 12, 14, 16, 18, 20], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30], [4, 8, 12, 16, 20, 24, 28, 32, 36, 40], [5, 10, 15, 20, 25, 30, 35, 40, 45, 50], [6, 12, 18, 24, 30, 36, 42, 48, 54, 60], [7, 14, 21, 28, 35, 42, 49, 56, 63, 70], [8, 16, 24, 32, 40, 48, 56, 64, 72, 80], [9, 18, 27, 36, 45, 54, 63, 72, 81, 90], [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]]

Affichage amélioré :

```
for row in M:
    for num in row:
        print(f"{num:3d}", end=" ")
    print("")
```

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

Exercice 5.27 (Défi Turing n°1 – somme de multiples)

Si on liste tous les entiers naturels inférieurs à 20 qui sont multiples de 5 ou de 7, on obtient 5, 7, 10, 14 et 15. La somme de ces nombres est 51.

Trouver la somme de tous les multiples de 5 ou de 7 inférieurs à 2013.

Correction

Reformulons le problème : on cherche à calculer la somme de tous les nombres inférieurs à 2013 divisibles par 5 ou 7. Un programme brute force (programme qui teste toutes les valeurs possibles) fonctionne très bien. Pour savoir si un nombre a est divisible par un nombre b, il faut que le reste de la division euclidienne de ces deux nombres soit égal à 0. En Python, il faut utiliser le signe % pour obtenir le reste d'une division.

```
somma = 0
for i in range(1,2014):
    if (i%5)==0 or (i%7)==0:
        somma += i
print(somma)
```

ou, en générant d'abord la liste,

```
somma = sum([i for i in range(1,2014) if (i%5)==0 or (i%7)==0])
print(somma)
```

Dans tous les cas le résultat est

```
636456
```

Le problème peut être entièrement résolu en utilisant des mathématiques. Trouver la somme de tous les nombres inférieurs à 2013 divisibles par 5 ou 7, revient à faire la somme de tous les multiples de 5 inférieurs à 2013, d'y ajouter la somme de tous les multiples de 7 inférieurs à 2013 et d'y soustraire tous les multiples de 35 inférieurs à 2013 (car les deux nombres sont premiers).

```
sept = [i for i in range(0,2014,7)]
cinq = [i for i in range(0,2014,5)]
doublons = [i for i in range(0,2014,5*7)]
somma = sum(sept)+sum(cinq)-sum(doublons)
print(somma)
```

On obtient

$$\sum_{k=1}^{\lfloor\frac{2013-1}{5}\rfloor} 5k+\sum_{k=1}^{\lfloor\frac{2013-1}{7}\rfloor} 7k-\sum_{k=1}^{\lfloor\frac{2013-1}{35}\rfloor} 35k = 5\sum_{k=1}^{402} k+7\sum_{k=1}^{287} k-35\sum_{k=1}^{57} k$$
$$= 5\frac{402\times 403}{2}+7\frac{287\times 288}{2}-35\frac{57\times 58}{2}$$
$$= 405015+289296-57855 = 636456$$

Exercice 5.28 (Nombres de Armstrong)

On dénomme nombre de Armstrong un entier naturel qui est égal à la somme des cubes des chiffres qui le composent. Par exemple 153 est un nombre de Armstrong puisque $153 = 1^3+5^3+3^3$. On peut montrer qu'il n'existe que 4 nombres de Armstrong et qu'ils ont tous 3 chiffres. Écrire la liste des 4 nombres de Armstrong.

Correction

Pour résoudre ce problème, on converti chaque nombre de 3 chiffres en chaîne de caractères, on lis les chiffres un par un, on les convertit en entier, enfin on additionne leur cube et on vérifie si on trouve encore le nombre initial :

```
Armstrong = []
for n in range(100,1000):
    S = sum([int(i)**3 for i in str(n)])
    if S==n:
        Armstrong.append(n)
print(Armstrong)
```

```
[153, 370, 371, 407]
```

ou, de façon plus compacte,

```
Armstrong = [n for n in range(100,1000) if n==sum([int(i)**3 for i in str(n)])]
print(Armstrong)
```

```
[153, 370, 371, 407]
```

Exercice 5.29 (Nombre de chiffres)

Un imprimeur a imprimé un livre de 1234 pages. Combien de fois il a utilisé le caractère '4'? Autrement-dit, combien de fois le chiffre '4' apparaît en écrivant les nombres de 1 à 1234 inclus? Répondre en une ligne.

Correction

```
print(sum( [str(i).count('4') for i in range(1,1235) ] ))
```

```
344
```

Exercice 5.30 (Défi Turing n°85 – Nombres composés de chiffres différents)

Il y a 32490 nombres composés de chiffres tous différents entre 1 et 100000, par exemple 4, 72, 1468, 53920, etc. Quelle est la somme de ces nombres?

Correction

```
print(sum([n for n in range(1,100001) if len(str(n))==len(set(str(n)))]))
```

```
1520464455
```

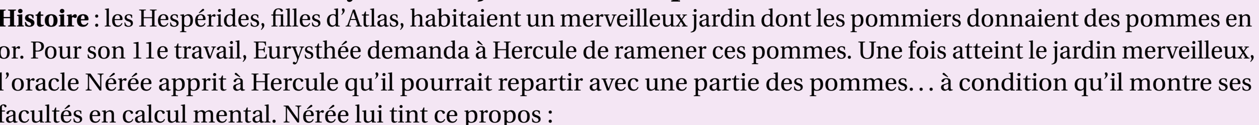
Exercice Bonus 5.31 (Pydéfi – Piège numérique à Pokémons)

Ossatueur et Mewtwo sont passionnés par les nombres. On le sait peu. Le premier apprécie tout particulièrement les multiples de 7 : 7, 14, 21... Le second adore les nombres dont la somme des chiffres vaut exactement 11 : 29, 38, 47...

Pour les attirer, vous chantonnez les nombres qu'ils préfèrent. Quels sont les nombres entiers positifs inférieurs à 1000 qui plaisent à la fois à Ossatueur et Mewtwo?

Source : https://pydefis.callicode.fr/defis/PokeNombresCommuns/txt

Exercice Bonus 5.32 (Pydéfi – Le jardin des Hespérides)



Histoire : les Hespérides, filles d'Atlas, habitaient un merveilleux jardin dont les pommiers donnaient des pommes en or. Pour son 11e travail, Eurysthée demanda à Hercule de ramener ces pommes. Une fois atteint le jardin merveilleux, l'oracle Nérée apprit à Hercule qu'il pourrait repartir avec une partie des pommes... à condition qu'il montre ses facultés en calcul mental. Nérée lui tint ce propos :

> J'ai empilé les pommes d'or pour toi, sous la forme d'une pyramide. L'étage le plus haut ne contient qu'une pomme. L'étage juste en dessous forme un carré 2×2 (contenant 4 pommes), l'étage juste en dessous forme un carré 3×3 (contenant 9 pommes). La pyramide que tu vois contient 50 étages. L'étage de base contient donc 2500 pommes... Je suis d'accord pour te laisser partir avec les pommes contenues dans certains étages. Précisément, si un étage contient un nombre de pommes multiple de 3, tu peux l'emporter. Si tu m'annonces combien de pommes tu emporteras au total, je te laisserai partir avec les pommes...

Défi : vous devez aider Hercule en lui indiquant le nombre de pommes qu'il pourra emporter pour une pyramide de 50 étages.

Testez votre code : par exemple, si la pyramide n'avait compté que 6 étages, chaque étage aurait été composé de : 1, 4, 9, 16, 25 et 36 pommes. Hercule aurait pu emporter les 9 pommes de l'étage 3 (car 9 est un multiple de 3) et les 36 pommes de l'étage 6 (car 36 est un multiple de 3). Au total il aurait donc emporté 45 pommes.

Source : https://pydefis.callicode.fr/defis/Herculito11Pommes/txt

Exercice Bonus 5.33 (Pydéfi – Constante de Champernowne)

La constante de Champernowne est un nombre compris entre 0 et 1, dont le dévelopopement décimal est obtenu en écrivant successivement les nombre entiers. Elle commence ainsi :

$$0,12345678910111213141515171819202122\dots$$

Numérotons les chiffres situés après la virgule. Le chiffre 1 est 1, le chiffre 9 est 9, le chiffre 10 est 1 et le chiffre 11 est 0 etc.

Donner la somme des chiffres de $n_1 = 104$ à $n_2 = 156$ inclus.

Par exemple, si $n_1 = 11$ et $n_2 = 21$, la réponse à donner serait alors $0+1+1+1+2+1+3+1+4+1+5 = 20$.

Source : https://pydefis.callicode.fr/defis/Champernowne/txt

Exercice Bonus 5.34 (Défi Turing n° 40 – La constante de Champernowne)

La constante de Champernowne est un nombre irrationnel créé en concaténant les entiers positifs :

$$0,123456789101112131415151718192021\dots$$

On peut voir que le 12-ème chiffre de la partie fractionnaire est 1.

Si d_n représente le n-ième chiffre de la partie fractionnaire, quelle est la valeur de l'expression suivante ?

$$d_1 \times d_{10} \times d_{100} \times d_{1000} \times d_{10000} \times d_{100000} \times d_{1000000} \times d_{10000000} \times d_{100000000}$$

Correction

```
s = ""
n = 1
while len(s)<10**8+1:
    s += str(n)
    n += 1

prod = 1
for i in range(1,9):
```

```
    prod *= int(s[10**i-1])
print(prod)
```

```
11760
```

Exercice Bonus 5.35 (Pydéfi – Nos deux chiffres préférés)

Calculer la somme des nombres compris entre deux bornes $L = 140$ et $R = 1007$ (incluses) qui contiennent le chiffre 7 ou le chiffre 4 (ou les deux).

Testez votre code : si les bornes sont $L = 10$ et $R = 54$, le total à donner est 652, car la liste des nombres à ajouter est $[14, 17, 24, 27, 34, 37, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 54]$.

Source : https://pydefis.callicode.fr/defis/ChiffresPreferes/txt

Exercice Bonus 5.36 (Pydéfi – Série décimée...)

Considérons la série harmonique

$$S(n) = \sum_{i=1}^{i=n} \frac{1}{i}$$

Soit $T(n)$ la série obtenue en excluant de $S(n)$ toutes les fractions qui contiennent le chiffre 9 au dénominateur (par exemple 1/9 et 1/396). Elle s'appelle "série de Kempner". Que vaut $T(242)$?

Source : https://pydefis.callicode.fr/defis/SuiteDecimation/txt

L'intérêt de cette suite réside dans le fait que contrairement à la série harmonique, elle converge. Ce résultat fut démontré en 1914 par Aubrey J. Kempner : le nombre d'entiers à n chiffres, dont le premier est compris entre 1 et 8 et les $n-1$ suivants entre 0 et 8, est $8 \times 9^{n-1}$, et chacun d'eux est minoré par 10^{n-1}, donc la série est majorée par la série géométrique $8\sum_{n=1}^{\infty}\left(\frac{9}{10}\right)^{n-1} = 80$.

Exercice Bonus 5.37 (Pydéfi – Désamorçage de bombe à distance (II))

Après leur cuisant échec face à Black Widow, les Maîtres du Mal ont placé une nouvelle bombe dévastatrice à Los Angeles. Cette fois-ci, impossible de s'en approcher : Œil de faucon doit la désamorcer à distance, en coupant deux fils avec son arc et ses flèches.

Pour connaître les fils à couper, il y a un certain nombre d'instructions à suivre.

Les fils de la bombe sont numérotés. Supposons pour l'exemple qu'il n'y ait «que» 500 fils numérotés de 1 à 500. Pour connaître les deux fils à couper, on procède par élimination :

- Conserver les fils dont le numéro est multiple de 5 ou de 7. Les fils conservés sont :
  `5, 7, 10, 14, 15, 20, 21, 25, 28, 30, 35, 40, 42, 45, 49, 50, 55, 56, 60, 63, 65, 70, 75, 77, 80, 84, 85, 90, 91, 95, 98, 100, 105, 110, 112, 115, 119, 120, 125, 126, 130, 133, 135, 140, 145, 147, 150, 154, 155, 160, 161, 165, 168, 170, 175, 180, 182, 185, 189, 190, 195, 196, 200, 203, 205, 210, 215, 217, 220, 224, 225, 230, 231, 235, 238, 240, 245, 250, 252, 255, 259, 260, 265, 266, 270, 273, 275, 280, 285, 287, 290, 294, 295, 300, 301, 305, 308, 310, 315, 320, 322, 325, 329, 330, 335, 336, 340, 343, 345, 350, 355, 357, 360, 364, 365, 370, 371, 375, 378, 380, 385, 390, 392, 395, 399, 400, 405, 406, 410, 413, 415, 420, 425, 427, 430, 434, 435, 440, 441, 445, 448, 450, 455, 460, 462, 465, 469, 470, 475, 476, 480, 483, 485, 490, 495, 497, 500`
- Dans ce qui reste, conserver les fils dont le chiffre des dizaines est inférieur ou égal au chiffre des unités. Il reste les fils :
  `5, 7, 14, 15, 25, 28, 35, 45, 49, 55, 56, 77, 100, 105, 112, 115, 119, 125, 126, 133, 135, 145, 147, 155, 168, 189, 200, 203, 205, 215, 217, 224, 225, 235, 238, 245, 255, 259, 266, 300, 301, 305, 308, 315, 322, 325, 329, 335, 336, 345, 355, 357, 378, 399, 400, 405, 406, 413, 415, 425, 427, 434, 435, 445, 448, 455, 469, 500`
- Dans ce qui reste, conserver les fils dont le voisin de droite a un chiffre des unités strictement plus petit que 5 (il faudra opérer en parcourant les fils de gauche à droite). Le fil le plus à droite n'est pas conservé. Après cette

opération, il restera les fils :
`7, 77, 105, 126, 189, 200, 217, 266, 300, 315, 399, 406, 427, 469`

- Dans ce qui reste, conserver les fils dont le chiffre des dizaines est impair. Il reste :
`77, 217, 315, 399`
- Une fois ces opérations faites, tous les fils ont été écartés sauf un nombre pair d'entre eux. Pour désamorcer la bombe, il faut couper les deux fils du milieu dans ceux qui restent. Dans notre cas, il ne reste que quatre fils, il faut donc couper les fils 217 et 315.

En réalité, **la bombe contient 4200 fils**. Pour résoudre le défi, indiquez à Œil de faucon les numéros des deux fils à couper.

Source : https://pydefis.callicode.fr/defis/Desamorcage02/txt

☆ Exercice Bonus 5.38 (Défi Turing n°29 – puissances distincts)

Considérons a^b pour $2 \le a \le 5$ et $2 \le b \le 5$:

$$\begin{array}{llll} 2^2 = 4, & 2^3 = 8, & 2^4 = 16, & 2^5 = 32 \\ 3^2 = 9, & 3^3 = 27, & 3^4 = 81, & 3^5 = 243 \\ 4^2 = 16, & 4^3 = 64, & 4^4 = 256, & 4^5 = 1024 \\ 5^2 = 25, & 5^3 = 125, & 5^4 = 625, & 5^5 = 3125 \end{array}$$

Si l'on trie ces nombres dans l'ordre croissant, en supprimant les répétitions, on obtient une suite de 15 termes distincts : $[4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125]$.

Combien y a-t-il de termes distincts dans la suite obtenue comme ci-dessus pour $2 \le a \le 1000$ et $2 \le b \le 1000$?

Correction

```
>>> print (len(set(a**b for a in range(2,1001) for b in range(2,1001))))
977358
```

☆ Exercice Bonus 5.39 (Défi Turing n°45 – Nombre triangulaire, pentagonal et hexagonal)

Les nombres triangulaires, pentagonaux et hexagonaux sont générés par les formules suivantes :

Nom	n-ème terme de la suite	Suite
triangulaire	$T_n = \frac{n(n+1)}{2}$	$1, 3, 6, 10, 15, \dots$
pentagonal	$P_n = \frac{n(3n-1)}{2}$	$1, 5, 12, 22, 35, \dots$
hexagonal	$H_n = n(2n-1)$	$1, 6, 15, 28, 45, \dots$

1 est un nombre triangulaire, pentagonal et hexagonal car $1 = T_1 = P_1 = H_1$. Le suivant est 40755 car $40755 = T_{285} = P_{165} = H_{143}$. Trouver le suivant.

Correction

Un code naïf est le suivant mais on se rend compte que la complexité est trop élevée (*e.g.* l'exécution prend trop de temps) :

```
N = 10**5
T = [int(n*(n+1)/2) for n in range(1,N)]
P = [int(n*(3*n-1)/2) for n in range(1,N)]
H = [int(n*(2*n-1)) for n in range(1,N)]
for x in T:
	if x in P and x in H:
		print(f"{x}=T_{T.index(x)}=P_{P.index(x)}=H_{H.index(x)}")
```

On va alors réfléchir différemment : pour tout $x \in T$, on cherche s'il existe $n \in \mathbb{N}$ tel que $n(3n-1) = 2x$:

$$n = \frac{1+\sqrt{1+24x}}{6} \in \mathbb{N} \iff \texttt{(1+(1+24*x)**0.5)\%6==0}$$

Si c'est le cas, on cherche s'il existe $m \in \mathbb{N}$ tel que $m(2m-1) = x$:

$$m = \frac{1+\sqrt{1+8x}}{4} \in \mathbb{N} \iff \texttt{(1+(1+8*x)**0.5)\%4==0}$$

```
cond = False
n = 0

while cond==False:
    n += 1
    x = n*(n+1)//2
    p1 = (1+(1+24*x)**0.5)
    if p1%6==0:
        h1 = (1+(1+8*x)**0.5)
        if h1%4==0:
            print(f"{x}=T_{n}=P_{int(p1/6)}=H_{int(h1/4)}")
            if x>40755 :
                cond = True
```

```
1=T_1=P_1=H_1
40755=T_285=P_165=H_143
1533776805=T_55385=P_31977=H_27693
```

Exercice Bonus 5.40 (Défi Turing n°94 – Problème d'Euler n° 92)

Une suite d'entiers est créée de la façon suivante : le nombre suivant de la liste est obtenu en additionnant les carrés des chiffres du nombre précédent :

$$44 \xrightarrow[4^2+4^2]{} 32 \xrightarrow[3^2+2^2]{} 13 \xrightarrow[1^2+3^2]{} 10 \xrightarrow[1^2+0^2]{} 1 \xrightarrow[1^2]{} 1$$

$$85 \xrightarrow[8^2+5^2]{} 89 \xrightarrow[8^2+9^2]{} 145 \xrightarrow[1^2+4^2+5^2]{} 42 \xrightarrow[4^2+2^2]{} 20 \xrightarrow[2^2+0^2]{} 4 \xrightarrow[4^2]{} 16 \xrightarrow[1^2+6^2]{} 37 \xrightarrow[3^2+7^2]{} 58 \xrightarrow[5^2+8^2]{} 89$$

On peut voir qu'une suite qui arrive à 1 ou 89 restera coincée dans une boucle infinie. Le plus incroyable est qu'avec n'importe quel nombre de départ strictement positif, toute suite arrivera finalement à 1 ou 89.

Combien de nombres de départ inférieurs ou égal à 5 millions arriveront à 89 ?

Correction

```
d = {1:0,89:0}
for i in range(1,5000000+1):
    while i!=1 and i!=89:
        i = sum([int(i)**2 for i in str(i)])
    d[i] += 1
print(d)
```

```
{1: 704156, 89: 4295844}
```

Exercice Bonus 5.41 (Pydéfi – Le pistolet de Nick Fury)

Recherche du fonctionnement périodique : le pistolet de Nick Fury émet des impulsions successives dont l'intensité varie selon une loi mathématique. Pour calculer l'intensité de l'impulsion suivante, il suffit d'écrire en binaire l'intensité de l'impulsion émise, de renverser l'écriture de ce nombre binaire (lire de droite à gauche), de convertir le nombre obtenu en base 10 puis de lui ajouter 2 et recommencer.

Sur le pistolet, on peut régler l'intensité de l'impulsion initiale. Par exemple, si le pistolet est réglé sur 39, alors, lors

d'un tir, les impulsions émises auront pour intensité

$$39 \xrightarrow[\text{binaire}]{} 100111 \xrightarrow[\text{miroir}]{} 111001 \xrightarrow[\text{base 10}]{} 57 \xrightarrow[+2]{}$$
$$59 \xrightarrow[\text{binaire}]{} 111011 \xrightarrow[\text{miroir}]{} 110111 \xrightarrow[\text{base 10}]{} 55 \xrightarrow[+2]{}$$
$$57 \xrightarrow[\text{binaire}]{} 111001 \xrightarrow[\text{miroir}]{} 100111 \xrightarrow[\text{base 10}]{} 39 \xrightarrow[+2]{}$$
$$41 \xrightarrow[\text{binaire}]{} 101001 \xrightarrow[\text{miroir}]{} 100101 \xrightarrow[\text{base 10}]{} 37 \xrightarrow[+2]{} 39$$

On constate que pour le réglage 39, les amplitudes sont périodiques et la période est 4 ($39 \to 59 \to 57 \to 39$).

Voici un autre exemple, obtenu avec une impulsion initiale de 86 : $86 \to 55 \to 61 \to 49 \to 37 \to 43 \to 55$. Bien qu'on ne retourne jamais à la valeur 86, on obtient aussi un cycle, de longueur 5.

En revanche, pour certaines valeurs, l'amplitude n'est jamais périodique, et le comportement du pistolet est imprévisible. Si Nick le règle sur une telle valeur, le pistolet peut exploser dès qu'il a changé plus de 1024 fois d'intensité.

Afin d'améliorer l'arme de Nick Fury et ainsi rendre service au Shield, il convient de ne permettre que les réglages des valeurs de départ qui donnent lieu à un comportement périodique.

Défi : donner la séquence de toutes les valeurs convenables comprises entre 1 et 500 c'est-à-dire celles qui donnent lieu à un comportement périodique.

Source : https://pydefis.callicode.fr/defis/PistoletFury

Exercice Bonus 5.42 (Pydéfi – Les nombres heureux)

Les nombres sont parfois pourvus de qualificatifs surprenants. Il existe en effet des nombres heureux et des nombre malheureux. Pour savoir si un nombre est heureux, il faut calculer la somme des carrés de ses chiffres, et recommencer avec le résultat. Si on finit par tomber sur 1, alors le nombre est heureux. Sinon, il est malheureux.

Par exemple, le nombre 109 est heureux. En effet :

$$109 \to 1^2+0^2+9^2 = 82 \to 8^2+2^2 = 68 \to 6^2+8^2 = 100 \to 1^2+0^2+0^2 = 1$$

Par contre, 106 est malheureux. En effet :

$$\begin{aligned}
106 &\to 1^2+0^2+6^2 = 37\\
&\to 3^2+7^2 = 58\\
&\to 5^2+8^2 = 89\\
&\to 8^2+9^2 = 145\\
&\to 1^2+4^2+5^2 = 42\\
&\to 4^2+2^2 = 20\\
&\to 2^2+0^2 = 4\\
&\to 4^2 = 16\\
&\to 1^2+6^2 = 37
\end{aligned}$$

Le nombre 37 a déjà été obtenu en début de séquence, on sait donc que la série $37, 58, 89, 145, 42, 20, 4, 16$ va se répéter indéfiniment. Le nombre 1 ne sera donc jamais atteint.

Défi : l'entrée du problème est la donnée de deux bornes mini et maxi. Il faut répondre en donnant la liste des nombres heureux compris entre ces deux bornes (incluses), par ordre croissant.

Testez votre code : par exemple, si les bornes données étaient mini=109 et maxi=141, il faudrait répondre en indiquant $(109, 129, 130, 133, 139)$.

Source : https://pydefis.callicode.fr/defis/NombresHeureux

Exercice Bonus 5.43 (Pydéfi – Le problème des boîtes à sucres)

La société Syntactic Sugar emballe depuis des années des sucres cubiques en boîtes parallélépipédiques. La tradition veut que chaque boîte contienne 252 sucres disposés en 4 couches de 7×9 sucres. Les sucres étant cubiques, et les boîtes de la société Syntactic Sugar pouvant s'ouvrir sur n'importe quelle face, on peut tout aussi bien considérer qu'il s'agit d'une boîte contenant 7 couches de 4×9 sucres ou encore 9 couches de 7×4 sucres.

Un beau matin, mu par un irrépressible besoin d'innovation, le service commercial de la société Syntactic Sugar décide que des boîtes contenant 3 couches de 7×12 sucres (et donc toujours 252 sucres) seraient bien plus attractives. Il s'en suivit de nombreuses querelles sur la forme des boîtes, certains souhaitant plutôt fabriquer maintenant des boîtes contenant 3 couches de 14×6 sucres...

Jusqu'au moment où l'idée ne put que germer : et si on changeait le nombre de sucres par boîte ?

Une boîte raisonnable contenant entre 137 et 479 sucres, quel nombre de sucres choisir pour qu'il n'y ait qu'une seule forme de boîte possible, et par là même couper court aux tergiversations du service commercial, sachant que sur toutes les dimensions, on met au minimum 2 sucres (c'est à dire qu'une boîte de dimensions 15x15x1 contenant 225 sucres ne conviendrait pas, car aurait une dimension égale à 1).

Par exemple, en faisant des boîtes de 385 sucres, on est dans l'obligation de réaliser de boîtes de dimensions $5 \times 7 \times 11$. Aucune autre forme de boîte ne convient.

Défi : donner la liste des nombres de sucres qui imposent une taille de boîte unique. Notez que 316, qui impose une taille de boîte de $2 \times 2 \times 79$ est convenable, quoique fort peu pratique.

Source : https://pydefis.callicode.fr/defis/BoitesSucres

Exercice Bonus 5.44 (Défi Turing n°60 – Suicide collectif)

Les 2013 membres d'une secte ont décidé de se suicider. Pour effectuer le rituel funèbre, ils se mettent en cercle, puis se numérotent dans l'ordre de 1 à 2013. On commence à compter, à partir du numéro 1. Toutes les 7 positions, la personne désignée devra mourir. Ainsi, la première à mourir aura le n°7, la deuxième le 14, la troisième le 21, etc. Vous faites partie de cette secte, mais vous n'avez aucune envie de mourir ! Il s'agit donc de trouver la position sur le cercle qui vous permettra d'être désigné en dernier, et donc d'échapper à la mort.

Quelle est la position qui vous sauvera ?

Correction

```
k,n = 7,2013
L = list(range(1,n+1))

while n>=k:
    q,r = divmod(n,k)
    LG = L[:q*k]
    LD = L[q*k:]
    L = LD+[ LG[i] for i in range(len(LG)) if (i+1)%k!=0 ]
    n = len(L)

while n>1:
    r = k%n
    LG = L[:r-1]
    LD = L[r:]
    L = LD+LG
    n = len(L)

print(L)
```

```
[1868]
```

Mieux :

```
k,n = 6,2013 # parce que les indices commencent à 0
L = list(range(1,n+1))
```

```
while len(L)>1:
    del L[k]
    k = (k+6)%len(L) # k+6 parce qu'on vient de supprimer le k-ième
print(L)
```

```
[1868]
```

☆ Exercice Bonus 5.45 (Triplets pytagoriciens)

Le triplet d'entiers naturels non nuls (a, b, c) est pytagoricien si $a^2 + b^2 = c^2$. Pour $c = 2020$ il existe 4 triplets pytagoricien différents : $(400, 1980)$, $(868, 1824)$, $(1212, 1616)$ et $(1344, 1508)$

Pour chaque c in $[2010, 2020]$ calculer combien de triplets pytagoriciens existent.

Correction

```
dico = { c : [(a,b) for a in range(1,c) for b in range(a,c) if a**2+b**2==c**2] for c in
↪ range(2010,2021)}
print(dico) # Pour vérification
dico1 = { c : len(v) for (c,v) in dico.items() }
print(dico1)
```

```
{2010: [(1206, 1608)], 2011: [], 2012: [], 2013: [(363, 1980)], 2014: [(1064, 1710)], 2015:
↪ [(496, 1953), (775, 1860), (1023, 1736), (1209, 1612)], 2016: [], 2017: [(792, 1855)],
↪ 2018: [(1118, 1680)], 2019: [(1155, 1656)], 2020: [(400, 1980), (868, 1824), (1212, 1616),
↪ (1344, 1508)]}
{2010: 1, 2011: 0, 2012: 0, 2013: 1, 2014: 1, 2015: 4, 2016: 0, 2017: 1, 2018: 1, 2019: 1,
↪ 2020: 4}
```

☆ Exercice Bonus 5.46 (Dictionnaire ordonné)

Soit `L` une liste d'entiers positifs. Construite la liste `Y` des valeurs uniques de `L` triées par leur fréquence. Si deux valeurs apparaissent le même nombre de fois, les trier du plus petit au plus grand.

Par exemple, si L=[1, 2, 2, 2, 3, 3, 7, 7, 9] alors Y=[1, 9, 3, 7, 2].

Correction

```
L = [1, 2, 2, 2, 3, 3, 7, 7, 9]
# dictionnaire { valeur : fréquence }
dico = { i : L.count(i) for i in set(L) }
print(dico)
# sorted() sur le deuxième élément renvoyé par dico.items()
dico_sorted = { k:v for k,v in sorted(dico.items(), key=lambda item: item[1])}
print(dico_sorted)
# on extrait juste les clés
Y = [k for k in dico_sorted.keys()]
print(Y)
```

```
{1: 1, 2: 3, 3: 2, 7: 2, 9: 1}
{1: 1, 9: 1, 3: 2, 7: 2, 2: 3}
[1, 9, 3, 7, 2]
```

CHAPITRE 6

Fonctions

Une fonction est un ensemble d'instructions regroupées sous un nom et s'exécutant à la demande. Elle peut recevoir des paramètres et peut renvoyer des objets. Les fonctions aident à séparer le code en sections plus petites. Comme ça on garde le fil sur ce que chaque partie est censée faire. On obtient un code mieux écrit, mieux structuré et plus lisible.

Maintenant deux règles de base lorsqu'on écrit une fonction :

1. DRY : ne vous répétez pas (*Don't Repeat Yourself* en anglais). Si vous vous retrouvez à copier et coller souvent des morceaux de code, c'est un signe que vous pouvez isoler ce à quoi servent ces lignes de code.
2. Responsabilité unique : chaque fonction a un but unique pour une meilleure lisibilité. Le fait de séparer les responsabilités permet de définir un nom plus descriptif pour la fonction, de débugger le code plus facilement et de mieux comprendre le code. D'ailleurs, votre code doit être suffisamment lisible pour ne pas avoir besoin de trop de commentaires pour que quelqu'un d'autre le comprenne.

6.1. Fonctions prédéfinies

De nombreuses fonctions sont déjà disponibles dans Python. La liste pour la dernière version de Python est disponible ici : https://docs.python.org/3/library/functions.html

```
# A
abs()
aiter()
all()
any()
anext()
ascii()

# B
bin()
bool()
breakpoint()
bytearray()
bytes()

# C
callable()
chr()
classmethod()
compile()
complex()

# D
delattr()
dict()
dir()
divmod()

# E
enumerate()
eval()
exec()

# F
filter()
float()
format()
frozenset()

# G
getattr()
globals()

# H
hasattr()
hash()
help()
hex()

# I
id()
input()
int()
isinstance()
issubclass()
iter()

# L
len()
list()
locals()

# M
map()
max()
memoryview()
min()

# N
next()

# O
object()
oct()
open()
ord()

# P
pow()
print()
property()

# R
range()
repr()
reversed()
round()

# S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

# T
tuple()
type()

# V
vars()

# Z
zip()

# _
__import__()
```

Il arrive régulièrement de se rappeler du nom d'une fonction, mais pas forcément de ce qu'elle fait ou pas forcément de ces arguments, etc. La fonction `help` est là pour ça. Si on exécute `help(nomFonction)`, cela affichera la documentation de cette fonction résumant : son but, des recommandations d'utilisation, la liste et la description des paramètres, parfois des exemples. Voici un exemple avec la fonction `sorted` :

```
>>> help(sorted)
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

6.2. Définition d'une fonction

On peut définir nos propres fonctions au moyen de la commande `def` et les utiliser dans plusieurs scripts. La syntaxe est la suivante :

```
def FunctionName(parameters):
    statements
    return values
```

- La déclaration d'une nouvelle fonction commence par le mot-clé `def`.
- Ensuite, toujours sur la même ligne, vient le nom de la fonction (ici `FunctionName`) suivi des paramètres formels de la fonction `parameters`, placés entre parenthèses, le tout terminé par deux-points (on peut mettre autant de paramètres formels qu'on le souhaite et éventuellement aucun). [1]
- Une fois la première ligne saisie, on appuie sur la touche «Entrée» : le curseur passe à la ligne suivante avec une indentation. On écrit ensuite les instructions.
- Dès que Python atteint l'instruction `return something`, il renvoi l'objet `something` et abandonne aussitôt après l'exécution de la fonction (on parle de code mort pour désigner les lignes qui suivent l'instruction `return`). Cela peut être très pratique par exemple dans une boucle `for` : dès qu'on a le résultat voulu, on le renvoie sans avoir besoin de finir la boucle. Si l'instruction `return` est absente, la fonction renvoi l'objet `None` (ce sera le cas lorsqu'on passe un objet mutable et la fonction le modifie directement sans renvoyer un nouvel objet, par exemple une liste).

EXEMPLE
Voici un exemple : supposons de vouloir calculer les images de certains nombres par une fonction polynomiale donnée. Si la fonction en question est un peu longue à saisir, par exemple $f: x \mapsto 2x^7 - x^6 + 5x^5 - x^4 + 9x^3 + 7x^2 + 8x - 1$, il est rapidement fastidieux de la saisir à chaque fois que l'on souhaite calculer l'image d'un nombre par cette fonction. On défini alors d'abord la fonction $f: x \mapsto 2x^7 - x^6 + 5x^5 - x^4 + 9x^3 + 7x^2 + 8x - 1$, on peut ensuite l'évaluer en $x = 2$ ou en un ensemble de points :

```
def f(x):
    return 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1

print( f(2) )
print( [f(x) for x in range(5)] )
```

```
451
[-1, 28, 451, 5108, 34255]
```

Lorsque l'on définit une fonction, les variables qui apparaissent entre les parenthèses sont appelées les paramètres; par contre, lorsque l'on appelle la fonction, les valeurs entre les parenthèses sont appelées les arguments. Dans l'exemple, x est le paramètre, 2 est l'argument.

ATTENTION
Ne pas confondre la fonction `print` et l'instruction `return` :

1. Les paramètres figurant entre parenthèses dans l'en-tête d'une fonction se nomment *paramètres formels*, par opposition aux paramètres fournis lors de l'appel de la fonction qui sont appelés *paramètres effectifs*.

```
def f(x):
    return x**2

y=f(2)
print(y)
```

```
4
```

```
def f(x):
    print(x**2)

y=f(2)
print(y)
```

```
4
None
```

Par définition de `function`, toute fonction est censée renvoyer une valeur. Une fonction qui ne renvoie pas de valeur n'est pas une fonction : on appelle cela en programmation une procédure.

En Python, en fait, même les fonctions sans instruction `return` explicite renvoient une valeur qui est `None`. Le valeur `None` est une valeur qui correspond justement à l'absence de valeur. Cette valeur sert à indiquer "il n'y a pas de valeur". C'est la raison pour laquelle on appelle malgré tout fonction même les fonctions qui ne possèdent pas de `return` explicite.

On peut bien sûr composer plusieurs fonctions : soit $f, g, h, k\colon \mathbb{R} \to \mathbb{R}$ définies par $f(x) = x^2$, $g(x) = x+2$, $h(x) = f(g(x))$ et $k(x) = g(f(x))$. Nous écrirons

```
def f(x):
    return x**2

def g(x):
    return x+2

def h(x):
    return f(g(x))

def k(x):
    return g(f(x))

print(f(3),g(3),h(3),k(3))
```

```
9 5 25 11
```

En effet, $f(3) = 3^2 = 9$, $g(3) = 3+2 = 5$, $h(3) = f(g(3)) = f(5) = 5^2 = 25$ et $k(3) = g(f(3)) = g(9) = 9+2 = 11$.

Remarque (Les tests)
Il faut toujours tester ses fonctions. Un test consiste à appeler **chaque** fonctionnalité, avec un scénario qui correspond à un cas d'utilisation, et à vérifier que cette fonctionnalité se comporte comme prévu.

Visibilité d'une variable Les variables définies à l'intérieur d'une fonction **ne sont pas «visibles» depuis l'extérieur** de la fonction. On exprime cela en disant qu'une telle variable est locale à la fonction. De plus, si une variable existe déjà avant l'exécution de la fonction, tout se passe comme si, durant l'exécution de la fonction, cette variable était masquée momentanément, puis restituée à la fin de l'exécution de la fonction.

- Dans l'exemple suivant, la variable `x` est une variable locale à la fonction `f` : crée au cours de l'exécution de la fonction `f`, elle est supprimée une fois l'exécution terminée :

```
>>> def f(y):
...     x = 2
...     return 4*y
...
>>> print(f(5))
20
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

- Dans l'exemple suivant, la variable x est une variable qui vaut 6 à l'extérieur de la fonction et 7 au cours de l'exécution de la fonction f :

```
x = 6

def f(y):
    x = 7 # x est redefinie localement, m reste inchange
    return x*y*m

m = 2
print(f(1))
print(x)
```

```
14
6
```

- Dans cet exemple, en revanche, la variable x vaut 6 partout car définie à l'extérieur de la fonction et non redéfinie à l'intérieur de la fonction f :

```
def f(y):
    return x*y*m

x = 6
m = 2
print(f(1))
print(x)
```

```
12
6
```

On remarque que les variables utilisées dans la définition d'une fonction peuvent être affectées après la définition de la fonction mais avant de l'appeler.

Voici un exemple qui résume les différents cas :

```
def ma_fonction():
    ma_variable = 17
    s = f"Valeur de ma_variable dans ma_fonction : {ma_variable}"
    return s
    
ma_variable = 10
print(f"Valeur de ma_variable dans le script : {ma_variable}")
s = ma_fonction()
print(s)
print(f"Valeur de ma_variable dans le script après l'appel : {ma_variable}")
```

```
Valeur de ma_variable dans le script : 10
Valeur de ma_variable dans ma_fonction : 17
Valeur de ma_variable dans le script après l'appel : 10
```

Pour que la variable dans la fonction "écrase" la variable hors de la fonction (après appel de la focntion bien-sûr), il faut indiquer explicitement qu'il s'agit d'une variable globale (déconseillé !) :

```
def ma_fonction():
    global ma_variable
    ma_variable = 17
    s = f"Valeur de ma_variable dans ma_fonction : {ma_variable}"
    return s

ma_variable = 10
print(f"Valeur de ma_variable dans le script : {ma_variable}")
s = ma_fonction()
print(s)
print(f"Valeur de ma_variable dans le script après l'appel : {ma_variable}")
```

 © G. FACCANONI

```
Valeur de ma_variable dans le script : 10
Valeur de ma_variable dans ma_fonction : 17
Valeur de ma_variable dans le script après l'appel : 17
```

ATTENTION
Si une liste est passée comme paramètre d'une fonction et cette fonction la modifie, cette modification se répercute sur la liste initiale. Si ce n'est pas le résultat voulu, il faut travailler sur une copie de la liste.

```
>>> def squares(a):
...     for i in range(len(a)):
...         a[i] = a[i]**2
...
>>> a = [1,2,3,4]
>>> print(a)
[1, 2, 3, 4]
>>> squares(a)
>>> print(a)
[1, 4, 9, 16]
```

Notons en passant qu'ici on n'a pas écrit d'instruction `return` : une fonction qui agit sur un objet modifiable peut se passer de l'instruction `return`.

En résumé :

- Il peut y avoir zéro, un ou plusieurs paramètres en entrée.
- Il peut y avoir plusieurs résultats en sortie.
- Très important ! Il **ne faut pas confondre afficher et renvoyer une valeur**. L'affichage (par la commande `print()`) affiche juste quelque chose à l'écran. La plupart des fonctions n'affichent rien, mais renvoient une valeur (ou plusieurs). C'est beaucoup plus utile car cette valeur peut être utilisée ailleurs dans le programme (et affichée si besoin).
- Dès que le programme rencontre l'instruction `return`, la fonction s'arrête et renvoie le résultat. Il peut y avoir plusieurs fois l'instruction `return` dans une fonction mais une seule sera exécutée. On peut aussi ne pas mettre d'instruction `return` si la fonction ne renvoie rien.
- Dans les instructions d'une fonction, on peut bien sûr faire appel à d'autres fonctions !
- Il est important de bien commenter ses programmes. Pour documenter une fonction, on peut décrire ce qu'elle fait en commençant par un `docstring`, c'est-à-dire une description entourée par trois guillemets : `""" Ma fonction fait ceci et cela. """` à placer juste après l'entête. Les docstrings en début de fonction sont spéciales, et on peut y accéder soit avec `help(ma_fonction)` ou en utilisant l'attribut `ma_fonction.__doc__`. Exemple :

```
def somme_produit(a, b):
    """
    Calcule somme et produit de deux nombres

    Attrs:
    - a (int or float), b (int or float): les deux  nombres
    
    Returns:
    - somme, _produit
    """
    return a+b , a*b

# print(somme_produit.__doc__)
help(somme_produit)
```

```
Help on function somme_produit in module __main__:

somme_produit(a, b)
    Calcule somme et produit de deux nombres
```

```
    Attrs:
    - a (int or float), b (int or float): les deux  nombres

    Returns:
    - somme, _produit
```

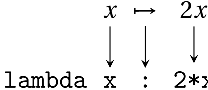
Notez que la docstring ne doit jamais décrire le fonctionnement interne de sa fonction ou méthode, mais seulement son retour. Il ne s'agit pas ici d'expliquer le code, mais d'expliquer son usage. Savoir écrire une docstring, c'est aussi savoir expliquer son code. Les docstrings sont donc un bon moyen de garder un recul dans notre façon de coder. Expliquer son code à soi-même évite bon nombre de bugs, en témoigne la fameuse méthode du canard en plastique.

6.3. Fonctions Lambda (fonctions anonymes)

Quand on définit une fonction avec `def f(x): return 2*x` on fait deux choses : on crée l'objet «fonction qui a x associe $f(x)$» puis on affecte cet objet à une variable (globale) f. Ensuite, on peut l'évaluer :

```
>>> def f(x):
...     return 2*x
...
>>> f(3)
6
```

On peut aussi créer une fonction sans lui donner de nom, c'est une fonction `lambda` :



Cette écriture se lit «fonction qui a x associe $2x$» (*i.e.* $x \mapsto 2x$).

En écrivant `lambda x: 2*x` on crée l'objet «fonction qui a x associe $2x$» sans lui donner de nom ; si on veut affecter cet objet à une variable (globale) g et l'évaluer on écrira

```
>>> g = lambda x : 2*x
>>> g(3)
6
```

ce qui équivaut à

```
>>> def g(x):
...     return 2*x
...
>>> g(3)
6
```

Une fonction lambda peut avoir plusieurs paramètres :

```
>>> somme = lambda x,y : x + y
>>> S = somme(10, 3)
>>> print(S)
13
```

On peut bien-sûr composer deux fonctions lambda. Par exemple, soit $f\colon x \mapsto x^2$ et $g\colon x \mapsto x-1$ et considérons h la composition de g avec f (i.e. $h(x) = g(f(x)) = g(x^2) = x^2 - 1$). On peut définir la fonction h tout naturellement comme suit :

 © G. Faccanoni

```
>>> f = lambda x : x**2
>>> g = lambda x : x-1
>>> h = lambda x : g(f(x))
>>> print(h(0))
-1
```

Pour éviter la tentation de code illisible, Python limite les fonctions `lambda` : **une seule ligne** et `return` **implicite**. Si on veut écrire des instructions plus compliquées, on utilise `def` (on peut toujours).

6.3.1. ☆ Arguments nommés

Si on souhaite s'assurer que les valeurs passées à une fonction vont bien correspondre à tel ou tel paramètre, on peut passer à nos fonctions des **arguments nommés**. Un argument nommé est un argument qui contient le nom d'un paramètre présent dans la définition de la fonction suivi de la valeur qu'on souhaite passer. On va pouvoir passer les arguments nommés dans n'importe quel ordre puisque Python pourra faire le lien grâce au nom avec les arguments attendus par notre fonction.

- Appelle classique : deux paramètres, passées selon l'ordre

```
def bonjour(nom,prenom):
    return f"Bonjour {prenom} {nom.upper()} !"

print(bonjour("Faccanoni","Gloria"))
```

```
Bonjour Gloria FACCANONI !
```

- Appelle avec arguments nommés : deux paramètres, passées selon le même ordre

```
def bonjour(nom,prenom):
    return f"Bonjour {prenom} {nom.upper()} !"

print(bonjour(nom="Faccanoni",prenom="Gloria"))
```

```
Bonjour Gloria FACCANONI !
```

- Appelle avec arguments nommés : deux paramètres, passées selon un autre ordre

```
def bonjour(nom,prenom):
    return f"Bonjour {prenom} {nom.upper()} !"

print(bonjour(prenom="Gloria",nom="Faccanoni"))
```

```
Bonjour Gloria FACCANONI !
```

6.3.2. ☆ Arguments par défaut

Il est possible de préciser des valeurs par défaut. Utiliser des valeurs par défaut pour les paramètres de fonctions permet aux utilisateurs d'appeler cette fonction en omettant de passer les arguments relatifs aux paramètres possédant des valeurs par défaut.

On placera toujours les paramètres sans valeur par défaut au début et ceux avec valeurs par défaut à la fin afin que les arguments passés remplacent en priorité les paramètres sans valeur.

Notez cependant qu'il faudra ici passer les arguments nommés en dernier, après les arguments sans nom.

```
def bonjour(nom,prenom="Gloria"):
    return f"Bonjour {prenom} {nom.upper()} !"

print(bonjour("Faccanoni","Daniela"))
print(bonjour("Faccanoni",prenom="Daniela"))
print(bonjour("Faccanoni"))
```

```
Bonjour Daniela FACCANONI !
Bonjour Daniela FACCANONI !
Bonjour Gloria FACCANONI !
```

Par ailleurs, aucun argument ne peut recevoir de valeur plus d'une fois. Faites donc bien attention à ne pas passer une valeur à un argument sans le nommer puis à repasser cette valeur en le nommant par inattention.

6.4. ☆ Fonctions récursives

Une fonction récursive est tous simplement une fonction qui s'appelle elle même. Lorsqu'on définit une fonction récursive, il faudra toujours faire bien attention à fournir une condition qui sera fausse à un moment ou l'autre au risque que la fonction s'appelle à l'infini.

Une fonction mathématique définie par une relation de récurrence et une condition initiale peut être programmée de manière récursive de façon naturelle. Par exemple :

$$\begin{cases} u_0 = 1, \\ u_{n+1} = 2u_n \end{cases}$$

```python
#Version recursive
def UR(n):
    if n==0: # condition de sortie
        return 1
    else:
        return 2*UR(n-1)

print(UR(6))
```

```
64
```

Parfois on peut expliciter la récursion. Dans notre exemple nous avons

$$u_{n+1} = 2u_n = 2^2u_{n-1} = \cdots = 2^{n+1}u_0,$$

ainsi $u_6 = 2^6 \times 1 = 64$ que l'on peut implémenter comme suit :

```python
#Version itérative
def UI(u):
    return 2*u

N,u = 5,1
for n in range(N+1):
    u = UI(u)
print(u)
```

```
64
```

```python
#Version explicite
def UE(u0,n):
    return 2**n*u0

N, u0 = 5, 1
u = UE(u0,N+1)
print(u)
```

```
64
```

Prenons un autre exemple : le calcul de la somme des entiers entre 1 et n. L'idée est d'expliquer comment il faut commencer et ce qu'il faut faire pour passer de l'étape $n-1$ à l'étape n. Pour commencer, si n vaut 1, la somme vaut 1. Ensuite si on a déjà calculé la somme de 1 à $n-1$, il suffit de lui ajouter n pour obtenir la somme de 1 à n. Ici aussi on peut expliciter la récursion :

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2},$$

ainsi $1+2+3+\cdots+10 = \frac{10\times 11}{5} = 55$.

Version récursive :

```
def somme(n):
    if n==1:
        return 1
    else:
        return somme(n-1)+n

print(somme(10))
```

```
55
```

Version itérative :

```
N=10
somme=0
for n in range(1,N+1):
	somme+=n
print(somme)
```

```
55
```

Version explicite :

```
N=10
somme=N*(N+1)/2
print(somme)
```

```
55.0
```

Attention :

- ne pas oublier d'initialiser c'est à dire d'expliquer ce que doit faire le programme pour les valeurs initiales. Si on n'explique pas dans notre exemple quoi faire si $n = 1$ alors le programme va chercher à calculer `somme(1)` puis `somme(0)` puis `somme(-1)` sans jamais s'arrêter ;
- ne pas demander des calculs trop complexes. L'avantage d'une écriture récursive est que c'est en général simple de notre coté mais pas forcément pour l'ordinateur. Cela veut dire qu'il faut quand même se demander si ce qu'on lui demande ne va pas être trop complexe et s'il n'y a pas des moyens de lui simplifier la tache ;
- en python, de base, on ne peut faire que 1000 appels de la même fonction de façon récursive (c'est-à-dire que, dans notre exemple, on ne pourra pas calculer `somme(1001)`.

6.5. Exercices

Exercice 6.1 (Devine le résultat - variables globales *vs* locales)
Si un script provoque une erreur, expliquer pourquoi.

Cas 1 :
```
def test():
    x = "hello"

test()
print(x)
```

Cas 2 :
```
def test():
    print(x)

x = "hello"
test()
```

Cas 3 :
```
def test():
    x="ciao"
    print(x)

x = "hello"
test()
print(x)
```

Correction
On s'intéresse à la portée des variables en Python. Le terme de "portée des variables" sert à désigner les différents espaces dans le script dans lesquels une variable est accessible c'est-à-dire utilisable. Une variable peut avoir une portée locale ou une portée globale.

Les variables définies dans une fonction sont appelées variables locales. Elles ne peuvent être utilisées que localement c'est-à-dire qu'à l'intérieur de la fonction qui les a définies. Tenter d'appeler une variable locale depuis l'extérieur de la fonction qui l'a définie provoquera une erreur. Cela est dû au fait que chaque fois qu'une fonction est appelée, Python réserve pour elle (dans la mémoire de l'ordinateur) un nouvel espace de noms (c'est-à-dire une sorte de dossier virtuel). Les contenus des variables locales sont stockés dans cet espace de noms qui est inaccessible depuis l'extérieur de la fonction. Cet espace de noms est automatiquement détruit dès que la fonction a terminé son travail, ce qui fait que les valeurs des variables sont réinitialisées à chaque nouvel appel de fonction.

Les variables définies dans l'espace global du script, c'est-à-dire en dehors de toute fonction sont appelées des variables globales. Ces variables sont accessibles (= utilisables) à travers l'ensemble du script et accessible en lecture seulement à l'intérieur des fonctions utilisées dans ce script. Pour le dire très simplement : une fonction va pouvoir utiliser la valeur d'une variable définie globalement mais ne va pas pouvoir modifier sa valeur c'est-à-dire la redéfinir. En effet, toute variable définie dans une fonction est par définition locale ce qui fait que si on essaie de redéfinir une variable globale à l'intérieur d'une fonction on ne fera que créer une autre variable de même nom que la variable globale qu'on souhaite redéfinir mais qui sera locale et bien distincte de cette dernière. Cette remarque n'est plus valable dès qu'on travaille avec des objet mutables comme les listes.

Cas 1 : Une variable déclarée dans une fonction ne sera visible que dans cette fonction. On parle alors de variable locale :

```
>>> def test():
...     x = "hello"
...
>>> test()
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Cas 2 : Une variable déclarée en dehors d'une fonction est visible à l'intérieur de la fonction. On parle alors de variable globale.

```
>>> def test():
...     print(x)
...
>>> x = "hello"
>>> test()
hello
```

Cas 3 : Une variable globale (donc déclarée en dehors d'une fonction) peut être redéfinie à l'intérieur de la fonction mais elle reprendra sa valeur globale en dehors.

```
>>> def test():
...     x="ciao"
...     print(x)
...
```

```
>>> x = "hello"
>>> test()
ciao
>>> print(x)
hello
```

Remarque : dans certaines situations, il serait utile de pouvoir modifier la valeur d'une variable globale depuis une fonction, notamment dans le cas où une fonction se sert d'une variable globale et la manipule. Pour faire cela, il suffit d'utiliser le mot clef `global` devant le nom d'une variable globale utilisée localement afin d'indiquer à Python qu'on souhaite bien modifier le contenu de la variable globale et non pas créer une variable locale de même nom.

```
>>> def test():
...     global x
...     x="ciao"
...     print(x)
...
>>> x = "hello"
>>> test()
ciao
>>> print(x)
ciao
```

☆ Exercice Bonus 6.2 (Devine le résultat - Function scope)
Étudier le code ci-dessous et prévoir le résultat avant de l'exécuter.

```
def outer_func():
    def inner_func():
        a = 9
        print(f'inside inner_func, a is {a:d} (id={id(a):d})')
        print(f'inside inner_func, b is {b:d} (id={id(b):d})')
        print(f'inside inner_func, len is {len:d} (id={id(len):d})')

    len = 2
    print(f'inside outer_func, a is {a:d} (id={id(a):d})')
    print(f'inside outer_func, b is {b:d} (id={id(b):d})')
    print(f'inside outer_func, len is {len:d} (id={id(len):d})')
    inner_func()

a, b = 6, 7
outer_func()
print(f'in global scope, a is {a:d} (id={id(a):d})')
print(f'in global scope, b is {b:d} (id={id(b):d})')
print(f'in global scope, len is {len} (id={id(len):d})')
```

Source : https://scipython.com/book/chapter-2-the-core-python-language-i/examples/function-scope/

Correction

```
inside outer_func, a is 6 (id=140203788927376)
inside outer_func, b is 7 (id=140203788927408)
inside outer_func, len is 2 (id=140203788927248)
inside inner_func, a is 9 (id=140203788927472)
inside inner_func, b is 7 (id=140203788927408)
inside inner_func, len is 2 (id=140203788927248)
in global scope, a is 6 (id=140203788927376)
in global scope, b is 7 (id=140203788927408)
in global scope, len is <built-in function len> (id=140203789537888)
```

Ce programme définit une fonction, `inner_func`, imbriquée dans une autre, `outer_func`. Après ces définitions, l'exécution se déroule comme suit :

1. Les variables globales a=6 et b=7 sont initialisées.
2. outer_func est appelée :
   2.1. outer_func définit une variable locale, len=2.
   2.2. Les valeurs de a et b sont affichées ; elles n'existent ni localement ni dans la fonction englobante, de sorte que Python les cherche et les trouve globalement : leurs valeurs (6 et 7) sont affichées.
   2.3. La valeur de la variable locale len (2) est affichée.
   2.4. inner_func est appelé :
      2.4.1. Une variable locale, a=9, est définie.
      2.4.2. La valeur de cette variable locale est affichée.
      2.4.3. La valeur de b est affichée ; b n'existe pas localement, alors Python la cherche dans la fonction englobante outer_func. Elle n'y est pas trouvée non plus, alors Python procède à une recherche globale où elle est trouvée : la valeur b=7 est affichée.
      2.4.4. La valeur de len est affichée : len n'existe pas localement, mais elle se trouve dans la fonction englobante puisque len=2 est définie dans outer_func : sa valeur est affichée.
3. Une fois l'exécution de outer_func terminée, les valeurs de a et b globales sont affichées.
4. La valeur de len est affichée. Cette valeur n'est pas définie globalement, de sorte que Python recherche ses propres noms de fonctions built-in : len est la fonction intégrée permettant de déterminer la longueur des séquences. Cette fonction est elle-même un objet et elle fournit une brève description d'elle-même sous forme de chaîne de caractères lorsqu'elle est affichée.

Bien noter que dans outer_func le nom len est l'objet entier 2. Cela signifie que la fonction intégrée len d'origine n'est pas disponible dans cette fonction (et qu'elle n'est pas non plus disponible dans la fonction incluse, inner_func).

Exercice 6.3 (Devine le résultat - bêtisier)

Voici un bêtisier pour mieux comprendre les règles. Proposer une correction.

Cas 1 :
```
def f(x)
```

Cas 2 :
```
def f(x):
return 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
```

Cas 3 :
```
def f(x):
	2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1

print(f(2))
```

Cas 4 :
```
def f(x):
	a = 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
	return a
	print('Hello')

print(f(2))
```

Correction

Cas 1 : il manque les deux-points en fin de ligne :
```
>>> def f(x)
  File "<stdin>", line 1
    def f(x)
           ^
SyntaxError: expected ':'
```

Cas 2 : il manque l'indentation :
```
>>> def f(x):
... return 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
  File "<stdin>", line 2
    return 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
    ^^^^^^
IndentationError: expected an indented block after function definition on line 1
```

Cas 3 : il manque le mot `return` et donc tout appel de la fonction aura comme réponse `None` :

```
>>> def f(x):
...     2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
...
>>> print(f(2))
None
```

Cette fonction exécute le calcul mais elle ne renvoie pas d'information spécifique. Pour qu'une fonction renvoie une certaine valeur, il faut utiliser le mot-clé `return`.

Cas 4 : l'instruction `print('Hello')` n'est jamais lue par Python car elle apparaît après l'instruction `return` :

```
>>> def f(x):
...     a = 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
...     return a
...     print('Hello')
...
>>> print(f(2))
451
```

Attention : LE SILENCE EST D'OR!

À partir de ce moment, pour chaque exercice **on écrira des fonctions qui n'affichent quoi que ce soit, mais retournent silencieusement le résultat attendu**, même s'il s'agit d'une chaîne de caractères (sauf si c'est indiqué explicitement dans l'énoncé de l'exercice). Pour vérifier la fonction, on utilisera bien-sûr des `print` mais juste pour afficher ce que la fonction aura renvoyé. Voici un exemple

```
def ma_fonction(x):
    y = ...
    return y

# TESTS
x1 = ...
y1 = ma_fonction(x1)
print( f"Avec x={x} ma fonction renvoie {y1}")
```

On pourra bien-sûr utiliser quelques `print` pour le débogage pendant le développement, mais ne pas oublier de les commenter à la fin.

Exercice 6.4 (Devine le résultat)

Soit la fonction

```
def f(x,a,b):
    if a>b:
        a,b=b,a
    if x<=a:
        return a
    elif x>=b:
        return b
    else:
        if (x-a)>(b-x):
            return b
        else:
            return a
```

Calculer $f(0,0,0)$, $f(-2,0,3)$, $f(-2,3,0)$, $f(2,0,2)$, $f(1,0,2)$, $f(3,-1,-2)$.

Correction

$f(0,0,0)=0$ $f(-2,0,3)=0$ $f(-2,3,0)=0$ $f(2,0,2)=2$ $f(1,0,2)=0$ $f(3,-1,-2)=-1$

Écrivons cette fonction en langage mathématiques :

$$f(x,a,b)=\begin{cases} f(x,b,a) & \text{si } a>b\\ a & \text{si } x\le a \text{ avec } a\le b\\ b & \text{si } x\ge a \text{ avec } a\le b\\ b & \text{si } (x-a)>(b-x) \text{ avec } a\le b\\ a & \text{sinon}\end{cases}$$

Si $a>b$ on échange a et b : on peut donc restreindre l'étude au cas $a\le b$ et étudier :

$$g(x,a,b)=\begin{cases} a & \text{si } x\le a \text{ ou si } (x-a)\le(b-x)\\ b & \text{si } x\ge a \text{ ou si } (x-a)>(b-x)\end{cases}=\begin{cases} a & \text{si } x\le \frac{a+b}{2}\\ b & \text{si } x>\frac{a+b}{2}\end{cases}$$

et

$$f(x,a,b)=\begin{cases} g(x,b,a) & \text{si } a>b\\ g(x,a,b) & \text{sinon}\end{cases}$$

ainsi

$$f(0,0,0)=0,\qquad f(-2,0,3)=0,\qquad f(-2,3,0)=f(-2,0,3)=0,$$
$$f(2,0,2)=2,\qquad f(1,0,2)=0,\qquad f(3,-1,-2)=(3,-2,-1)=-1.$$

Exercice 6.5 (J'écris mes premières fonctions)

Pour chaque fonction à écrire, utiliser le mot clé `def`, vérifier s'il y a des paramètres en entrée, vérifier si elle doit renvoyer un objet avec le mot clé `return`, puis valider la fonction sur des tests bien choisis (dans cette exercice ne pas utiliser la fonction `print` à l'intérieur d'une fonction).

Sans paramètres d'entrée, avec un résultat en sortie

- écrire une fonction appelée `my_PI()` qui renvoi le nombre `3.1415`

Avec un ou plusieurs paramètres d'entrée et un résultat en sortie

- écrire une fonction appelée `euro2dollars(montant)` qui, pour une somme d'argent `montant` donnée en entrée (exprimée en euros), renvoie sa valeur en dollars (au moment de la rédaction de cet exercice 1 €=1.13$).
- écrire une fonction appelée `calcul_puissance(x,n)` qui renvoi x^n.
- écrire une fonction appelée `somme_produit(x,n)` qui calcule et renvoie la somme et le produit de deux nombres donnés en entrée.

Avec un paramètre d'entrée sans résultats en sortie Une fonction sans l'instruction `return` est appelée souvent "procédure". Dans ce cas la fonction renvoie implicitement la valeur `None`.

- écrire une fonction appelée `modifier_liste(L)` qui prend comme paramètre une liste L et qui la modifie en ajoutant à la fin l'élément `"coucou"`)(elle ne renvoie aucun objet). Par exemple, les instructions commande `L=[1,5,10]; print(L); modifier_liste([1,5,10]); print(L);` afficherons `[1,5,10] [1,5,10,"coucou"]`

Correction

La fonction `print` sert exclusivement à afficher ce que la fonction renvoie ou, dans le cas d'une liste donnée en entrée d'une fonction, à afficher la liste après qu'on ait fait appelle à la fonction.

~~Entrée~~, Sortie

```
def  my_PI():
    return 3.1415

# IDEM
# my_PI = lambda : 3.1415

resultat = my_PI()
print("La fonction renvoie {resultat}")
```

```
La fonction renvoie {resultat}
```

Attention, si on écrit `print(my_PI)` (sans les parenthèses), on obtient un message comme `<function my_PI at 0x7fddb64cb520>`.

Entrée, Sortie

```
def  euro2dollar(montant):
        return 1.13*montant

# IDEM
# euro2dollar = lambda montant :
 ↪   1.13*montant

mE = 10
mD = euro2dollar(mE)
print(f"{mE} euro = {mD:.2f} dollar")
```

```
10 euro = 11.30 dollar
```

Entrée, Sortie

```
def  calcul_puissance(x,n):
    return x**n

# IDEM
# calcul_puissance = lambda (x,n) : x**n

resultat = calcul_puissance(2,10)
print("La fonction renvoie {resultat}")
```

```
La fonction renvoie {resultat}
```

Entrée, Sortie

```
def somme_produit(a,b):
    return a+b, a*b

# IDEM
# somme_produit = lambda (a,b) : a+b, a*b

som, pro = somme_produit(6,7)
print(som)
print(pro)
```

```
13
42
```

Entrée, ~~Sortie~~

```
def  modifier_liste(L):
    L.append("coucou")

A = [1,5,10]
print(A)
modifier_liste(A)
print(A)
```

```
[1, 5, 10]
[1, 5, 10, 'coucou']
```

Exercice 6.6 (Valeur absolue)
Implémenter une fonction `myabs` : $x \mapsto |x|$ **sans** utiliser la fonction valeur absolue `abs` prédéfinie.

Correction
Quelques implémentations :

①
```
def myabs(x):
    if x<0:
        x=-x
    return x
# TESTS
print(myabs(-1),myabs(0),myabs(1))
```

```
1 0 1
```

② Avec un opérateur ternaire :

```
myabs = lambda x : x if (x>=0) else -x
# TESTS
print(myabs(-1),myabs(0),myabs(1))
```

```
1 0 1
```

③ On utilise la propriété `a*True=a` et `a*False=0` :

```
myabs = lambda x : (x>=0)*(x) + (x<0)*(-x)
# TESTS
print(myabs(-1),myabs(0),myabs(1))
```

```
1 0 1
```

Exercice 6.7 (Pair Impair)

Implémenter une fonction `oddEven` qui, pour un nombre entier N donné, renvoie la chaîne de caractère "even" ou "odd" selon que N soit pair ou impair.

Correction

①
```
def oddEven(N):
	if N%2==0:
		return "even"
	else:
		return "odd"

# oddEven = lambda N : "even" if N%2==0
↪ else "odd"

print(oddEven(2),oddEven(3))
```

```
even odd
```

②
```
def oddEven(N):
	L = ["even","odd"]
	return L[N%2]

# oddEven = lambda N : ["even","odd"]
↪ [N%2]

print(oddEven(2),oddEven(3))
```

```
even odd
```

Exercice 6.8 (Premières fonctions λ)

Écrire les fonctions suivantes d'abord comme une fonction `lambda` puis avec le mot clé `def`. Vérifier leur définition sur des tests bien choisis.

$$f_1\colon \mathbb{R}\to\mathbb{R},\quad x\mapsto x^2+5$$

$$f_2\colon \mathbb{R}^2\to\mathbb{R},\quad (x,y)\mapsto x+y-2$$

$$f_3\colon \mathbb{R}\to\mathbb{R}^2,\quad x\mapsto (x^2,x^3)$$

$$f_4\colon \mathbb{R}^2\to\mathbb{R}^2,\quad (x,y)\mapsto (x+y,x-y)$$

$$f_5\colon \mathbb{R}\to\mathbb{R},\quad x\mapsto \begin{cases}3 & \text{si } x<10\\ -2x & \text{si } 10\le x<15\\ x^2 & \text{sinon}\end{cases}$$

Correction

- Avec des fonctions lambda :

①
```
f1 = lambda x : x**2+5
print( f1(2) )
```

```
9
```

②
```
f2    = lambda x,y : x+y-2 # ne pas mettre les paramètres entre parenthèses ou
↪ crochets
f2bis = lambda t : t[0]+t[1]-2 # le paramètre est un tuple ou une liste (ou un
↪ string)
print( f2(2,3) )
print( f2bis((2,3)) ) # noter les doubles parenthèses: l'argument est un tuple
print( f2bis([2,3]) ) # noter ([...]) : l'argument est une liste
```

```
3
3
3
```

③
```
f3    = lambda x : (x**2,x**3) # ne pas oublier les () ou les [] pour le résultat
f3bis = lambda x : [x**2,x**3] # ne pas oublier les () ou les [] pour le résultat
print( f3(2) )
print( f3bis(2) )
```

```
(4, 8)
[4, 8]
```

④
```
f4        = lambda x,y : (x+y,x-y)
f4bis     = lambda x,y : [x+y,x-y]
f4ter     = lambda t : (t[0]+t[1],t[0]-t[1])
f4quatuor = lambda t : [t[0]+t[1],t[0]-t[1]]
print( f4(2,3) )
print( f4bis(2,3) )
print( f4ter((2,3)) )
print( f4ter([2,3]) )
print( f4quatuor((2,3)) )
print( f4quatuor([2,3]) )
```

```
(5, -1)
[5, -1]
(5, -1)
(5, -1)
[5, -1]
[5, -1]
```

⑤
```
f5    = lambda x : 3*(x<10) - 2*x*(10<=x<15) + x**2*(x>=15)
f5bis = lambda x : 3 if (x<10) else ( -2*x if (10<=x<15) else x**2 )
print( f5(2), f5(12), f5(22) )
print( f5bis(2), f5bis(12), f5bis(22) )
```

```
3 -24 484
3 -24 484
```

Notons que ces deux implémentations ne sont pas tout à fait équivalentes. En effet, les tests dans `f5` sont toujours fait, alors que dans `f5bis` on entre dans un `else` si et seulement si le premier test n'est pas satisfait.

- Avec `def` :

```
def f1(x) :
	return x**2+5

def f2(x,y) :
	return x+y-2

def f2bis(t) :
	return t[0]+t[1]-2

def f3(x) :
	return (x**2,x**3)

def f4(x,y) :
	return (x+y,x-y)
```

```python
def f5(x) :
    if x<10:
        return 3
    elif x<15:
        return -2*x
    else:
        return x**2
        
def f5bis(x) :
    if x<10:
        return 3
    else:
        if x<15:
            return -2*x
        else:
            return x**2
        
# TESTS
print(f1(2))
print(f2(2,3))
print(f2bis((2,3)))
print(f2bis([2,3]))
print(f3(2))
print(f4(2,3))
print(f5(2),f5(12),f5(22))
print(f5bis(2),f5bis(12),f5bis(22))
```

```
9
3
3
3
(4, 8)
(5, -1)
3 -24 484
3 -24 484
```

Exercice 6.9 (Doublons)
Écrire une fonction qui retourne `True` si la liste L contient au moins deux éléments identiques, `False` sinon.

Correction

```python
# Version 1
# def doublons(L):
#     return len(L) == len(set(L))

# Version 2
doublons = lambda L : len(L) == len(set(L))

print(doublons(['a','b','a']), doublons([1,2,5,89]))
```

```
False True
```

Exercice 6.10 (Divisibilité)
Écrire une fonction qui retourne 1 si n est un multiple de 2014, 0 sinon.

Correction

```
# Version 1
# def mul2014(n):
#     if n%2014 == 0 :
#         return 1
#     else :
#         return 0

# Version 2
# def mul2014(n):
#     return 1 if n%2014 == 0 else 0

# Version 3
mul2014 = lambda n : 1 if n%2014 == 0 else 0

# Version 4 : on utilise la propriété 1*True=1 et 1*False=0
# mul2014 = lambda n : 1*(n%2014==0)

print(mul2014(2014), mul2014(2015))
```

```
1 0
```

Exercice 6.11 (Multiple de 10 le plus proche)

Soit $n \in \mathbb{N}^*$. Écrire une fonction qui arrondi n au multiple de 10 le plus proche. Si deux multiples sont à la même distance, elle renvoie le plus petit.

Correction

```
def roundToNearest(N):
    q,r=divmod(N,10)
    if r<=5:
        return q*10
    else:
        return q*10+10

# IDEM
# roundToNearest = lambda N : (N//10)*10 if (N%10)<=5 else (N//10)*10+10

print(roundToNearest(15))
print(roundToNearest(29))
```

```
10
30
```

Exercice 6.12 (Radars routiers)

Les radars routiers permettent de mesurer la vitesse d'un véhicule et le verbaliser s'il dépasse la limite autorisée. Comme pour toute mesure, il y a un des incertitudes. C'est pour cela que la loi prévoit que pour toute mesure faite par un radar fixe, on doit retirer 5 km/h à la vitesse mesurée si elle est inférieure à 100 km/h et 5% de la vitesse si elle est supérieure à 100 km/h. Le résultat est la vitesse retenue pour le véhicule pour savoir si elle dépasse la limitation de vitesse. Ainsi, pour une vitesse mesurée de 60 km/h, on retiendra comme vitesse 55 km/h et pour une vitesse mesurée de 110 km/h, on retiendra une vitesse de 104.5 km/h.

Écrire une fonction qui prend en entrée la vitesse mesurée v et renvoie la vitesse retenue pour la verbalisation après retrait de la marge d'erreur prévue par la loi.

Correction

```
# Version 1
#def v_retenue(v):
```

```
#    if v<100:
#        return v-5
#    else:
#        return v*0.95

# Version 2
v_retenue = lambda v : (v-5) if (v<100) else (v*0.95)

# Version 3
# v_retenue = lambda v : (v-5)*(v<100)+(v*0.95)*(v>=100)

# TESTS
for v in [60,100,110]:
    print(f"vitesse = {v}, vitesse retenue = {v_retenue(v)}")
```

```
vitesse = 60, vitesse retenue = 55
vitesse = 100, vitesse retenue = 95.0
vitesse = 110, vitesse retenue = 104.5
```

☆ Exercice Bonus 6.13 (Soirée parents-enfants)

À une soirée, chaque enfant (indiqué par une lettre minuscule) doit être accompagné d'au moins un parent (indiqué par la même lettre en majuscule). Afficher les enfants qui sont venus sans parent.

Par exemple, `soiree('AbaaBBc')` donnera `'c'` et `soiree('bbDeaeBfA')` donnera `'eef'`.

Source : https://twitter.com/riko_schraf/status/1555265499373752321

Correction

```
def soiree(s):
    sol = ""
    for c in s:
        if c.islower() and c.upper() not in s:
            sol = sol+c
    return sol

for s in [ 'AbaaBBc' , 'bbDeaeBfA' ]:
    print(f"{s = }, {soiree(s) = }")
```

```
s = 'AbaaBBc', soiree(s) = 'c'
s = 'bbDeaeBfA', soiree(s) = 'eef'
```

☆ Exercice Bonus 6.14 (Leet speak)

Un hacker remplace les «A» par des «4», les «E» par des «3», les «I» par des «1», les «O» par des «0» et les «S» par des «5». Écrire une fonction qui transforme une chaîne de caractère en majuscules avec les changements indiqués. Par exemple, `hacker("un quizz facile")` donnera `"UN QU1ZZ F4C1L3"`.

Source : https://twitter.com/riko_schraf/status/1555609768898793475

Pour en savoir plus sur le *leet speak* on pourra consulter la page https://fr.wikipedia.org/wiki/Leet_speak.

Correction

```
def hacker(s):
    s = s.upper()
    return
      s.replace("A","4").replace("E","3").replace("I","1").replace("O","0").replace("S","5")

# TESTS
for s in [ 'un quizz facile' , "AEIOS"]:
    print(f"{s = }, {hacker(s) = }")
```

```
s = 'un quizz facile', hacker(s) = 'UN QU1ZZ F4C1L3'
s = 'AEIOS', hacker(s) = '43105'
```

En version compacte :

```
hacker = lambda s :
↪  s.upper().replace("A","4").replace("E","3").replace("I","1").replace("O","0").replace("S","5")

# TESTS
print(*[ f"hacker('{s}') = {hacker(s)}" for s in ('un quizz facile' , "AEIOS") ],sep="\n")
```

```
hacker('un quizz facile') = UN QU1ZZ F4C1L3
hacker('AEIOS') = 43105
```

Exercice 6.15 (Masquer tickets de caisse)

Écrire une fonction qui masque tous les chiffres d'une chaîne sauf les 4 derniers, comme sur les tickets de caisse. Exemples `masquer("123456")` donnera `"**3456"`; `masquer("789")` donnera `"789"`.

Source : https://twitter.com/riko_schraf/status/1554472875163324416

Correction

```
def masquer(s):
    l = len(s)
    if l<5:
        return s
    else:
        return "*"*(l-4)+s[-4:]

# TESTS
for s in ( "123456" , "78" ):
    print(f"{s = }, {masquer(s) = }")
```

```
s = '123456', masquer(s) = '**3456'
s = '78', masquer(s) = '78'
```

Exercice Bonus 6.16 (Fusion de listes)

Deux succursales A et B d'une même entreprise ont chacune un fichier relatif aux articles qu'elles ont en stock. Chaque enregistrement est une liste de tuples. Chaque tuple comprend le code d'un article et la quantité disponible en stock pour la succursale concernée. Les deux succursales utilisent les mêmes codes pour les mêmes articles, mais elles n'ont pas obligatoirement les mêmes articles en stock. Par exemple, la liste $A = [(1,10),(2,20)]$ indique que la succursale A a en stock 10 éléments de la référence 1 et 20 éléments de la référence 2; la liste $B = [(2,1),(3,2)]$ indique que la succursale B a en stock 1 élément de la référence 2 et 2 éléments de la référence 3.

Les deux succursales décident de fusionner leur stock. Écrire une fonction qui permet de remplacer les deux listes par une seule liste de tuples où chaque tuple contient la référence de l'article et la quantité disponible en stock sur l'ensemble des deux succursales. Dans notre exemple, on devra obtenir la liste $F = [(1,10),(2,21),(3,2)]$.

Correction

Les références sont des entiers qui peuvent être utilisés comme clé d'une liste :

```
def fusion(A,B):
    max_ref = max([r for r,q in A+B])
    LF = [0 for _ in range(max_ref+1)] # pas optimale car on pourrait alloer une liste très
    ↪  grande même avec une seule référence
    for r,q in A+B :
        LF[r] += q
    return [(r,v) for r,v in enumerate(LF) if v>0]
```

```
# TEST
A = [ (1,10), (2,20) ]
B = [ (2,1), (3,2) ]
print(fusion(A,B))
```

```
[(1, 10), (2, 21), (3, 2)]
```

Meilleur choix : dictionnaires.[2]

```
def fusion(A,B):
    dA, dB = dict(A), dict(B)
    dF = dict(A+B) # si une clé est dans A et B, la valeur dans B écrase celle dans A
    for key in dF:
        if key in dA and key in dB:
            dF[key] = dA[key]+dB[key]
    return [(k,v) for k,v in dF.items()]

# TEST
A = [ (1,10), (2,20) ]
B = [ (2,1), (3,2) ]
print(fusion(A,B))
```

```
[(1, 10), (2, 21), (3, 2)]
```

Exercice 6.17 (Liste qui transforme en 0 les termes à partir du premier 0)

Écrire une fonction qui transforme un tableau numérique en mettant des 0 à partir du premier 0 trouvé. Exemple `zero([6,8,0,4,6,3])` donnera `[6,8,0,0,0,0]`. S'il n'y a pas de 0, le tableau reste inchangé.

Source : https://twitter.com/riko_schraf/status/1554156116128440321

Correction

```
def zero(L):
    if 0 in L:
        idx = L.index(0)
        S = L[:idx+1] + [0]*len(L[idx+1:])
    else:
        S = L
    return S

# TESTS
for L in ( [6,8,0,4,3,6] , [6,8,4,3,6] ):
    print(f"{L = }, {zero(L) = }")
```

```
L = [6, 8, 0, 4, 3, 6], zero(L) = [6, 8, 0, 0, 0, 0]
L = [6, 8, 4, 3, 6], zero(L) = [6, 8, 4, 3, 6]
```

Exercice 6.18 (Password)

Stephan et Sophia ont oublié la sécurité et utilisent des mots de passes simples pour tout. Aide Nikola à développer un module de sécurité de mot de passes. Le mot de passe est considéré comme fort si sa longueur est supérieur ou égale à 10 symboles, il a au moins un chiffre, et doit contenir au moins une lettre majuscule et une lettre minuscule.

Entrée : une chaîne de caractères contenant uniquement des lettres latines ASCII ou des chiffres.

Sortie : le booléen `True` si le mot de passe respecte toutes les contraintes, `False` sinon.

Source : https://py.checkio.org/fr/mission/house-password/

2. Sujet 20.1 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

Correction

```
password = lambda data : not( len(data)<10 or data.isdigit() or data.isalpha() or
   data.isupper() or data.islower() )

# TESTS
for p in ['A1213pokl', 'bAse730onE4', 'asasasasasasasaas', 'QWERTYqwerty', '123456123456',
   'QwErTy911poqqqq']:
    print(f"La password {p} est acceptable ? {password(p)}")
```

```
La password A1213pokl est acceptable ? False
La password bAse730onE4 est acceptable ? True
La password asasasasasasasaas est acceptable ? False
La password QWERTYqwerty est acceptable ? False
La password 123456123456 est acceptable ? False
La password QwErTy911poqqqq est acceptable ? True
```

Exercice 6.19 (Distance de Hamming)

La distance de Hamming entre deux chaînes de caractères est le nombre de positions auquels les caractères correspondants dans les deux chaînes sont différents. Par exemple, la distance de Hamming entre “noiseless” et “nuisances” est 5.

n	o	i	s	e	l	e	s	s
n	u	i	s	a	n	c	e	s

Écrire une fonction qui renvoie la distance de Hamming de deux chaînes de caractères.

Correction

```
hamming = lambda s1, s2 : sum([c1 != c2 for c1, c2 in zip(s1, s2)])

s1 = 'noiseless'
s2 = 'nuisances'
H = hamming(s1, s2)
print(f'The Hamming distance between the strings "{s1:s}" and "{s2:s}" is {H:d}.')
```

```
The Hamming distance between the strings "noiseless" and "nuisances" is 5.
```

Exercice 6.20 (Nombre de Harshad)

Un nombre Harshad est un nombre entier divisible par la somme de ses chiffres (par exemple, 21 est divisible par $2+1=3$ et est donc un nombre de Harshad).

Écrire une fonction qui renvoye `True` ou `False` si n est un nombre d’Harshad ou non respectivement.

Écrire une fonction lambda qui prend en entrée L, R avec $L<R$ et renvoie la liste des nombres entre L et R (inclus) qui sont de Harshad.

Correction

```
is_harshad = lambda n: not n % sum([int(c) for c in str(n)])
print(is_harshad(201),is_harshad(211))

list_harshad = lambda L,R: [n for n in range(L,R+1) if is_harshad(n)]
print(list_harshad(10,40))
```

```
True False
[10, 12, 18, 20, 21, 24, 27, 30, 36, 40]
```

Exercice 6.21 (Validité d'un code de photocopieuse)

Le code d'une photocopieuse est un numéro N composé de 4 chiffres. Les codes corrects ont le chiffre le plus à droite égal au reste de la division par 7 de la somme des trois autres chiffres. Ainsi, le code 5739 est incorrect car $5+7+3=15$ et $15 \mod 7 = 1 \neq 9$ tandis que 5731 est correct.

Le but de cet exercice est de créer une fonction qui prend en entrée le code et qui renvoie `"VALIDE"` ou `"NON VALIDE"` (bien noter : on renvoie une chaîne de caractère, on n'utilise pas `print` dans la fonction).

Correction

```
def photocopieuse(N):
    sN = str(N)
    n = sum(int(x) for x in sN[:-1])
    if n%7 == int(sN[-1]):
        return "VALIDE"
    else:
        return "NON VALIDE"

# photocopieuse = lambda N : photocopieuse = lambda N : ["NON VALIDE", "VALIDE"][sum(int(x)
↪  for x in str(N)[:-1])%7 == int(str(N)[-1])]

# TESTS
for N in [5739,5731]:
    print(f"N = {N} {photocopieuse(N)}")
```

```
N = 5739 NON VALIDE
N = 5731 VALIDE
```

Notons qu'on peut utiliser `[s0,s1][test]` qui donne `s0` si `test` est `False` (interprété comme 0), `s1` si `test` est `True` (interprété comme 1) :

```
def photocopieuse(N):
    sN = str(N)
    n = sum(int(x) for x in sN[:-1])
    return ["NON VALIDE", "VALIDE"][n%7 == int(sN[-1])]
```

Exercice 6.22 (Numéro de sécurité sociale)

Le numéro N de sécurité social est composé de 13 chiffres (=n) suivis d'une clé de 2 chiffres (=c). La clé permet de vérifier s'il n'y a pas eu d'erreur en reportant le numéro sur un formulaire ou lors du transfert informatique par exemple, car on doit avoir

$$c = 97 - r, \qquad \text{où } r = \text{reste de la division euclidienne de } n \text{ par } 97.$$

Exemple : si le numéro de sécurité social est N = 9 700 000 000 100 94 alors $n = 9\,700\,000\,000\,100$ et le reste de la division euclidienne de n par 97 est 3. La clé est bien $97-3=94$. Si on a commis une erreur en écrivant n, il y a peu de chances d'avoir écrit un nombre qui a la même clé.

Le but est de créer une fonction qui prend en entrée le numéro de sécurité sociale et la clé et qui renvoie `"VALIDE"` si la clé correspond au numéro et `"NON VALIDE"` sinon.

Correction

```
def verific(N):
    sN = str(N)
    n = int(sN[:13])
    cle = int(sN[-2:])
    if 97-n%97==cle:
        return "VALIDE"
    else:
        return "NON VALIDE"
```

```
#def verific(N):
#     sN = str(N)
#     n = int(sN[:13])
#     cle = int(sN[-2:])
#     return ["VALIDE", "NON VALIDE"][97-n%97!=cle]

# TESTS
for N in [970000000010094, 123456789101193, 223456789101193, 223455789101120]:
    print(f"N={N} {verific(N)}")
```

```
N=970000000010094 VALIDE
N=123456789101193 VALIDE
N=223456789101193 NON VALIDE
N=223455789101120 NON VALIDE
```

Exercice 6.23 (Numéro ISBN-10)

L'*International Standard Book Number* ISBN-10 est un numéro N à 10 chiffres qui sert à identifier un livre. Le premier chiffre représente le pays, ensuite un bloc de chiffres est attribué à un éditeur, plus un autre bloc est le numéro donné par l'éditeur au livre. Le dernier chiffre est la clé, calculé de telle sorte que si $a_0a_1\dots a_9$ désigne un numéro ISBN, alors l'entier

$$\sum_{i=0}^{9}(i+1)a_{9-i}$$

soit divisible par 11.

Créer deux fonctions : une fonction qui vérifie si un code ISBN-10 donné est correct et une fonction qui, pour 9 chiffres donnés, renvoie la clé.

Correction

```
def verific(code):
    s = sum( (i+1)*int(str(code)[9-i]) for i in range(10) )
    return s%11==0

def cle(code_sans_cle):
    s=sum( (i+1)*int(str(code_sans_cle)[9-i]) for i in range(1,10) )
    return -s%11

# TESTS
for N in [ 2100574223 , 2729812563 ] :
    code_sans_cle = int(str(N)[:-1])
    print(f"verific({N})={verific(N)}. En effet cle({code_sans_cle})={cle(code_sans_cle)}")
```

```
verific(2100574223)=False. En effet cle(210057422)=1
verific(2729812563)=True. En effet cle(272981256)=3
```

Exercice Bonus 6.24 (CheckSum)

Pour réduire la probabilité d'erreurs de transmission d'un code, on introduit un chiffre final (clé) calculé à partir des chiffres précédents. Écrire une fonction `cheksum(N)` qui, pour N donné, renvoie la clé. Les étapes à suivre pour obtenir la clé sont les suivantes :

- notons N le code et c la clé
- soit L la liste des chiffres de N dans l'ordre inverse
- pour les chiffres en position paire, doubler la valeur et en calculer la somme de ses chiffres
- calculer r le reste de la division par 10 de la somme des éléments de L
- si le reste est 0, poser $c=0$, sinon poser $c=10-r$

Exemple : cheksum(992739871) renvoie 3.

© G. Faccanoni

Source : https://py.checkio.org/en/mission/check-digit/

Bonus : généraliser au cas d'entrées alphanumériques (combinaison possible de 10 chiffres et 26 lettres majuscules). Pour cela, utiliser la valeur ASCII de chaque caractère. Par exemple : «A» a une valeur ASCII 65 obtenue par `ord("A")`. Pour déterminer sa séquence dans notre cas on devra alors soustraire 48.

Correction

```
def cheksum(N):
    L = [ int(c) for c in str(N)[::-1] ]
    M = [ ell if i%2==1 else sum( int(x) for x in str(ell*2)) for i,ell in enumerate(L) ]
    r = sum(M)%10
    return 0 if r==0 else 10-r # idem que (10-(r%10))%10

for N in (7992739871,123,61820923155) :
    print( cheksum(N), end=", " )
```

```
3, 0, 3,
```

Bonus

```
def cheksum(s):
    L = [ ord(c)-48 for c in s[::-1] if c.isnumeric() or c.isalpha() ]
    M = [ ell if i%2==1 else sum( int(x) for x in str(ell*2)) for i,ell in enumerate(L) ]
    r = sum(M)%10
    return 0 if r==0 else 10-r

for s in ("7992739871","123","61820923155","799 273 9871","139-MT","+61 820 9231 55"):
    print( cheksum(s), end=", " )
```

```
3, 0, 3, 3, 8, 3,
```

Exercice 6.25 (Algorithme d'Euclide)

Coder les deux algorithmes suivantes et les valider (voir aussi l'exercice 4.22).

- Algorithme des soustractions successives pour le calcul du **reste de la division euclidienne** de deux entiers :

```
Initialiser a,b in N, b non nul
Répéter tant que a>=b
    a ← a-b
fin répéter
Le reste vaut a
```

  Autrement dit, `r=a%b` (mais vous devez coder cet algorithme et ne pas utiliser l'opérateur `%`).

- Algorithme d'Euclide pour le calcul du **PGCD** de deux entiers :

```
Initialiser a,b in N et a>=b>0
Répéter tant que r>0
    r ← reste de la division euclidienne de a par b (avec l algorithme précédent)
    a ← b
    b ← r
fin répéter
Le PGCD vaut a
```

- Variante (Jacob Hacks, 1893)

$$\gcd(a,b) = a + b - ab + 2\sum_{i=1}^{b-1}\left\lfloor \frac{ai}{b} \right\rfloor$$

Correction

```
def reste(a,b):
    while a>=b:
        a -= b
    return a
```

```
def PGCD(a,b):
    b,a = sorted([a,b]) # ainsi a≥b
    while b>0:
        a,b = b,reste(a,b)
    return a

# TESTS
for a,b in [ (2*3,3*5) , (2*3,5*7) , (2*2,2*2*3) ]:
    print(f"PGCD({a},{b})={PGCD(a,b)}")
```

```
PGCD(6,15)=3
PGCD(6,35)=1
PGCD(4,12)=4
```

Variante

```
for a,b in [(2*3,3*5),(2*3,5*7),(2*2,2*2*3)]:
    print(f"PGCD({a},{b})={  a+b-a*b+2*sum( int(a*i/b) for i in range(1,b)) }")
```

```
PGCD(6,15)=3
PGCD(6,35)=1
PGCD(4,12)=4
```

Pour vérifier, on peut aussi s'appuyer sur la fonction gcd du module `math` :

```
from math import gcd
for a,b in [(2*3,3*5),(2*3,5*7),(2*2,2*2*3)]:
    print(f"PGCD({a},{b})={ gcd(a,b) }")
```

```
PGCD(6,15)=3
PGCD(6,35)=1
PGCD(4,12)=4
```

Exercice 6.26 (Indicatrice d'Euler)

L'indicatrice d'Euler est une fonction qui, à tout entier naturel n non nul, associe le nombre d'entiers compris entre 1 et n (inclus) qui sont premiers avec n :

$$\begin{aligned}\varphi\colon \mathbb{N}^* &\to \mathbb{N}^*\\ n &\mapsto \operatorname{card}\left\{ m\in\mathbb{N}^* \mid m\le n \text{ et } \gcd(m,n)=1 \right\}.\end{aligned}$$

Elle est aussi appelée fonction phi d'Euler ou simplement fonction phi, car la lettre φ (ou ϕ) est communément utilisée pour la désigner.

Exemples :

- $\varphi(8)=4$ car parmi les nombres de 1 à 8, seuls les quatre nombres 1, 3, 5 et 7 sont premiers avec 8 ;
- $\varphi(12)=4$ car parmi les nombres de 1 à 12, seuls les quatre nombres 1, 5, 7 et 11 sont premiers avec 12 ;
- $\varphi(1)=1$ car 1 est premier avec lui-même (c'est le seul entier naturel qui vérifie cette propriété).

Notons qu'un entier $p>1$ est premier si et seulement si tous les nombres de 1 à $p-1$ sont premiers avec p, c.-à-d. si et seulement si $\varphi(p)=p-1$.

Écrire une fonction qui, pour $n\in\mathbb{N}$, renvoie $\varphi(n)$.

Correction

```
from math import gcd
phi = lambda n: sum(gcd(n, k) == 1 for k in range(1, n+1))
print(*[f'phi({n}) = {phi(n)}' for n in range(1, 13)], sep=", ", end=". ")
```

```
phi(1) = 1, phi(2) = 1, phi(3) = 2, phi(4) = 2, phi(5) = 4, phi(6) = 2, phi(7) = 6, phi(8) =
↪ 4, phi(9) = 6, phi(10) = 4, phi(11) = 10, phi(12) = 4.
```

 © G. Faccanoni

Exercice 6.27 (Prix d'un billet)

Voici la réduction pour le prix d'un billet de train en fonction de l'âge du voyageur :

- réduction de 50% pour les moins de 10 ans ;
- réduction de 30% pour les 10 à 18 ans ;
- réduction de 20% pour les 60 ans et plus.

Écris une fonction qui renvoie la réduction en fonction de l'âge et dont les propriétés sont rappelées ci-dessous :

- Nom : `reduction()`
- Usage : `reduction(age)`
- Entrée : un entier correspondant à l'âge
- Sortie : un entier correspondant à la réduction
- Exemples : `reduction(17)` renvoie 30 ; `reduction(23)` renvoie 0

Écris une fonction qui calcule le montant à payer en fonction du tarif normal et de l'âge du voyageur :

- Nom : `montant()`
- Usage : `montant(tarif_normal,age)`
- Entrée : un nombre `tarif_normal` correspondant au prix sans réduction et `age` (un entier)
- Sortie : un nombre correspondant au montant à payer après réduction
- Remarque : utilise la fonction `reduction()`
- Exemples : `montant(100,17)` renvoie 70.

Considérons une famille qui achète des billets pour différents trajets, voici le tarif normal de chaque trajet et les âges des voyageurs :

- tarif normal 30 euros, enfant de 9 ans ;
- tarif normal 20 euros, pour chacun des jumeaux de 16 ans ;
- tarif normal 35 euros, pour chacun des parents de 40 ans.

Quel est le montant total payé par la famille ?

Correction

Rappel : une réduction de x% d'un prix p signifie payer $p \times (100-x)/100$.

```python
def reduction(age):
    if age<=10:
        return 50
    elif age<=18:
        return 30
    elif age>=60:
        return 20
    else:
        return 0

def montant(tarif_normal,age):
    coeff=100-reduction(age)
    return tarif_normal*coeff/100

# TEST
Total = montant(30,9) + 2*montant(20,16) + 2*montant(35,40)
print(f'Montant total payé par la famille : {Total} euros')
```

```
Montant total payé par la famille : 113.0 euros
```

En version compacte :

```python
#reduction = lambda age : 50 if (age<=10) else ( 30 if (age<=18) else ( 20 if (age>=60) else
↪   0 ) )
reduction = lambda age : 50*(age<=10)+30*(10<age<=18)+20*(age>=60)
montant   = lambda tarif_normal,age : tarif_normal*(100-reduction(age))/100
```

```
# TEST
Total = montant(30,9) + 2*montant(20,16) + 2*montant(35,40)
print(f'Montant total payé par la famille : {Total} euros')
```

```
Montant total payé par la famille : 113.0 euros
```

Exercice 6.28 (Rendu monnaie)

La fonction `rendu_monnaie` prend en paramètres deux nombres entiers positifs `somme_due` et `somme_versee` et elle permet de procéder au rendu de monnaie de la différence `somme_versee - somme_due` pour des achats effectués avec le système de pièces de la zone Euro. Toutes les sommes sont exprimées en euros. Les valeurs possibles pour les pièces sont donc $[1,2,5,10,20,50,100,200]$. On utilisera le nombre minimum de pièces.

On utilise pour cela un algorithme glouton qui commence par rendre le maximum de pièces de plus grandes valeurs et ainsi de suite. Par la suite, on assimilera les billets à des pièces.

La fonction `rendu_monnaie` renvoie une liste contenant les pièces qui composent le rendu. Ainsi, l'instruction `rendu_monnaie(452, 500)` renvoie la liste des pièeces/billets à rendre $[20,20,5,2,1]$. En effet, la somme à rendre est de $500-452=48$ euros soit $20+20+5+2+1$.

Correction
Exercice 13.2 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

```
def rendu_monnaie(due, versee):
    montant_a_rendre = versee - due
    coupures = [1, 2, 5, 10, 20, 50, 100, 200]
    pieces_rendues = []
    for c in coupures[::-1]:
        q,montant_a_rendre = divmod(montant_a_rendre,c)
        pieces_rendues.extend( [c]*q )
    return pieces_rendues

due, versee = 452, 500
pieces_rendues = rendu_monnaie(due,versee)
print(f"{due = }, {versee = }, {pieces_rendues = }")
```

```
due = 452, versee = 500, pieces_rendues = [20, 20, 5, 2, 1]
```

Exercice 41.2 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

Version recursive :

```
coupures = [200, 100, 50, 20, 5, 2, 1]
def rendu_monnaie(montant_a_rendre, rang):
    if montant_a_rendre == 0:
        return []
    coin = coupures[rang]
    if  coin <= montant_a_rendre :
        return [coin] + rendu_monnaie(montant_a_rendre-coin, rang)
    else:
        return rendu_monnaie(montant_a_rendre, rang+1)

due, versee = 452, 500
pieces_rendues = rendu_monnaie(versee-due,0)
print(f"{due = }, {versee = }, {pieces_rendues = }")
```

```
due = 452, versee = 500, pieces_rendues = [20, 20, 5, 2, 1]
```

Exercice Bonus 6.29 (Pydéfis – Monnaie)

Dans une monnaie imaginaire, vous disposez d'un nombre illimité de pièces de valeur 50, 20, 10, 5, 2 et 1. Vous devez utiliser ces pièces pour rembourser des sommes dues à plusieurs personnes différentes. Pour chaque personne, vous

utiliserez le nombre minimum de pièces. L'entrée du problème est constituée de la liste des sommes à rembourser. Vous devez répondre en fournissant une liste de 6 entiers indiquant le nombre de pièces de 50, 20, 10, 5, 2 et 1 à utiliser. Par exemple, si les sommes à payer sont $(43, 32, 21)$, vous devez répondre $(0, 4, 1, 0, 2, 2)$.

Si les sommes à payer sont $(77, 94, 80, 67, 37, 53, 61, 53, 59, 3, 92, 17, 44, 11, 13, 75, 93, 98, 91, 9)$, quelles pièces de monnaie doit-on utiliser ?

Source : https://pydefis.callicode.fr/defis/Monnaie/t

Exercice 6.30 (Can Balance)

Soit L une liste de nombre. On cherche l'indice de l'élément pivot définit comme suit : le produit scalaire des éléments de la sous-liste à gauche par leur distance à l'élément pivot est égale au produit scalaire des éléments de la sous-liste à droite par leur distance à l'élément pivot. Voici deux exemples :

1. Si $L = [6, 1, 10, 5, 4]$, l'élément pivot est en position $n = 2$. En effet,
   - sous-liste gauche : $L[: n] = [6, 1]$; distances : $[2, 1]$; produit scalaire : $6 \times 2 + 1 \times 1 = 13$;
   - sous-liste droite : $L[n + 1 :] = [5, 4]$; distances : $[1, 2]$; produit scalaire : $5 \times 1 + 4 \times 2 = 13$.

6	1	10	5	4
↑	↑		↑	↑
2	1		1	2

2. Si $L = [10, 3, 3, 2, 1]$, l'élément pivot est en position $n = 1$. En effet,
   - sous-liste gauche : $L[: n] = [10]$; distances : $[1]$; produit scalaire : $10 \times 1 = 10$;
   - sous-liste droite : $L[n + 1 :] = [3, 2, 1]$; distances : $[1, 2, 3]$; produit scalaire : $3 \times 1 + 2 \times 2 + 1 \times 3 = 10$.

10	3	3	2	1
↑		↑	↑	↑
1		1	2	3

Correction
https://py.checkio.org/en/mission/can-balance/

Soit p l'indice du pivot et $\ell_i = L[i]$. La distance de l'élément ℓ_i de l'élément pivot ℓ_p est $|i - p|$. Par définition de pivot on a alors

$$\sum_{i=0} (i - p)\ell_i = 0$$

ainsi

$$p = \frac{\sum_{i=0} i\ell_i}{\sum_{i=0} \ell_i}.$$

```python
def can_balance(L):
    p = sum(i*ell for i, ell in enumerate(L)) / sum(L)
    return int(p) if int(p) == p else -1

print(can_balance([6, 1, 10, 5, 4])) # == 2
print(can_balance([6, 1, 10, 5, 4])) # == 2
print(can_balance([10, 3, 3, 2, 1])) # == 1
print(can_balance([7, 3, 4, 2, 9, 7, 4])) # == -1
print(can_balance([42])) # == 0
```

```
2
2
1
-1
0
```

Exercice Bonus 6.31 (Matching Brackets)

Soient `chaine` une chaîne de caractères qui contient des parenthèses (,), des crochets [,], et des guillemets {, }. Compléter la fonction `is_paired(chaine)` qui renvoie `True` si les parenthèse dans `chaine` sont équilibrées, `False` sinon.

Exemple :

- `is_paired( "[]" )` renvoie `True`
- `is_paired( "[)" )` renvoie `False`
- `is_paired( "[()]" )` renvoie `True`
- `is_paired( "[(])" )` renvoie `False`

Correction

```
def is_paired(chaine):
    stack = []
    for c in chaine:
        if c in "({[":
            stack.append(c)
        elif c==")" :
            if stack[-1]=="(" and len(stack) > 0 :
                stack = stack[:-1]
            else:
                return False
        elif c=="]" :
            if stack[-1]=="[" and len(stack) > 0 :
                stack = stack[:-1]
            else:
                return False
        elif c=="}" :
            if stack[-1]=="{" and len(stack) > 0 :
                stack = stack[:-1]
            else:
                return False
    return len(stack) == 0

# TESTS
print( is_paired("{[2*(7-6)+3*(4-5)]/8}") )
print( is_paired("[2+3*(4-5])") )
```

```
True
False
```

Exercice 6.32 (Liste impairs)

Écrire une fonction lambda qui prend en entrée L, R avec L < R et renvoi la liste des nombre impairs entre L et R (inclus).

Correction

Si on se donne L et R on pourra écrire par exemple

```
L,R = 1,15
impaires = [i for i in range(L,R+1) if i%2!=0]
print(impaires)
```

```
[1, 3, 5, 7, 9, 11, 13, 15]
```

Transformons ce raisonnement en une fonction :

```
impaires = lambda L,R : [i for i in range(L,R+1) if i%2!=0]
print(impaires(1,15))
```

```
[1, 3, 5, 7, 9, 11, 13, 15]
```

 © G. Faccanoni

Exercice 6.33 (Liste divisibles)

Écrire une fonction lambda qui prend en entrée L, R, n avec L < R et renvoi le liste des nombre entre L et R (inclus) qui sont divisibles par n.

Correction

```
>>> divisibles = lambda L,R,n : [i for i in range(L,R+1) if i%n==0]
>>> print(divisibles(50,100,7))
[56, 63, 70, 77, 84, 91, 98]
```

Exercice 6.34 (Nombre miroir)

On appellera “miroir d'un nombre n” le nombre n écrit de droite à gauche. Par exemple, `miroir(7423) = 3247`. Écrire une fonction qui prend en entrée n et renvoi son miroir (*cf.* exercice 4.57).

Bonus : soit x_1 un nombre à trois chiffres, x_2 son miroir, et supposons que x_1 et x_2 soient des carrés parfaits, autrement dit il existe y_1 et y_2 (à deux chiffres) tels que $y_1^2 = x_1$ et $y_2^2 = x_2$. Afficher tous les nombres x_1 tels que y_2 est le miroir de y_1.

Correction

```
miroir = lambda n : int(str(n)[::-1])
print(miroir(7423))
```

```
3247
```

Partie “bonus” : on construit les nombres à trois chiffres et on vérifie qu'ils sont des carrés parfaits en vérifiant si la racine carrée coïncide avec sa partie entière et si les deux racines sont l'une le miroir de l'autre :

```
for a in range(1,10):
	for b in range(10):
		for c in range(1,10):
			abc = 100*a+10*b+c
			cba = miroir(abc)
			sq_abc = abc**0.5
			sq_cba = cba**0.5
			if sq_abc==int(sq_abc) and sq_cba==int(sq_cba) and sq_abc==miroir(int(sq_cba)):
				print(f"{abc} car {abc}={int(sq_abc)}^2 et {cba}={int(sq_cba)}^2")
```

```
121 car 121=11^2 et 121=11^2
144 car 144=12^2 et 441=21^2
169 car 169=13^2 et 961=31^2
441 car 441=21^2 et 144=12^2
484 car 484=22^2 et 484=22^2
961 car 961=31^2 et 169=13^2
```

Indiquons par $\underline{abc}$ l'écriture décimale du nombre $100a+10b+c$. On remarque qu'un nombre à trois chiffres $\underline{abc}$ satisfait la condition ssi il existe un nombre à deux chiffres $\underline{de}$ tel que $\underline{de}^2 = \underline{abc}$ et $\underline{ed}^2 = \underline{cba}$. Ceci implique les conditions suivantes :

$$\begin{cases}(10d+e)^2 = 100a+10b+c,\\(10e+d)^2 = 100c+10b+a,\end{cases}$$

$$\implies (10d+e)^2-(10e+d)^2 = 99(a-b) \implies \Big((10d+e)+(10e+d)\Big)\Big((10d+e)-(10e+d)\Big) = 99(a-b)$$

$$\implies \Big(11(d+e)\Big)\Big(9(d+e)\Big) = 99(a-b) \implies 99(d^2-e^2) = 99(a-b) \implies d^2-e^2 = a-b$$

Exercice 6.35 (Point milieu)

On représente un point P du plan de coordonnées (P_x, P_y) par la liste `P=[Px,Py]`. Écrire une fonction qui renvoie le point milieu du segment d'extrémités les points P et Q.

Correction
Le point milieu a pour coordonnées $\left(\frac{P_x+Q_x}{2}, \frac{P_y+Q_y}{2}\right)$.

```
milieu = lambda P,Q : [ (p+q)/2 for p,q in zip(P,Q) ]

P = [0,0] ; Q = [2,2]
print(milieu(P,Q))
```

```
[1.0, 1.0]
```

Exercice 6.36 (Normaliser une liste)

La normalisation d'un ensemble de valeurs est une opération importante en mathématiques, consistant à transformer de manière affine un ensemble de valeurs dans un intervalle $[v_{\min}; v_{\max}]$ en un ensemble de valeurs dans $[0;1]$.

Générez une liste de valeurs aléatoires de cette façon :

```
import random
L=[random.randint(10,100) for i in range(10)]
```

À l'aide d'une fonction lambda et d'une liste de compréhension, écrire le code permettant de normaliser la liste, c'est à dire rapporter toutes ses valeurs entre 0 et 1 de manière affine, Par exemple la liste $[30,60,75,130,40,100]$ sera transformée dans la liste $0,0.3,0.45,1,0.1,0.7$.

Pour vérifier que votre code fonctionne, vérifiez bien que, dans votre matrice normalisée, vous avez au moins un 0 et un 1.

Aide : calculer au préalable l'équation de la droite qui interpole les deux points $(v_{min},0)$ et $(v_{max},1)$.

Correction
Soit $x \in [v_{\min}; v_{\max}]$ et soit $y \in [0;1]$. On cherche un changement de variable affine, *i.e.* une fonction $g : [v_{\min}; v_{\max}] \to [0;1]$ définie par $g(x) = mx + q$, qui envoie l'intervalle $[v_{\min}; v_{\max}]$ dans l'intervalle $[0;1]$, c'est-à-dire telle que

$$\begin{cases} g(v_{\min}) = 0, \\ g(v_{\max}) = 1. \end{cases}$$

Il s'agit simplement de l'équation de la droite qui interpole les deux points $(v_{\min},0)$ et $(v_{\max},1)$:

$$g(x) = \frac{1}{v_{\max} - v_{\min}}(x - v_{\min})$$

```
import random
L = [random.randint(10,100) for i in range(10)]
print(f"L={L}")
v_min = min(L)
v_max = max(L)
g = lambda x : (x-v_min)/(v_max-v_min)
G = [g(x) for x in L]
print(f"G={G}")
```

```
L=[57, 76, 16, 54, 30, 66, 32, 72, 41, 18]
G=[0.6833333333333333, 1.0, 0.0, 0.6333333333333333, 0.23333333333333334, 0.8333333333333334,
↪ 0.26666666666666666, 0.9333333333333333, 0.4166666666666667, 0.03333333333333333]
```

Exercice 6.37 (Couper un intervalle)

Soit $[a;b]$ un intervalle représenté sous la forme d'une liste : `I=[a,b]`.

- Écrire une fonction `Demi(I)` qui scinde l'intervalle $[a;b]$ en deux intervalles $[a;c]$ et $[c;b]$ de même longueur (que vaut c ?) et qui renvoie ces deux intervalles.
  Par exemple, `Demi([0,3])` donnera `[[0, 1.5], [1.5, 3]]`.

- Écrire une fonction `Tiers(I)` qui scinde l'intervalle $[a;b]$ en trois intervalles $[a;c_1]$, $[c_1;c_2]$ et $[c_2;b]$ de même longueur (que valent c_1 et c_2 ?) et qui renvoie ces trois intervalles.
  Par exemple, `Tiers([0,3])` donnera `[[0, 1.0], [1.0, 2.0], [2.0, 3]]`.
- Écrire une fonction `Couper(I,n)` qui scinde l'intervalle $[a;b]$ en n intervalles de même longueur (que vaut cette longueur ?) et qui renvoie ces n intervalles.

Correction

On cherche c tel que $b-c=a-c$ donc $c=\frac{a+b}{2}=a+\frac{b-a}{2}$. Posons $h=\frac{b-a}{2}$, alors $c=a+h$.

```
def Demi(I):
	a = I[0]
	b = I[1]
	c = (a+b)/2
	return [[a,c],[c,b]]

# Test
print(Demi([0,3]))
```

```
[[0, 1.5], [1.5, 3]]
```

On cherche c_1 et c_2 tels que $b-c_2=a-c_1=c_2-c_1$. Posons $h=\frac{b-a}{3}$, alors $c_1=a+h$ et $c_2=a+2h$.

```
def Tiers(I):
	a = I[0]
	b = I[1]
	h = (b-a)/3
	c1 = a+h
	c2 = a+2*h
	return [[a,c1],[c1,c2],[c2,b]]

# Test
print(Tiers([0,3]))
```

```
[[0, 1.0], [1.0, 2.0], [2.0, 3]]
```

Posons $h=\frac{b-a}{n}$ ainsi $a=a+0h$ et $b=a+nh$. Alors

```
def Couper(I,n):
	a = I[0]
	b = I[1]
	h = (b-a)/n
	return [[a+i*h,a+(i+1)*h] for i in range(n)]

# Test
print(Couper([0,3],2))
print(Couper([0,3],3))
print(Couper([0,3],6))
```

```
[[0.0, 1.5], [1.5, 3.0]]
[[0.0, 1.0], [1.0, 2.0], [2.0, 3.0]]
[[0.0, 0.5], [0.5, 1.0], [1.0, 1.5], [1.5, 2.0], [2.0, 2.5], [2.5, 3.0]]
```

Exercice 6.38 (Représentation et manipulation de polynômes)

Soit $\mathbb{R}_n[x]$ l'ensemble des polynômes de degré inférieur ou égale à n, $n\in\mathbb{N}$. Tout polynôme de cet espace vectoriel s'écrit de manière unique comme

$$p_n(x)=\sum_{i=0}^{n}a_ix^i=a_0+a_1x+\cdots+a_nx^n,\qquad \text{où } a_i\in\mathbb{R} \text{ pour } i=0,\dots n.$$

On peut représenter un polynôme par la liste de ses coefficients et stocker ces $n+1$ valeurs réels $a_0,a_1,\dots,a_n$ dans

une liste : [a]

$$\mathbf{p} = \text{coord}(p_n, \mathscr{C}_n) = [a_0, a_1, a_2, \dots, a_n] \in \mathbb{R}^{n+1}$$

(le premier élément de la liste étant le coefficient du terme de degré zéro).

Nous utiliserons la liste **p** pour manipuler un polynôme et nous construirons des fonctions pour opérer sur les polynômes à partir de cette représentation. Par exemple, pour construire le polynôme $p_2(x) = 2 - x + x^2$ nous écrirons `p = [2, -1, 1]` et le degré du polynôme est tout simplement `n=len(p)-1`.

1. Implémenter une fonction appelée `eval_pol(p,x)` permettant d'évaluer le polynôme p (la fonction polynomiale) en x où x est une valeur numérique ou une liste. Dans le second cas on doit obtenir une liste contenant les valeurs de la fonction polynomiale aux différents points spécifiés dans la liste **x**. Par exemple, pour évaluer le polynôme $p(x) = 1 + 2x + 3x^2$ en $\mathbf{x} = [-1, 0, 1, 2]$ nous écrirons
```
p = [1 2 3]
y = eval_pol(p,[-1,0,1,2])
```
et on veut obtenir la liste $\mathbf{y} = p(\mathbf{x}) = [2, 1, 6, 17]$. En effet on a
$$\begin{aligned} p(x) &= 1 + 2x + 3x^2 \\ \mathbf{p} &= \text{coord}[p, \mathscr{C}_2) = (1, 2, 3] \end{aligned} \qquad \begin{aligned} p(-1) &= 1 + 2 \times (-1) + 3 \times ((-1)^2) = 1 - 2 + 3 = 2 \\ p(0) &= 1 + 2 \times 0 + 3 \times (0^2) = 1 + 0 + 0 = 1 \\ p(1) &= 1 + 2 \times 1 + 3 \times (1^2) = 1 + 2 + 3 = 6 \\ p(2) &= 1 + 2 \times 2 + 3 \times (2^2) = 1 + 4 + 12 = 17 \end{aligned}$$

2. Implémenter une fonction appelée `sum_pol` renvoyant la somme de deux polynômes (attention, si les deux polynômes n'ont pas même degré, il faudra ajouter des zéros en fin du polynôme de plus petit degré afin de pouvoir calculer l'addition des deux vecteurs représentatifs). Par exemple, pour $\mathbf{p} = [1, 2, 3]$ et $\mathbf{q} = [1, -2]$, on veut obtenir $\mathbf{s} = [2, 0, 3]$:
$$\begin{aligned} p(x) &= 1 + 2x + 3x^2 & \mathbf{p} &= \text{coord}(p, \mathscr{C}_2) = [1, 2, 3] \\ q(x) &= 1 - 2x & \mathbf{q} &= \text{coord}(q, \mathscr{C}_1) = [1, -2] \implies \mathbf{q} = \text{coord}(q, \mathscr{C}_2) = [1, -2, 0] \\ s(x) = p(x) + q(x) &= 2 + 3x^2 & \mathbf{s} &= \text{coord}(p + q, \mathscr{C}_2) = [2, 0, 3] \end{aligned}$$

3. Implémenter une fonction appelée `prod_pol` renvoyant le produit de deux polynômes.
Exemple, pour $\mathbf{p} = [1, 0, 3]$ et $\mathbf{q} = [1, -2]$, on veut obtenir $\mathbf{u} = [1, -2, 3, -6]$.
$$\begin{aligned} p(x) &= 1 + 3x^2 & \mathbf{p} &= \text{coord}(p, \mathscr{C}_2) = [1, 0, 3] \\ q(x) &= 1 - 2x & \mathbf{q} &= \text{coord}(q, \mathscr{C}_2) = [1, -2, 0] \\ u(x) = p(x) \times q(x) &= 1 \times p(x) - 2x \times p(x) = 1 - 2x + 3x^2 - 6x^3 & \mathbf{u} &= \text{coord}(p \times q, \mathscr{C}_3) = [1, -2, 3, -6] \end{aligned}$$

4. Implémenter une fonction appelée `derivee_pol` renvoyant la dérivée d du polynôme p donné en entrée (attention, si $p \in \mathbb{R}_n[x]$, alors $d \in \mathbb{R}_{n-1}[x]$).
Exemple, pour $\mathbf{p} = [1, 2, 6]$, on veut obtenir $\mathbf{d} = [2, 12]$.
$$\begin{aligned} p(x) &= 1 + 2x + 6x^2 & \mathbf{p} &= \text{coord}(p, \mathscr{C}_2) = [1, 2, 6] \\ d(x) &= p'(x) = 2 + 12x & \mathbf{d} &= \text{coord}(d, \mathscr{C}_1) = [2, 12] \end{aligned}$$

5. Implémenter une fonction appelée `primitive_pol` renvoyant la primitive v du polynôme p donné en entrée ayant 0 pour racine (attention, si $p \in \mathbb{R}_n[x]$, alors $v \in \mathbb{R}_{n+1}[x]$).
Exemple, pour $\mathbf{p} = [1, 2, 6]$, on veut obtenir $\mathbf{v} = [0, 1, 1, 2]$.
$$\begin{aligned} p(x) &= 1 + 2x + 6x^2 & \mathbf{p} &= \text{coord}(p, \mathscr{C}_2) = [1, 2, 6] \\ v(x) &= \int_0^x p(t)\,\mathrm{d}t = \int_0^x 1 + 2t + 6t^2\,\mathrm{d}t = x + x^2 + 2x^3 & \mathbf{v} &= \text{coord}(v, \mathscr{C}_3) = [0, 1, 1, 2] \end{aligned}$$

6. Implémenter une fonction appelée `integrale_pol` renvoyant l'intégrale d'un polynôme entre deux valeurs a et b.

Exemple, pour $\mathbf{p} = [1,2,6]$, $a = 1$ et $b = 2$, on veut obtenir $c = 18$:

$$p(x) = 1 + 2x + 6x^2$$
$$c = \int_a^b p(t)\,\mathrm{d}t = \int_0^b p(t)\,\mathrm{d}t - \int_0^a p(t)\,\mathrm{d}t = v(b) - v(a) = b + b^2 + 2b^3 - a - a^2 - 2a^3 = 18.$$

a. Il s'agit des coordonnées de p_n dans la base canonique de l'espace vectoriel $\mathbb{R}_n[x]$, *i.e.* l'ensemble $\mathscr{C}_n = \{1, x, x^2, \dots, x^n\}$

Correction

1.
```
def eval_pol(p,x):
    f = lambda xi,pol : sum( pol[k]*xi**k for k in range(len(pol)) )
    return [ f(xi,p) for xi in x  ]
    
p = [1, 2, 3]
x = [-1,0,1,2]
y = eval_pol(p,x)
print( f"{x=}, {y=}" )
```

```
x=[-1, 0, 1, 2], y=[2, 1, 6, 17]
```

2. Sans perte de généralité, supposons que $n > m$, alors

$$p(x) = \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{m} a_i x^i + \sum_{i=m+1}^{n} a_i x^i \qquad \operatorname{coord}(p, \mathscr{C}) = [a_0, a_1, a_2, \dots, a_m, a_{m+1}, \dots, a_n]$$
$$q(x) = \sum_{i=0}^{m} b_i x^i = \sum_{i=0}^{m} b_i x^i + \sum_{i=m+1}^{n} 0 \times x^i \qquad \operatorname{coord}(q, \mathscr{C}) = [b_0, b_1, b_2, \dots, b_m]$$
$$(p+q)(x) = \sum_{i=0}^{m} (a_i + b_i) x^i + \sum_{i=m+1}^{n} a_i x^i \qquad \operatorname{coord}(p+q, \mathscr{C}) = [a_0 + b_0, a_1 + b_1, a_2 + b_2, \dots, a_m + b_m, a_{m+1}, \dots, a_n]$$

```
def sum_pol(p,q):
    S = [0 for _ in range(max(len(p),len(q)))]
    for i in range(len(p)):
        S[i] += p[i]
    for i in range(len(q)):
        S[i] += q[i]
    return S

p, q = [1, 2, 3] , [1,-2]
print( f"{p=}, {q=}, s={sum_pol(p,q)}" )
```

```
p=[1, 2, 3], q=[1, -2], s=[2, 0, 3]
```

3. prod_pol renvoyant le produit de deux polynômes.

```
def prod_pol(p,q):
  # Initialiser le résultat du produit à une liste de zéros de la longueur du produit des
  ↪ degrés des deux polynômes
  result = [0 for _ in range(len(p) + len(q))]
  # Pour chaque coefficient du premier polynôme
  for i in range(len(p)):
    # Pour chaque coefficient du second polynôme
    for j in range(len(q)):
      # Ajouter le produit des coefficients à la position correspondante dans le
      ↪ résultat
      result[i+j] += p[i] * q[j]
  return result

p, q = [1, 0, 3] , [1, -2]
print( f"{p=}, {q=}, s={prod_pol(p,q)}" )
```

```
p=[1, 0, 3], q=[1, -2], s=[1, -2, 3, -6, 0]
```

4. Remarquons que

$$p(x) = \sum_{i=0}^{n} a_i x^i \qquad d(x) = p'(x) = \sum_{i=0}^{n} i a_i x^{i-1}$$

$$\text{coord}(p, \mathscr{C}_n) = [a_0, a_1, a_2, \dots, a_n] \qquad \text{coord}(d, \mathscr{C}_{n-1}) = [a_1, 2a_2, \dots, na_n]$$

```
def derivee_pol(p):
    n = len(p)
    return [ i*p[i] for i in range(1,n)  ]

p=[1];       print(f"{p=}, d={derivee_pol(p)}")
p=[1,2];     print(f"{p=}, d={derivee_pol(p)}")
p=[1,2,6];   print(f"{p=}, d={derivee_pol(p)}")
p=[4,5,0,2]; print(f"{p=}, d={derivee_pol(p)}")
```

```
p=[1], d=[]
p=[1, 2], d=[2]
p=[1, 2, 6], d=[2, 12]
p=[4, 5, 0, 2], d=[5, 0, 6]
```

5. Remarquons que

$$p(x) = \sum_{i=0}^{n} a_i x^i \qquad v(x) = \int_0^x p(t)\,\mathrm{d}t = \sum_{i=0}^{n} a_i \int_0^x t^i\,\mathrm{d}t = \sum_{i=0}^{n} a_i \frac{x^{i+1}}{i+1}$$

$$\text{coord}(p, \mathscr{C}_n) = [a_0, a_1, a_2, \dots, a_n] \qquad \text{coord}(v, \mathscr{C}_{n+1}) = \left[0, \frac{a_0}{0+1}, \frac{a_1}{1+1}, \frac{a_2}{2+1}, \dots, \frac{a_n}{n+1}\right]$$

```
def primitive_pol(p):
    n = len(p)
    return [0] + [ p[i]/(i+1) for i in range(n)  ]

p=[1];       print(f"{p=}, d={primitive_pol(p)}")
p=[1,2];     print(f"{p=}, d={primitive_pol(p)}")
p=[1,2,6];   print(f"{p=}, d={primitive_pol(p)}")
p=[1,2,1,1]; print(f"{p=}, d={primitive_pol(p)}")
```

```
p=[1], d=[0, 1.0]
p=[1, 2], d=[0, 1.0, 1.0]
p=[1, 2, 6], d=[0, 1.0, 1.0, 2.0]
p=[1, 2, 1, 1], d=[0, 1.0, 1.0, 0.3333333333333333, 0.25]
```

6.
```
def integrale_pol(p,a,b):
    prim = primitive_pol(p)
    L = eval_pol(prim,[a,b])
    return L[1]-L[0]

p, a, b = [1,2,6] , 1, 2
print(f"{integrale_pol(p,a,b)=}")
```

```
integrale_pol(p,a,b)=18.0
```

 © G. Faccanoni

Exercice 6.39 (Triangle de Pascal)

On cherche à déterminer les valeurs du triangle de Pascal :

```
1
1  1
1  2   1
1  3   3   1   1
1  4   6   4   1
1  5  10  10   5  1
```

Dans ce triangle, chaque ligne commence et se termine par le nombre 1. De plus, on additionne deux valeurs successives d'une ligne pour obtenir la valeur qui se situe sous la deuxième valeur.

Compléter la fonction `pascal` prenant en paramètre un entier $n \geq 2$. Cette fonction doit renvoyer une liste correspondante au triangle de Pascal de la ligne 0 à la ligne n (incluse).

Correction

① Version du sujet 17.2 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

```python
def pascal(n):
    triangle = [[1]]
    for k in range(1, n+1):
        ligne_k = [1]
        for i in range(1, k):
            ligne_k.append(triangle[k-1][i-1] + triangle[k-1][i])
        ligne_k.append(1)
        triangle.append(ligne_k)
    return triangle

T = pascal(4)
# print(T)
# affichage amelioré
from tabulate import tabulate
print(tabulate(T,tablefmt='plain'))
```

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
```

② Sachant que $p_{ki} = \binom{k}{i}$ on peut écrire

```python
from math import comb
pascal = lambda n : [ [comb(k,i) for i in range(k+1)] for k in range(n+1)  ]
T = pascal(4)
from tabulate import tabulate
print(tabulate(T,tablefmt='plain'))
```

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
```

Exercice 6.40 (Voyelles)

Écrire une fonction qui prend en entrée une chaîne de caractères donnée et renvoie le nombre de voyelles.

Correction

```
VOYELLES_FR = "AaÀàEeÈèÉéÊêËëIiÎîÏïOoÖöUuÙùYy"

#def voyelles(s):
#	return len([x for x in s if x in VOYELLES_FR])

voyelles = lambda s : len([x for x in s if x in VOYELLES_FR])

# TESTS
for s in ["citron", "fraise", "aïoli", "pèche", "Supercalifragilisticexpidélilicieux" ]:
	print(f"{s} contient {voyelles(s)} voyelles")
```

```
citron contient 2 voyelles
fraise contient 3 voyelles
aïoli contient 4 voyelles
pèche contient 2 voyelles
Supercalifragilisticexpidélilicieux contient 16 voyelles
```

Exercice 6.41 (Pangramme)

Écrire une fonction qui prend en entrée une chaîne de caractères s donnée et renvoie `True` si elle contient toutes les lettres de l'alphabet (*i.e.* s est un pangramme [a]), `False` sinon.

On prendra comme alphabet `"abcdefghijklmnopqrstuvwxyz"`.

a. Un pangramme contient toutes les lettres de l'alphabet. Deux exemples classiques :
- "Portez ce vieux whisky au juge blond qui fume"
- "The quick brown fox jumps over the lazy dog"

Plus fort encore, ce pangramme de Gilles Esposito-Farèse, qui contient toutes les lettres accentuées et ligatures du français : "Dès Noël où un zéphyr haï me vêt de glaçons würmiens, je dîne d'exquis rôtis de bœuf au kir à l'aÿ d'âge mûr et cætera !"

Correction

```
alphabet="abcdefghijklmnopqrstuvwxyz"

def pangramme(s):
	for x in alphabet:
		if x not in s:
			print(f"Il manque la lettre {x}")
			return False
	return True

# TESTS
print(pangramme("portez ce vieux whisky au juge blond qui fume"))
print(pangramme("portez ce vieux whisky au juge blond qui boit"))
```

```
True
Il manque la lettre f
False
```

Exercice 6.42 (Swap)

Soit L une liste de nombres. Écrire une fonction `Swap(L,i,j)` qui échange les éléments d'indices `i` et `j` de la liste L. Par exemple, `Swap([0,1,2,10,80],1,4)=[0,80,2,10,1]`.

Correction

Puisque la liste est un objet mutable, il n'est pas nécessaire d'utiliser `return` :

```
def Swap(L,i,j):
	L[i],L[j] = L[j],L[i]
```

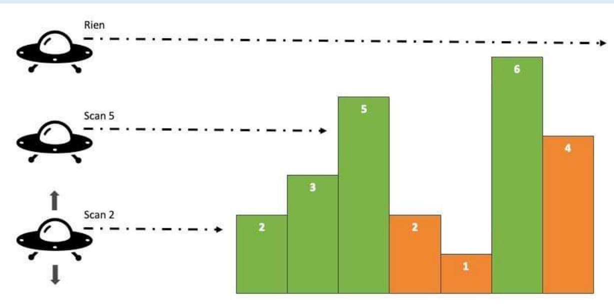
```python
L = [0,1,2,10,80]
print(L)
Swap(L,1,4)
print(L)
```

```
[0, 1, 2, 10, 80]
[0, 80, 2, 10, 1]
```

Exercice 6.43 (OVNI)

Un OVNI se déplace à la verticale et scanne horizontalement les immeubles devant lui. Connaissant les hauteurs d'immeubles, lesquels seront touchés par le scanner ?

Par exemple, si les immeubles ont pour hauteur 2, 3, 5, 2, 1, 6, 4, la réponse sera 2, 3, 5, 6 (voir la figure ci-dessous).



Source : https://twitter.com/riko_schraf/status/1554717715012673536

Correction

```python
def scan(L):
    S = [L[0]]
    for ell in L[1:]:
        if ell>S[-1]:
            S.append(ell)
    return S

# TESTS
for L in ( [2,3,5,2,1,6,4] , [3,1,0,2] ):
    print(f"{L = }, {scan(L) = }")
```

```
L = [2, 3, 5, 2, 1, 6, 4], scan(L) = [2, 3, 5, 6]
L = [3, 1, 0, 2], scan(L) = [3]
```

Exercice 6.44 (Nombres premiers)

Un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs : 1 et lui-même. Ainsi 1 n'est pas premier, mais 2 oui.

Écrire la liste des nombres premiers compris entre 1 et 100.

Remarques :

- Pour déterminer si un nombre n est premier, on va chercher s'il existe un nombre différent de 1 et de n qui le divise ;
- un diviseur de n (différent de 1 et de n) est forcement inférieur à n ;
- si n nest pas premier, il est divisible par un nombre inférieur ou égal à la racine entière de n.

Pour savoir si le nombre n est premier, il suffit alors de calculer les restes des divisions de n par 2, puis 3, puis 4... jusqu'à l'arrondi par défaut de $\sqrt{n}$. Si un des restes est nul, le nombre n'est pas premier. Si aucun reste n'est nul, le

nombre n est premier.

Correction

```
def isPrime(n):
    if n <= 3: return n>1
    if n%2 == 0 or n%3 == 0 : return False
    for i in range(3,int(n**0.5)+1,2):    # on s'arrête à √n
        if n%i==0: return False
    return True
print([i for i in range(2,101) if isPrime(i)])
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,
 97]
```

☆ Exercice Bonus 6.45 (Mot parfait)

On affecte à chaque lettre de l'alphabet un code selon la position dans l'alphabet : A $\rightsquigarrow$ 1, B $\rightsquigarrow$ 2, ..., Z $\rightsquigarrow$ 26. Pour un mot donné, on détermine d'une part son *code alphabétique concaténé*, obtenu par la juxtaposition des codes de chacun de ses caractères, et d'autre part, *son code additionné*, qui est la somme des codes de chacun de ses caractères.

Par ailleurs, on dit que ce mot est «*parfait*» si le code additionné divise le code concaténé.

Exemples :

- Pour le mot `"PAUL"`,
  - le code concaténé est la chaîne `"1612112"`, soit l'entier 1612112;
  - son code additionné est l'entier 50 car $16+1+21+12=50$;
  - le mot `"PAUL"` n'est pas parfait car 50 ne divise pas l'entier 1612112.
- Pour le mot `"ALAIN"`,
  - le code concaténé est la chaîne `"1121914"`, soit l'entier 1121914;
  - son code additionné est l'entier 37 car $1+12+1+9+14=37$;
  - le mot `"ALAIN"` est parfait car 37 divise l'entier 1121914.

Compléter la fonction `est_parfait` qui prend comme argument une chaîne de caractères `mot` (en lettres majuscules) et qui renvoie le code alphabétique concaténé, le code additionné de `mot`, ainsi qu'un booléen qui indique si `mot` est parfait ou pas.

Correction

Sujet 22.2 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

```
dico = { c:i+1 for i,c in enumerate("ABCDEFGHIJKLMNOPQRSTUVWXYZ") }

def est_parfait(mot):
    # mot est une chaîne de caractères (en lettres majuscules)
    code_concatene = ""
    code_additionne = 0
    for c in mot:
        code_concatene += str(dico[c])
        code_additionne += dico[c]
    code_concatene = int(code_concatene)
    if code_concatene % code_additionne == 0:
        mot_est_parfait = True
    else:
        mot_est_parfait = False
    return code_additionne, code_concatene, mot_est_parfait

for mot in ["PAUL","ALAIN"]:
    print(f"est_parfait({mot}) renvoie {est_parfait(mot)}")
```

```
est_parfait(PAUL) renvoie (50, 1612112, False)
est_parfait(ALAIN) renvoie (37, 1121914, True)
```

Exercice Bonus 6.46 (Pydéfis – Premier particulier (1))

Un nombre est premier s'il a exactement 2 diviseurs, 1 et lui même. D'autre part, il existe une infinité de nombres premiers de la forme $34k+35$ (avec $k \geq 0$).

L'entrée du problème est un nombre entier n. Vous devez répondre en donnant le n-ème nombre premier de la forme $34k+35$ ainsi que la valeur de k pour laquelle on l'obtient.

Testez votre code : par exemple, si nous prenions les nombres premiers de la forme $v(k) = 5k+7$ et cherchions le troisième (entrée du problème $n = 3$), les réponses seraient 37 et 6. En effet $v(0) = 7$ est premier, $v(1) = 12$ n'est pas premier, $v(2) = 17$ est premier, $v(3) = 22$, $v(4) = 27$, $v(5) = 32$ ne sont pas premiers, et $v(6) = 37$ est le troisième nombre premier obtenu. Pour cet exemple, il faudrait donc répondre $37, 6$.

Source : https://pydefis.callicode.fr/defis/PremierParticulier1/txt

Exercice Bonus 6.47 (Pydéfi – Einstein)

Un nombre d'Einstein E est tel que $E = mc^2$ où m et c sont des nombres premiers différents.

Par exemple, 68 est un nombre d'Einstein car il peut s'écrire sous la forme $68 = 17 \times 2^2$ où 2 et 17 sont des nombres premiers.

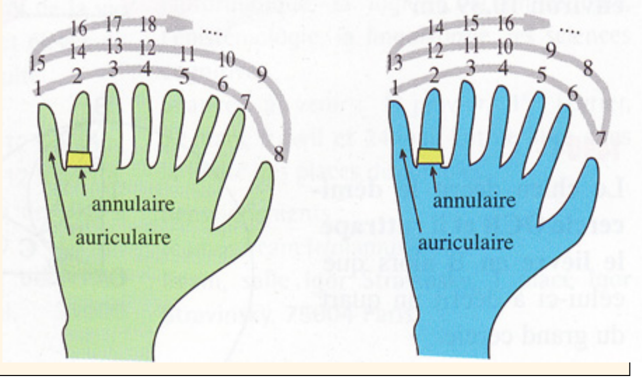
Défi : trouvez les 4 premiers nombres d'Einstein impairs consécutifs (la différence entre deux nombres impairs consécutifs est 2).

Source : https://pydefis.callicode.fr/defis/Einstein/txt

Exercice Bonus 6.48 (Défi Turing n° 19 – Rencontre du quatrième type)

Des petits hommes verts rencontrent des petits hommes bleus.

À leur grand étonnement, ils constatent que leurs mains ne comportent pas le même nombre de doigts : 7 pour les bleus et 8 pour les verts. Mais les savants des deux peuples remarquent que si l'on compte sur les doigts comme indiqué sur la figure, en faisant des allers-retours de l'auriculaire vers le pouce, puis du pouce vers l'auriculaire, certains nombres se comptent à la fois sur l'annulaire des mains bleues et sur celui des mains vertes (le 2 et le 14 par exemple). Ces nombres ont été qualifiés d'*annulaires* par les savants. Calculer la somme des nombres annulaires compris entre 1 et 2013.



Correction

Code *brute force* :

```python
def nb_sur_annulaire(dm,nb_max):
    # dm = doigts de la main
    A = [2]
    i = 1
    while A[-1]<nb_max-(dm-2)*2:
        A.append(A[-1]+(dm-2)*2)
        A.append(A[-1]+2)
        i+=1
    return A

# TEST
nb_max = 25
```

```
V = nb_sur_annulaire(8,nb_max)
B = nb_sur_annulaire(7,nb_max)
Communs = [i for i in V if i in B]
print(f"Nombres sur l'annulaire de la main verte: {V}")
print(f"Nombres sur l'annulaire de la main bleu: {B}")
print(f"Nombres en commun {Communs}")

# DEFI
nb_max = 2013
V = nb_sur_annulaire(8,nb_max)
B = nb_sur_annulaire(7,nb_max)
s = sum([i for i in V if i in B])
print(s)
```

```
Nombres sur l'annulaire de la main verte: [2, 14, 16]
Nombres sur l'annulaire de la main bleu: [2, 12, 14, 24, 26]
Nombres en commun [2, 14]
94848
```

En prenant l'auriculaire comme point de départ on voit qu'il y a un cycle de $2\times7-2=12$ pour les bleus et $2\times8-2=14$ pour les verts. On voit que les nombres annulaires sont à +2 ou 0 modulo 12 pour les bleus ou modulo 14 pour les verts.

```
def ann(x,dm1,dm2):
    m1 = 2*dm1-2
    m2 = 2*dm2-2
    if (x%m1==2 or x%m1==0) and (x%m2==2 or x%m2==0):
        return True
s = sum([x for x in range(1,2014) if ann(x,8,7)])
print(s)
```

```
94848
```

Exercice Bonus 6.49 (Pydéfi – Vif d'or)

Si Nicolas Flamel est célèbre pour sa pierre philosophale, Léonard de Pise, de 200 ans son aîné, a aussi laissé sa trace chez les sorciers, en créant le vif d'or. Bien que les mouvements du vif d'or semblent erratiques, celui-ci a en fait pour fonction de joindre des points de l'espace dans un ordre préétabli.

Précisément, la position du vif d'or est définie par trois coordonnées : abscisse, ordonnée et hauteur. À partir de sa position (x, y, z), le vif va alors se déplacer en ligne droite vers sa position suivante donnée par $(y, z, (x + y + z)\%n)$. Pour rappel, l'opération % donne le reste de la division entière, et n est ici une valeur positive, supérieure à 1, fixée à l'avance.

Le vif d'or entame toujours sa course en $(0,0,1)$. Supposons que n soit égal à 85, alors le vif d'or joindra successivement les positions suivantes

```
1.  (0, 0, 1)
2.  (0, 1, 1)
3.  (1, 1, 2)
4.  (1, 2, 4)
5.  (2, 4, 7)
6.  (4, 7, 13)
7.  (7, 13, 24)
8.  (13, 24, 44)
9.  (24, 44, 81)
10. (44, 81, 64)
11. (81, 64, 19)
...
```

Au bout d'un certain temps, le vif d'or reviendra à sa position d'origine et suivra donc à nouveau exactement le même parcours. Dans l'exemple précédent, cela se produit à la position 2977 :

```
...
2974. (0, 84, 1)
2975. (84, 1, 0)
2976. (1, 0, 0)
2977. (0, 0, 1)
2978. (0, 1, 1)
2979. (1, 1, 2)
...
```

En choisissant la valeur de n, on fait donc varier la trajectoire du vif d'or, ainsi que le temps qu'il met avant de revenir à sa position d'origine.

Léonard de Pise a calculé les 10 meilleures valeurs de n et les a entrées dans tous les vifs d'or. Il s'agit des valeurs de n, inférieures ou égales à 200, qui donnent les parcours les plus longs (c'est-à-dire pour lesquels le vif d'or effectue le plus de mouvements avant de repasser par sa position initiale).

Afin d'essayer de prévoir les déplacements du vif d'or lors du prochain match de Quidditch, vous avez en tête de découvrir ces 10 valeurs.

Source : https://pydefis.callicode.fr/defis/TrajetVifOr/txt

Exercice 6.50 (F: $\mathbb{R} \to \mathbb{R}^2$)

Devine les résultats :

```
f1 = lambda x:x+1
f2 = lambda x:x**2
F = [f1,f2]
print( [f(1) for f in F] )

F = lambda x : [x+1,x**2]
print( F(1) )

F = lambda x : [f1(x),f2(x)]
print( F(1) )
```

Correction

$$
\begin{aligned}
F \colon \mathbb{R} &\to \mathbb{R}^2 \\
x &\mapsto \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} x+1 \\ x^2 \end{pmatrix}
\end{aligned}
$$

On obtient

```
[2, 1]
[2, 1]
[2, 1]
```

Exercice Bonus 6.51 (Défi Turing n° 52 – multiples constitués des mêmes chiffres)

On peut constater que le nombre 125874 et son double 251748 sont constitués des mêmes chiffres, mais dans un ordre différent.

Trouver le plus petit $n \in \mathbb{N}^*$ tel que n, $2n$, $3n$, $4n$, $5n$ et $6n$ soient constitués des mêmes chiffres.

Correction

```
f = lambda n : sorted([int(c) for c in str(n)])

def Turing_52():
	n = 1
	B = f(n)
	while B!=f(2*n) or B!=f(3*n) or B!=f(4*n) or B!=f(5*n) or B!=f(6*n):
		n += 1
		B = f(n)
	return n
```

```
n = Turing_52()
print(f"n={n}, 2n={2*n}, 3n={3*n}, 4n={4*n}, 5n={5*n}, 6n={6*n}")
```

```
n=142857, 2n=285714, 3n=428571, 4n=571428, 5n=714285, 6n=857142
```

Exercice Bonus 6.52 (Défi Turing n° 61 – Non à l'isolement !)

Un chiffre est isolé s'il a comme voisin gauche et voisin droit un chiffre différent de lui-même. Par exemple, dans 776 444, 6 est isolé, mais les autres chiffres ne le sont pas.

D'autre part, les trois premiers nombres non multiples de 10 dont les carrés ne contiennent aucun chiffre isolé sont : $88^2 = 7744$, $74162^2 = 5500002244$ et $105462^2 = 11122233444$. Quel est le quatrième ?

Correction

```
def is_isolated(N):
	s = str(N)
	if s[0]!=s[1] or s[-1]!=s[-2]: # s[0] ou s[-1] est isolé
		return False
	for i in range(1,len(s)-1):
		if s[i-1]!=s[i] and s[i]!=s[i+1]: # s[i] est isolé
			return False
	return True

#print( is_isolated(87**2), is_isolated(88**2) )

L = []
n = 88
while len(L)<4:
	if is_isolated(n*n): L.append(n)
	n+=1
	if n%10==0: n+=1
print(L)
```

```
[88, 74162, 105462, 2973962]
```

Exercice 6.53 (Code César)

Le codage de César est une manière de crypter un message de manière simple : on choisit un nombre n (appelé clé de codage) et on décale toutes les lettres de notre message du nombre choisi. Exemple avec $n = 2$: la lettre “A” deviendra “C”, le “B” deviendra “D” … et le “Z” deviendra “B”. Ainsi, le mot “MATHS” deviendra, une fois codé, “OCVJU” (pour décoder, il suffit d'appliquer le même algorithme avec $n = -2$). La question à laquelle il faut répondre a été codée : si la question est “TRGZKRCVWIRETV” et la clé de codage est $n = 17$, quelle est la réponse à la question ?

Correction

```
def cesar(n,msg):
	abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
	new = ""
	for lettre in msg:
		i = abc.index(lettre)
		new += abc[(i+n)%26]
	return new

print(cesar(2,'MATHS'))
print(cesar(-2,'OCVJU'))
print(cesar(-17,'TRGZKRCVWIRETV'))
```

```
OCVJU
MATHS
CAPITALEFRANCE
```

La réponse est donc Paris.

Remarque : pour connaître le code ASCII (entier) associé à un caractère on peut utiliser la fonction `ord()` :

```
>>> ord("a")
97
>>> ord("z")
122
>>> ord("z")-ord("a")+1 # nb de caractères entre 'a' et 'z'
26
```

Réciproquement, pour connaître le caractère associé à un code ASCII on peut utiliser la fonction `chr()` :

```
>>> chr(65)
'A'
>>> chr(90)
'Z'
>>> chr(ord("a")+6) # 6ième caractère après 'a'
'g'
```

On peut alors réécrire le code de César comme

```
def cesar(n,msg):
	new = ""
	for lettre in msg:
		new += chr(((ord(lettre) - ord("A") + n) % 26) + ord("A"))
	return new

print(cesar(2,'MATHS'))
print(cesar(-2,'OCVJU'))
print(cesar(-17,'TRGZKRCVWIRETV'))
```

```
OCVJU
MATHS
CAPITALEFRANCE
```

☆ Exercice Bonus 6.54 (Défi Turing n° 17 – Nombres amicaux)

Pour un entier $n \in \mathbb{N}$ donné, on note $d(n)$ la somme des diviseurs propres de n (diviseurs de n inférieurs à n). On dit que deux entiers $a, b \in \mathbb{N}$ avec $a \neq b$ sont amicaux si $d(a) = b$ et $d(b) = a$. Par exemple,

$$a = 220 \text{ est divisible par } 1,2,4,5,10,11,20,22,44,55,110 \qquad b = 284 \text{ est divisible par } 1,2,4,71,142$$
$$d(a) = 1+2+4+5+10+11+20+22+44+55+110 = 284 = b \qquad d(b) = 1+2+4+71+142 = 220 = a$$

donc 220 et 284 sont amicaux.

Trouvez la somme de tous les nombres amicaux inférieurs à 10000.

Correction

Le plus simple est de tester, pour tous les nombres $n > 1$ si, lorsque $m = d(n)$, alors $n = d(m)$.

La fonction dp prend en entrée un entier $n \in \mathbb{N}$ et renvoi la somme des diviseurs propres de n.

```
dp = lambda n : sum([x for x in range(1,int(n/2)+1) if (n%x==0)])

Amis = []
for n in range(1,10000+1):
	m=dp(n)
	if m!=n and dp(m)==n :
		Amis.append(n)
print(sum(Amis))
```

```
31626
```

Exercice Bonus 6.55 (map)

Soit $n \in \mathbb{N}$ (par exemple, $n = 123$). Que fait la commande `list(map(int,str(n)))` ? Écrire un script qui donne le même résultat sans utiliser `map` mais en utilisant une liste de compréhension.

Correction

```
>>> n = 123
>>> list(map(int,str(n)))
[1, 2, 3]
>>> [int(x) for x in str(n)]
[1, 2, 3]
```

Explications :

- `str(n)` transforme n en une chaîne de caractères
- `map(f,s)` équivaut à `f(x) for x in s`
- `list(M)` transforme `M` en une liste :

Exercice Bonus 6.56 (Tickets t+ RATP)

Au moment où cet exercice est rédigé, 1 ticket de métro vaut 1.90 € mais si on les achète par carnet de dix, le prix du carnet est de 16.90 €.

On veut calculer le nombre de tickets et le nombre de carnet à acheter en minimisant la dépense (quitte à garder des tickets inutilisés)

Par exemple, avec 9 tickets il convient d'acheter un carnet et garder 2 tickets inutilisés plutôt qu'acheter 9 tickets à l'unité car sans carnet on paye $1.90 \times 9 = 17.10$ € tandis qu'un carnet coûte 16.90 €.

Écrire une script qui calcule la somme minimale à payer si on prévoit `n` voyages ; il devra aussi afficher combien de carnet acheter et s'il faut acheter des tickets à l'unité ou combien de voyages non utilisés restent sur un carnet.

Correction

```
prixTicket = 1.90
prixCarnet = 16.90

def RATP(n):
    somme_due = n*prixTicket
    nbCarnets = 0
    nbTickets = n
    if somme_due <= prixCarnet:
        print(f"Pour {n} voyage(s)")
        print(f"\ton achète {nbTickets} ticket(s) à l'unité et on paye {somme_due:.2f}€")
        return
    while somme_due > prixCarnet:
        nbCarnets += 1
        nbTickets -= 10
        somme_due -= prixCarnet
    nbTicketsRestes = max(0,10*nbCarnets-n)
    nbTicketsAAcheter = max(0,nbTickets)
    somme_due=nbCarnets*prixCarnet+nbTicketsAAcheter*prixTicket

    print(f"Pour {n} voyage(s)")
    print(f"\tsans carnet on payerait {n*prixTicket:.2f}")
    if nbTicketsRestes>0:
        print(f"\tsi on achète {nbCarnets} carnet(s) on paye {somme_due:.2f}€ et il reste
        ↪ {nbTicketsRestes} voyage(s)")
    else:
        if nbTicketsAAcheter>0:
            print(f"\tsi on achète {nbCarnets} carnet(s) et {nbTicketsAAcheter} ticket(s) à
            ↪ l'unité on paye {somme_due:.2f}€")
```

```
        else:
            print(f"\tsi on achète {nbCarnets} carnet(s) on paye {somme_due:.2f}€")
    return

# TEST
for n in [8,9,10,11,12,20,21]: # nombres de voyages
    RATP(n)
```

```
Pour 8 voyage(s)
    on achète 8 ticket(s) à l'unité et on paye 15.20€
Pour 9 voyage(s)
    sans carnet on payerait 17.10
    si on achète 1 carnet(s) on paye 16.90€ et il reste 1 voyage(s)
Pour 10 voyage(s)
    sans carnet on payerait 19.00
    si on achète 1 carnet(s) on paye 16.90€
Pour 11 voyage(s)
    sans carnet on payerait 20.90
    si on achète 1 carnet(s) et 1 ticket(s) à l'unité on paye 18.80€
Pour 12 voyage(s)
    sans carnet on payerait 22.80
    si on achète 1 carnet(s) et 2 ticket(s) à l'unité on paye 20.70€
Pour 20 voyage(s)
    sans carnet on payerait 38.00
    si on achète 2 carnet(s) on paye 33.80€
Pour 21 voyage(s)
    sans carnet on payerait 39.90
    si on achète 2 carnet(s) et 1 ticket(s) à l'unité on paye 35.70€
```

Variante :

```
def RATP(n):
    nbCarnets, nbTickets = divmod(n,10)
    if nbTickets*prixTicket>=prixCarnet :
        nbCarnets += 1
        nbTickets -= 10
    nbTicketsRestes = max(0,10*nbCarnets-n)
    nbTicketsAAcheter = max(0,nbTickets)
    somme_due = nbCarnets*prixCarnet+nbTicketsAAcheter*prixTicket
    if nbCarnets==0:
        print(f"Pour {n} voyage(s))
        print(f"\ton achète {nbTickets} ticket(s) à l'unité et on paye {somme_due:.2f}€")
        return
    print(f"Pour {n} voyage(s)")
    print(f"\tsans carnet on payerait {n*prixTicket:.2f}")
    if nbTicketsRestes>0:
        print(f"\tsi on achète {nbCarnets} carnet(s) on paye {somme_due:.2f}€ et il reste
         {nbTicketsRestes} voyage(s)")
    else:
        if nbTicketsAAcheter>0:
            print(f"\tsi on achète {nbCarnets} carnet(s) et {nbTicketsAAcheter} ticket(s) à
             l'unité on paye {somme_due:.2f}€")
        else:
            print(f"\tsi on achète {nbCarnets} carnet(s) on paye {somme_due:.2f}€")
    return
```

☆ Exercice Bonus 6.57 (Tickets réseau Mistral)
Écrire une script qui calcule la somme minimale à payer et le type de Tickets du réseau Mistral à acheter si on prévoit

vBus voyages en bus et vBat voyages en bateau-bus. Au moment où cet exercice est rédigé voici les prix :

- 1 voyage terrestre coûte 1.40 €
- 1 voyage maritime coûte 2.00 €
- 1 carnet de 10 voyages terrestres ou maritimes coûte 10.00 €

Correction

```
cBus = 1.40
cBat = 2.00
cCar = 10.00

def Mistral(vBus,vBat):
    pBus = vBus*cBus
    pBat = vBat*cBat
    p = pBus+pBat
    print(f"\tsans carnet on paye {p:.2f} €")

    nCar = 0
    v = vBus+vBat
    while v>10 or p>cCar : # 10 car 1 carnet = 10 voyages
        nCar += 1
        v -= 10
        if vBat>=10:
            vBat -= 10
        else:
            vBus = vBus-(10-vBat)
            vBat = 0
        p = vBus*cBus+vBat*cBat

    return nCar, vBus, vBat, v, nCar*cCar+max(0,vBus)*cBus+max(0,vBat)*cBat

# TEST
for vBus,vBat in [(7,0),(8,0),(0,6),(5,6),(3,11),(3,13)]:
	print(f"Pour {vBus+vBat} voyage(s) dont {vBus} en bus et {vBat} en mer")
	nCar, vBus, vBat, v, p = Mistral(vBus,vBat)
	if nCar>0:
		if v<=0:
			print(f"\tavec {nCar} carnet(s), on paye {p:.2f} € et il reste {-v} voyages")
		else:
			print(f"\tavec {nCar} carnet(s), {vBus} ticket(s) de bus et {vBat} de bateau, on
			↪ paye {p:.2f} €")
```

```
Pour 7 voyage(s) dont 7 en bus et 0 en mer
	sans carnet on paye 9.80 €
Pour 8 voyage(s) dont 8 en bus et 0 en mer
	sans carnet on paye 11.20 €
	avec 1 carnet(s), on paye 10.00 € et il reste 2 voyages
Pour 6 voyage(s) dont 0 en bus et 6 en mer
	sans carnet on paye 12.00 €
	avec 1 carnet(s), on paye 10.00 € et il reste 4 voyages
Pour 11 voyage(s) dont 5 en bus et 6 en mer
	sans carnet on paye 19.00 €
	avec 1 carnet(s), 1 ticket(s) de bus et 0 de bateau, on paye 11.40 €
Pour 14 voyage(s) dont 3 en bus et 11 en mer
	sans carnet on paye 26.20 €
	avec 1 carnet(s), 3 ticket(s) de bus et 1 de bateau, on paye 16.20 €
Pour 16 voyage(s) dont 3 en bus et 13 en mer
	sans carnet on paye 30.20 €
	avec 2 carnet(s), on paye 20.00 € et il reste 4 voyages
```

Variante

```
cBus = 1.40
cBat = 2.00
cCar = 10.00

def Mistral(vBus,vBat):
    print(f"\tsans carnet on paye {cBus*vBus+cBat*vBat:.2f}€")
    nbCarnets, r = divmod(vBus+vBat,10) # 10 car 1 carnet = 10 voyages
    if 10*nbCarnets >= vBat:
        vBat = 0
        vBus = r
        if cBus*vBus > cCar :
            nbCarnets += 1
            vBus -= 10
    else :
        vBat = r-vBus
        if cBat*vBat+cBus*vBus >= cCar :
            nbCarnets += 1
            vBus -= 10-vBat
            vBat = 0

    v = vBus+vBat
    p = nbCarnets*cCar+max(0,vBus)*cBus+max(0,vBat)*cBat
    return nbCarnets, vBus, vBat, v, p

# TEST
for vBus,vBat in [(7,0),(8,0),(0,6),(5,6),(3,11),(3,13)]:
	print(f"Pour {vBus+vBat} voyage(s) dont {vBus} en bus et {vBat} en mer")
	nCar, vBus, vBat, v, p = Mistral(vBus,vBat)
	if nCar>0:
		if v<=0:
			print(f"\tavec {nCar} carnet(s), on paye {p:.2f}€ et il reste {-v} voyages")
		else:
			print(f"\tavec {nCar} carnet(s), {vBus} ticket(s) de bus et {vBat} de bateau, on
			↪ paye {p:.2f}€ ")
```

☆ Exercice Bonus 6.58 (Problème d'Euler n°9)

Le triplet d'entiers naturels non nuls (a, b, c) est pythagoricien si $a^2 + b^2 = c^2$. Par exemple, $(3, 4, 5)$ est un triplet pythagoricien car $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

Trouver le seul triplet Pythagoricien pour lequel $a + b + c = 1000$.

Correction

Nous savons que $a < c$, $b < c$ et nous pouvons étudier seulement les cas $a < b$. Nous n'avons pas besoin de faire varier les 3 valeurs, en faire varier 2 suffit car si nous connaissons, par exemple, la valeur de b et a, nous pouvons en déduire celle de c en résolvant l'équation $a + b + c = p$ où p est le périmètre (ici $p = 1000$).

L'intérêt d'utiliser une fonction est de pouvoir quitter le calcul dès que le triplet a été trouvé (utiliser une instruction `break` ne permet de sortir que de la boucle la plus interne) :

```
>>> def Pytagore(p):
... 	for a in range(1,int(p/3)+1): # au lieu de range(1,p-1) car a<b<c implique 3a<a+b+c=p
... 		for c in range(int(p/3),p-a): # au lieu de range(a,p-a) car a<b<c implique
↪ 3c>a+b+c=p
... 				b = p-a-c
... 				if (a*a+b*b)==(c*c):
... 					return a,b,c
...
```

```
>>> print(Pytagore(1000))
(200, 375, 425)
```

Exercice Bonus 6.59 (Pydéfis – Cerbère)

Histoire : pour son douzième et dernier travail, Eurysthée tenta de se débarasser d'Hercule en lui demandant de ramener Cerbère, le molosse à trois têtes qui gardait la porte des enfers. Pour Hercule, la première étape était de commencer à naviguer sur le Styx, au bon endroit, afin d'être guidé jusqu'au trône d'Hadès.

Hermès lui indiqua tout d'abord à quelle distance exacte, en kilomètres, était situé le point d'embarquement sur le Styx. La recherche d'Hercule se limitait donc à un grand cercle autour de Mycènes. Puis, Athéna ajouta une précision d'importance :

> Si tu marches vers l'ouest, pendant un nombre entier de kilomètres, puis vers le nord, pendant un nombre entier de kilomètres, alors, tu arriveras précisément au point d'embarquement sur le Styx. Si plusieurs choix s'offrent à toi, choisis celui qui t'emmeneras le plus au nord.

Défi : aide Hercule en lui indiquant la position de l'entrée du Styx. Pour cela, il suffit de donner les deux valeurs : nombre de kilomètres à l'ouest, nombre de kilomètres au nord.

Testez votre code : si Hermès avait initialement indiqué à Hercule que la porte des enfers était située à 50 kilomètres, alors Hercule aurait pu déterminer sa position exacte. En effet, il y a quatre triangles rectangles avec des côtés entiers qui ont un grand côté (hypothénuse) de longeur 50 : les couples (base, hauteur) $(14,48)$, $(48,14)$, $(30,40)$, $(40,30)$. En choisissant le point qui mène le plus au nord, Hercule aurait su que la porte des enfers était située à 14 km à l'ouest et 48 km au nord. Pour aider Hercule, il aurait donc suffit de répondre $(14,48)$.

Source : https://pydefis.callicode.fr/defis/Herculito12Cerbere/txt

Exercice 6.60 (Palindromes)

On dit qu'un entier $n \geq 0$ est un palindrome s'il se lit à l'identique de gauche à droite ou de droite à gauche. Par exemple $n = 178496694871$ est un palindrome.

1. Écrire une fonction `isPal(n)` prenant en argument un entier n et répondant `True` ou `False` suivant que n est (ou n'est pas) un nombre palindrome.
2. Écrire une liste en compréhension qui contient les palindromes de l'intervalle $[100,300]$ (en utilisant la fonction précédente).
3. Écrire une liste en compréhension qui contient les nombre $n < 10^6$ non palindromes mais dont le carré est palindromique. Par exemple 836 en fait partie car $836^2 = 698896$ est palindromique.
4. Trouver le plus petit n non palindrome dont le cube est palindromique.

Correction

```
isPal = lambda n : str(n)==str(n)[::-1]

print( [ i for i in range(100,300+1) if isPal(i) ] )

[101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262, 272,
↪ 282, 292]

L = [ i for i in range(1,10**6+1) if (isPal(i**2) and not isPal(i))  ]
print(L)
print(f"En effet, les carrés sont {[ell**2 for ell in L]}")

[26, 264, 307, 836, 2285, 2636, 22865, 24846, 30693, 798644]
En effet, les carrés sont [676, 69696, 94249, 698896, 5221225, 6948496, 522808225, 617323716,
↪ 942060249, 637832238736]

def cubPal():
    n = 1
    while not ( isPal(n**3) and not isPal(n) ) :
        n += 1
    return n
```

```
n = cubPal()
print(f"Plus petit entier non palindrome dont le cube est palindrome: {n = }, {n**3 = }")
```

```
Plus petit entier non palindrome dont le cube est palindrome: n = 2201, n**3 = 10662526601
```

Exercice Bonus 6.61 (Défi Turing n°73 – Palindrome et carré palindrome)
Un nombre entier est un palindrome lorsqu'il peut se lire de droite à gauche comme de gauche à droite. Par exemple, 235532 et 636 sont des palindromes.

Quel est le plus grand palindrome de 7 chiffres dont le carré est aussi un palindrome?

Correction

```
for n in range(10**7,10**6,-1):
    if str(n)==str(n)[::-1] and str(n**2)==str(n**2)[::-1] :
        print(n,n**2)
        break
```

```
2001002 4004009004004
```

Exercice Bonus 6.62 (Défi Turing n°4 – nombre palindrome)
Un nombre palindrome se lit de la même façon de gauche à droite et de droite à gauche. Le plus grand palindrome que l'on peut obtenir en multipliant deux nombres de deux chiffres est $9009 = 99 \times 91$.

Quel est le plus grand palindrome que l'on peut obtenir en multipliant un nombre de 4 chiffres avec un nombre de 3 chiffres?

Correction
Il s'agit de trouver le plus grand palindrome (nombre qui se lit de la même façon de gauche à droite que de droite à gauche) issu du produit de 2 nombres, l'un à 3 chiffres l'autre à 4. Nous pouvons donc résoudre ce problème de deux façons différentes :

- soit faire le produit de tous les nombres à 3 chiffres avec tous les nombres à 4 chiffres et regarder quel est le plus grand palindrome formé,
- soit lister tous les palindromes inférieurs à $999 \times 9999 = 9989001$ dans l'ordre décroissant et regarder le premier de cette liste qui est égal au produit de 2 nombres l'un à 3 chiffres l'autre à 4.

1-ère piste :

```
L = [a*b for a in range(100,1000) for b in range(1000,10000) if a*b==int(str(a*b)[::-1])]
print(max(L))
```

```
9744479
```

2-nde piste (plus rapide) : on utilise une fonction ainsi on pourra quitter le calcul dès qu'elle trouve le triplet (utiliser une instruction `break` ne permet de sortir que de la boucle la plus interne).

```
def cherche():
        for p in range(1000*10000,100*1000,-1):
                rev = int(str(p)[::-1])
                if p==rev:
                        for a in range(100,1000):
                                b,r1 = divmod(p,a)
                                if r1==0 and 1000<=b<10000:
                                        return(p,a,b)

p,a,b = cherche()
print(f"Le plus grand palindrome est {p}={a}x{b}")
```

```
Le plus grand palindrome est 9744479=983x9913
```

Exercice Bonus 6.63 (Pydéfi – Les bœufs de Géryon)

Histoire : pour son dixième travail, Eurysthée demanda à Hercule de lui rapporter les bœufs de Géryon. Celui-ci, un géant à trois torses, possédait un troupeau de magnifiques bœufs. Naturellement, il veillait jalousement sur son troupeau, aidé par Orthros, son chien tricéphale, et Eurythion, son dragon à sept têtes. Hercule n'eut pas à user de force pour s'emparer du troupeau, car Géryon était joueur et sous-estima les facultés de calcul d'Hercule. Géryon proposa le marché suivant :

> Comme tu vois, j'ai trois sortes de bœufs : des blancs, qui sont les moins nombreux; des roux; des noirs, qui sont les plus nombreux. Et comme j'aime beaucoup les nombres, j'ai fait en sorte que si on multiplie le nombre de bœufs roux par le nombre de bœufs blancs et enfin par le nombre de bœufs noirs, le nombre obtenu est 777 fois plus grand que le nombre total de bœufs. Si tu m'indiques combien j'ai de bœufs en tout, tu pourras partir avec mon troupeau.

Hercule réfléchit un instant et annonça :

> Tu as 21 bœufs blancs, 74 roux et 95 noirs. En effet : $21 \times 74 \times 95 = 147630$ et ton troupeau compte $21+74+95 = 190$ animaux. Et nous avons bien $190 \times 777 = 147630$. Ma réponse est donc 190. Laisse-moi partir maintenant.

Géryon répondit :

> Tu calcules vite, Hercule, mais tu observes mal et ta réponse n'est pas correcte. Tu vois bien que dans mon troupeau, en réalité, le nombre de bœufs noirs vaut moins que le double du nombre de bœufs blancs. Et mon troupeau compte moins de 1000 têtes... Je te laisse une dernière chance...

Défi : pourrez-vous aider Hercule en lui soufflant le nombre de bœufs du troupeau ?

Source : https://pydefis.callicode.fr/defis/Herculito10Boeufs

Exercice Bonus 6.64 (Pydéfi – Le lion de Némée)

Histoire : le premier travail qu'Eurysthée demanda à Hercule fut de lui ramener la peau du lion de Némée. Le terrible animal vivait dans la forêt d'Argolide et terrorisait les habitants de la région. Hercule traversa donc la forêt d'Argolide à la recherche du lion. Là, il vit que des petits panneaux avaient été fixés sur certains arbres. Sur chaque panneau, le nom d'une divinité était inscrit. Hercule nota tous les noms qu'il rencontra. Approchant de l'antre du lion, il vit, gravé dans la pierre :

> La lettre "A" vaut 1, la lettre "B" vaut 2, jusqu'à la lettre "Z" qui vaut 26. Ainsi, le mot : "BABA" vaut 6 (=2+1+2+1). Cherche la valeur de chaque mot, classe-les de la plus petite valeur à la plus grande, et prononce les mots dans cet ordre : le lion se rendra à toi.

Hercule considéra sa liste de divinités :

```
ARTEMIS ASCLEPIOS ATHENA ATLAS CHARON CHIRON CRONOS DEMETER EOS ERIS EROS GAIA HADES
↪ HECATE HEPHAISTOS HERA HERMES HESTIA HYGIE LETO MAIA METIS MNEMOSYNE NYX OCEANOS
↪ OURANOS PAN PERSEPHONE POSEIDON RHADAMANTHE SELENE THEMIS THETIS TRITON ZEUS
```

Voyons : ARTEMIS vaut 85, donc il faut la placer avant ASCLEPIOS qui vaut 99...

Défi : pouvez-vous aider Hercule, en lui indiquant dans quel ordre il doit citer les divinités ?

Source : https://pydefis.callicode.fr/defis/Herculito01Lion

Exercice 6.65 (Écriture d'un entier dans une base quelconque et entiers brésiliens)

1. On considère un entier n écrit en base 10. Écrire une fonction qui renvoie ses chiffres dans son écriture dans une base b quelconque : c'est la liste des restes obtenus dans l'ordre inverse.
   Exemple : 11 en base 10 (*i.e.* $1 \times 10^1 + 1 \times 10^0$) s'écrit $\underline{1011}_2$ en base 2 car $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8+0+2+1$.
   http://villemin.gerard.free.fr/Wwwgvmm/Numerati/Base.htm
2. On dit qu'un entier n est brésilien en base b si tous ses chiffres sont égaux dans cette base de numération. On suppose $1 < b < n-1$ (car on a toujours $n = \underline{11}_{n-1}$ en base $b = n-1$ ainsi, si $b \geq n$, l'entier n se réduirait au seul chiffre n). Montrer que l'entier 80 est cinq fois brésilien (c'est-à-dire dans cinq bases de numération différentes) en utilisant la fonction précédente.
   Un article sur les nombres brésiliens https://oeis.org/A125134/a125134.pdf

Correction
Changement de base :

```
def base(n,b):
    L = []
    q,r = divmod(n,b)
    L.append(r)
    while q>0:
        q,r = divmod(q,b)
        L.append(r)
    return L[::-1]

print(base(11,2))
```

```
[1, 0, 1, 1]
```

80 est brésiliens dans les bases suivantes :

```
def bres80():
    B = []
    for b in range(2,80):
        ecriture = base(80,b)
        
        s = ""
        for c in ecriture:
            s += str(c)
        
        if len(set(ecriture))==1:
            B.append( (b,f"car il s'écrit {s}") )
            if len(B)==5:
                return B

print(f"80 est brésilien dans les bases suivantes: {bres80()}")
```

```
80 est brésilien dans les bases suivantes: [(3, "car il s'écrit 2222"), (9, "car il s'écrit
↪ 88"), (15, "car il s'écrit 55"), (19, "car il s'écrit 44"), (39, "car il s'écrit 22")]
```

☆ Exercice Bonus 6.66 (Suites de Kaprekar))

Soit $n \in \mathbb{N}$. Soit d (resp. c) l'entier formé des chiffres dans l'ordre décroissant (resp. croissant) de u. On pose $\mathrm{K}(u) = d - c$ On va construire une suite définie par récurrence comme suit :

$$\begin{cases} u_0 \\ u_{n+1} = \mathrm{K}(u_n) \end{cases}$$

Écrire une fonction qui calcule la suite.

Il est possible de montrer qu'il existe un indice p et un indice $m > p$ tel que la suite boucle sur la sous-liste $[u_p, u_{p+1}, \dots, u_{m-1}]$ (appelée cycle de u_0 de période $m - p$).

En particulier, si $m = p + 1$, on a $u_m = u_p$ pour tout $m \ge p$ (la suite est stationnaire à partir de p, la période est donc $m - p = 1$).

1. Vérifier que, si $u_0 = 78545$, alors on a un cycle de période 4 (la sous-liste est $[82962, 75933, 63954, 61974]$).
2. Vérifier que, si $u_0 = 1100$, la suite est stationnaire à partir de $n = 4$ de valeur 6174.
3. Vérifier que, si u_0 a exactement 3 chiffres, la suite est stationnaire avec $u_p = 495$ ou 0. Quel est la valeur maximale de p ?
4. Vérifier que, si u_0 a exactement 4 chiffres, la suite est stationnaire avec $u_p = 6174$ ou 0. Quel est la valeur maximale de p ?

https://fr.wikipedia.org/wiki/Algorithme_de_Kaprekar

Correction

```
def K(n):
	c = sorted(list(str(n)))
	d = c[::-1]
	return int(''.join(d))-int(''.join(c))

def suite(u0):
	uu = [u0]
	while True :
		uu.append(K(uu[-1]))
		if uu[-1] in uu[:-1]:
			p = uu.index(uu[-1])
			m = len(uu)-1
			return p,m,uu[p:-1],u0
```

Base 10

```
t = suite(78545)
sol = f"u_0={t[-1]}, p={t[0]}, m={t[1]}, periode={len(t[2])}, cycle={t[2]}"
print(sol)

t = suite(7641)
sol = f"u_0={t[-1]}, p={t[0]}, m={t[1]}, periode={t[1]-t[0]}, cycle={t[2]}"
print(sol)
t = suite(495)
sol = f"u_0={t[-1]}, p={t[0]}, m={t[1]}, periode={len(t[2])}, cycle={t[2]}"
print(sol)

t = suite(1100)
sol = f"u_0={t[-1]}, p={t[0]}, m={t[1]}, periode={t[1]-t[0]}, cycle={t[2]}"
print(sol)
```

```
u_0=78545, p=2, m=6, periode=4, cycle=[82962, 75933, 63954, 61974]
u_0=7641, p=1, m=2, periode=1, cycle=[6174]
u_0=495, p=0, m=1, periode=1, cycle=[495]
u_0=1100, p=4, m=5, periode=1, cycle=[6174]
```

u_0 à trois chiffres :

```
T = []
for n in range(100,1000):
	t = suite(n)
	if len(t[2])>1:
		print("Non constante")
		break
	T.append(t)

print(set([t[2][0] for t in T ]))
```

```
{0, 495}
```

u_0 à quatre chiffres :

```
T = []
for n in range(1000,10000):
	t = suite(n)
	if len(t[2])>1:
		print("Non constante")
		break
	T.append(t)

print(set([t[2][0] for t in T ]))
```

```
{0, 6174}
```

Exercice Bonus 6.67 (Pydéfi – Numération Gibi : le Neutubykw)

Les Gibis sont plus calés en histoire des sciences que les Shadoks. Ils ont leur système de numération spécial, en base 4, mais leurs chiffres sont une référence à des personnages célèbres de l'histoire de l'informatique :

KWA pour le 0 en l'honneur d'Al Khwarizmi

BY pour le 1, rappelant Ada Byron (plus connue sous le nom d'Ada Lovelace)

TU pour le 2, afin que personne n'oublie Alan Turing

NEU pour le 3, nous rappelant combien John von Neumann était brillant

Cette numération s'appelle le Neutubykwa. On trouve l'écriture d'un nombre en Neutubykwa en divisant la quantité à écrire en paquets de 4.

Pour écrire 118, par exemple, on réalise 29 paquets de 4, et il reste 2 unités (118=29*4+2). TU (2) est donc le chiffre des unités. Pour avoir le chiffre des quatrains (celui qui est juste à gauche des unités), on prend les 29 paquets qu'on regroupe par groupes de paquets de 4. Cela donne 7 groupes de paquets de 4, et il restera un seul paquet de 4. Le chiffre des quatrains est donc BY (1). Pour le chiffre situé à gauche de BY, on recommence : les 7 paquets sont groupés en 1 seul tas de 4 et il reste 3 paquets. Le chiffre suivant est donc NEU (3). Et enfin, le tas restant est groupé en 0 paquets de 4, et il reste 1 tas. Le premier chiffre du nombre est donc BY. Ainsi, 118, s'écrit BYNEUBYTU.

Testez votre code : si la liste d'entrée était $[118, 3, 24]$, il faudrait répondre `'BYNEUBYTU', 'NEU', 'BYTUKWA'`.

Défi : donnant leur équivalent en Neutubykwa des nombres $[363, 204, 108, 181, 326, 154, 259, 289, 332, 137]$.

Source : https://pydefis.callicode.fr/defis/Neutubykwa

Exercice Bonus 6.68 (Pydéfi – Pokédex en vrac)

Dans l'univers Pokémon, un dresseur est une personne qui capture des Pokémons sauvages, les élève et les entraîne à combattre les Pokémons d'autres dresseurs. Le protagoniste de chaque version des jeux vidéo Pokémon est un dresseur ambitieux; Sacha est le plus célèbre d'entre eux.

Sacha a déjà capturé de nombreux Pokémons et son Pokédex est presque rempli! Mais il n'est pas très ordonné, il a juste noté au fur et à mesure les numéros des Pokémons qu'il a capturés... Il vous confie son Pokédex et vous demande de lui fournir une vue synthétique de sa liste de Pokémons :

- les numéros des Pokémons sont listés par séquences séparées par des virgules,
- les séquences peuvent être de la forme :
  - $i - j$ indiquant que le Pokédex contient les Pokémons du numéro i au numéro j inclus, mais pas les Pokémons $i-1$ et $j+1$,
  - k indiquant que le Pokédex contient le Pokémon de numéro k, mais pas les Pokémons $k-1$ et $k+1$
  - les séquences sont ordonnées de manière croissante.

Par exemple si le Pokédex contenait les numéros suivants :

$$2, 10, 5, 1, 7, 9, 8$$

la vue synthétique serait représentée par :

$$1-2, 5, 7-10$$

Le Pokédex de Sacha est bien plus vaste, vous pouvez le télécharger ici : https://pydefis.callicode.fr/defis/C22_GenList/input. Saurez-vous lui indiquer sa vue synthétique?

https://pydefis.callicode.fr/defis/C22_GenList/txt

Exercice Bonus 6.69 (Défi Turing n°141 – Combien de 6 ?)

On multiplie les chiffres composant un nombre plus grand que 9. Si le résultat est un nombre à un chiffre, on l'appelle l'image du nombre de départ. S'il a plus d'un chiffre, on répète l'opération jusqu'à obtenir un nombre à un chiffre.

Par exemple

$$666 \xrightarrow[6\times 6\times 6]{} 216 \xrightarrow[2\times 1\times 6]{} 12 \xrightarrow[1\times 2]{} 2$$

Combien de nombres entre 10 et 10 millions ont comme image 6 ?

Correction

```python
# def prod(L):
#     p = 1
#     for ell in L:
#         p *= ell
#     return p

# depuis python 3.8 on peut écrire
from math import prod

d = { i:0 for i in range(10) }
for i in range(10,10**7+1):
    while len(str(i))>1:
        i = prod([int(i) for i in str(i)])
    d[i] += 1
print(d)
```

```
{0: 9394518, 1: 6, 2: 86092, 3: 27, 4: 6173, 5: 7413, 6: 318456, 7: 27, 8: 187196, 9: 83}
```

☆ Exercice Bonus 6.70 (Défi Turing n°33 – Pâques en avril)

Durant les années 2001 à 9999 (bornes comprises), combien de fois la date de Pâques tombera-t-elle en avril, dans le calendrier grégorien ?

Cf. https://fr.wikipedia.org/wiki/Calcul_de_la_date_de_P%C3%A2ques_selon_la_m%C3%A9thode_de_Gauss

Correction

```python
def paques(a):
    n = a%19 # cycle de Méton
    c,u = divmod(a,100) # centaine et rang de l'année
    s,t = divmod(c,4) # siècle bissextile
    p = (c+8)//25 # cycle de proemptose
    q = (c-p+1)//3 # proemptose
    e = (19*n+c-s-q+15)%30 # épacte
    b,d = divmod(u,4) # année bissextile
    L = (2*t+2*b-e-d+32)%7 # lettre dominicale
    h = (n+11*e+22*L)//451 # correction
    m,j = divmod(e+L-7*h+114,31)
    #print(f'a={a}, n={n}, c={c}, u={u}, s={s}, t={t}, p={p}, q={q}, e={e}, b={b}, d={d},
    ↪ L={L}, h={h}, m={m}, j={j} ')
    return m,j

def afficher(a):
    x = paques(a)
    m = x[0]
    j = x[1]
    if m==3:
        print(f"En {a} le dimanche de Pâques est le {j+1} mars")
    elif m==4:
        print(f"En {a} le dimanche de Pâques est le {j+1} avril")
    else:
        print("Error")
```

```
# TEST
for a in range(2020,2031):
    afficher(a)

# # DEFIS
# dico = {3:0,4:0}
# for a in range(2001,9999+1):
#     m,_ = paques(a)
#     dico[m] += 1
#
# print(dico)
```

```
En 2020 le dimanche de Pâques est le 12 avril
En 2021 le dimanche de Pâques est le 4 avril
En 2022 le dimanche de Pâques est le 17 avril
En 2023 le dimanche de Pâques est le 9 avril
En 2024 le dimanche de Pâques est le 31 mars
En 2025 le dimanche de Pâques est le 20 avril
En 2026 le dimanche de Pâques est le 5 avril
En 2027 le dimanche de Pâques est le 28 mars
En 2028 le dimanche de Pâques est le 16 avril
En 2029 le dimanche de Pâques est le 1 avril
En 2030 le dimanche de Pâques est le 21 avril
```

☆ Exercice Bonus 6.71 (Sun angle)

Tout vrai voyageur doit savoir faire 3 choses : réparer le feu, trouver l'eau et extraire des informations utiles de la nature qui l'entoure. La programmation ne vous aidera pas avec le feu et l'eau, mais en ce qui concerne l'extraction d'informations, c'est peut-être exactement ce dont vous avez besoin. Votre tâche est de trouver l'angle du soleil au-dessus de l'horizon en connaissant l'heure de la journée.

On suppose que le soleil se lève à l'Est à 6h00, ce qui correspond à l'angle de 0° ; à 12h00, le soleil atteint son zénith, ce qui signifie que l'angle est égal à 90° ; 18h00 est l'heure du coucher du soleil, l'angle est donc de 180°.

Écrire une fonction `sun_angle(s)`, où s est une chaîne de caractère contenant l'heure au format "hh :mm", qui renvoie l'angle en dégrées du soleil au-dessus de l'horizon. Si l'heure d'entrée est avant 6h00 ou après 18h00, la fonction devra retourner "Je ne vois pas le soleil !".

Source : https://py.checkio.org/en/mission/sun-angle/

Correction

```
def sun_angle(time):
    h,m = time.split(":")
    m_tot = int(h)*60+int(m)
    if m_tot<6*60 or m_tot>18*60:
        return "I don't see the sun!"
    else:
        return 180/((18-6)*60)*(m_tot-6*60)

print(sun_angle("07:00"))
print(sun_angle("12:15"))
print(sun_angle("01:23"))
```

```
15.0
93.75
I don't see the sun!
```

☆ Exercice Bonus 6.72 (Angle entre les aiguilles d'une horloge)

Écrire une fonction qui prend en entrée une chaîne de caractères représentant l'heure au format `HH:MM:SS` et retourne l'angle le plus petit entre les aiguilles des heures et des minutes de l'horloge pour ce moment donné.

L'angle doit être exprimé en degrés, être compris entre 0 et 180 et arrondi à deux décimales près. Par exemple, si l'entrée est `15:15:30`, la sortie sera `7.50`.

Ne pas oublier de prendre en compte les secondes qui font avancer l'aiguille des heures.

Correction

1. On crée la fonction `angle_aiguilles_horloge` avec comme seul paramètre d'entrée "`heure`" qui sera la chaîne de caractères représentant l'heure au format `HH:MM:SS`.
2. On divise cette chaîne en ses composantes heures, minutes et secondes en utilisant la méthode `split()` et on les convertit en nombres entiers.
3. Si l'heure est égale à 12, on la remplace par 0.
4. On procède ensuite au calcul du nombre total de minutes : aux minutes donnés on ajoute une fraction de minute qui est due aux secondes (converties en minutes selon `secondes/60`).
5. Pour calculer l'angle entre les aiguilles des heures et des minutes, on utilise alors la formule suivante :

$$\vartheta = \left| \underbrace{360\left(\frac{\text{heures}}{12} + \frac{\text{total_minutes}}{60\times 12}\right)}_{\text{Position de l'aiguille des heures}} - \underbrace{360\left(\frac{\text{total_minutes}}{60}\right)}_{\text{Pos. aig. minutes}} \right|$$

6. Si l'angle calculé dépasse 180 degrés, on prendra l'angle complémentaire $(360 - \vartheta)$ pour obtenir l'angle le plus petit.
7. Enfin, on retourne l'angle arrondi à deux décimales près.

```
def angle_aiguilles_horloge(heure):

    composantes = heure.split(':')
    heures, minutes, secondes = [int(c) for c in composantes]

    heures = heures%12
    total_minutes = minutes + secondes / 60

    angle = abs(360 * (heures/12 + total_minutes/60/12) - 360 * (total_minutes/60))
    angle = min( angle , 360 - angle)

    return round(angle, 2)

print(angle_aiguilles_horloge("12:00:00"))
print(angle_aiguilles_horloge("12:00:30"))
print(angle_aiguilles_horloge("12:01:00"))
print(angle_aiguilles_horloge("15:15:00"))
```

```
0.0
2.75
5.5
7.5
```

☆ Exercice Bonus 6.73 (Most Wanted Letter)

Dans un texte en anglais, qui contient des lettres et des signes de ponctuation, il faut déterminer la lettre la plus fréquente (*cf.*. exercice 4.33). La lettre renvoyée doit être en casse minuscule. Dans cette recherche, la casse n'a pas d'importance. On considère par exemple, pour compter le nombre de `"a"`, que `"A" == "a"`. Assurez vous de ne compter ni les signes de ponctuation, ni les chiffres, ni les espaces : uniquement les lettres.

Si deux lettres ou plus apparaissent à la même fréquence, il faut renvoyer la liste de ces lettres classées par ordre alphabétique. Par exemple : `"one"` contient `"o"`, `"n"`, `"e"` une fois chacun, donc on doit renvoyer `["e","n","o"]`.

© G. Faccanoni

Input : Un texte à analyser comme chaîne de caractères.

Output : La liste des lettres les plus fréquentes en minuscule.

Source : https://py.checkio.org/fr/mission/most-wanted-letter-2/

Correction

```
def myhisto(text: str) -> dict:
	lista = [c for c in text.lower() if c.isalpha()]
	lettre = {}
	for c in lista:
		lettre[c] = lettre.get(c,0)+1
	return lettre

def most_wanted(text: str) -> str:
	lettre = myhisto(text)
	out = [key for key in lettre.keys() if lettre[key] == max(lettre.values())]
	out.sort()
	return out

# TESTS
TEST = [ "Hello World!","How do you do?","One","Oops!","AAaooo!!!!","abe","a" * 9000 + "b" *
↪ 1000]

print( *[ f"{most_wanted(text)}" for text in TEST ] , sep = "\n" )
```

```
['l']
['o']
['e', 'n', 'o']
['o']
['a', 'o']
['a', 'b', 'e']
['a']
```

☆ Exercice Bonus 6.74 (Bigger Price)

Vous avez une liste avec tous les produits disponibles dans un magasin. Les données sont représentées sous forme de liste de dictionnaires, chaque dictionnaire représentant un produit avec deux clés : `"name"` et `"price"`. Le nombre des produits les plus chers sera donné comme premier argument (`int`) et toutes les données comme second argument. Votre mission ici est de trouver les produits les plus chers (et renvoyer une liste de dictionnaires avec juste ces éléments).

Source : https://py.checkio.org/mission/bigger-price/solve/

Correction

```
bigger_price = lambda n,data : sorted( data , key=lambda x:x["price"] ) [len(data)-n:]

# TESTS
data=[{"name": "bread",      "price": 100},\
      {"name": "wine",       "price": 138},\
      {"name": "meat",       "price": 15},\
      {"name": "water",      "price": 1},\
      {"name": "pen",        "price": 5},
      {"name": "whiteboard", "price": 170}]

# print(bigger_price(2,data))
# print(bigger_price(1,data))

from tabulate import tabulate
print(tabulate(bigger_price(2,data), headers="keys"))
# print(tabulate(bigger_price(1,data), headers="keys"))
```

```
name          price
----------  -------
wine            138
whiteboard      170
```

☆ Exercice Bonus 6.75 (Sum by Types)

Vous avez une liste. Chaque valeur de cette liste peut être une chaîne de caractères ou un entier. Votre tâche ici est de renvoyer deux valeurs. Le premier est une concaténation de toutes les chaînes de la liste donnée. Le second est une somme de tous les entiers de la liste donnée.

Source : https://py.checkio.org/mission/sum-by-type/solve/

Correction

```python
def sum_by_types(L):
	s = ''.join([ell for ell in L if type(ell)==str])
	n = sum([ell for ell in L if type(ell)==int])
	return (s, n)

# TESTS
print(sum_by_types([]))
print(sum_by_types([1, 2, 3]))
print(sum_by_types(['1', 2, 3]))
print(sum_by_types(['1', '2', 3]))
print(sum_by_types(['1', '2', '3']))
print(sum_by_types(['size', 12, 'in', 45, 0]))
```

```
('', 0)
('', 6)
('1', 5)
('12', 3)
('123', 0)
('sizein', 57)
```

☆ Exercice Bonus 6.76 (Common Words)

Continuons à examiner les mots. On vous donne deux chaînes de caractères avec des mots séparés par des virgules. Votre fonction doit trouver tous les mots qui apparaissent dans les deux chaînes de caractères. Le résultat doit être représenté comme une chaîne de mots séparés par des virgules dans l'ordre alphabétique.

Préconditions :

- tous les mots sont séparés par des virgules,
- tous les mots sont composés de lettres romaines en minuscule,
- les mots ne sont pas répétés au sein d'une même chaîne de caractères.

Source : https://py.checkio.org/mission/common-words/solve/

Correction

```python
def common_words(line1: str, line2: str) -> str:
	L1 = line1.split(',')
	L2 = line2.split(',')
	L3 = [mot for mot in L1 if mot in L2]
	L3.sort()
	if len(L3)!=0:
		return ','.join(L3)
	else:
		return ''

# TESTS
```

```
print(common_words('hello,world', 'hello,earth'))
print(common_words('one,two,three', 'four,five,six'))
print(common_words('one,two,three', 'four,five,one,two,six,three'))
```

```
hello

one,three,two
```

☆ Exercice Bonus 6.77 (Flatten list)

Soit en entrée une liste qui contient des entiers ou d'autres listes d'entiers imbriquées. Vous devez mettre toutes les valeurs entières dans une seule liste. L'ordre doit être tel qu'il était dans la liste d'origine.

Exemples :

- `flat_list([1, 2, 3])` renvoie `[1, 2, 3]`
- `flat_list([1, [2, 2, 2], 4])` renvoie `[1, 2, 2, 2, 4]`
- `flat_list([[[2]], [4, [5, 6, [6], 6, 6, 6], 7]])` renvoie `[2, 4, 5, 6, 6, 6, 6, 6, 7]`
- `flat_list([-1, [1, [-2], 1], -1])` renvoie `[-1, 1, -2, 1, -1]`

Source : https://py.checkio.org/en/mission/flatten-list/

Correction
Avec une fonction récursive :

```
def flat_list(array):
    L = []
    for item in array:
        if type(item) == list: # idem que if isinstance(item,list):
            L.extend(flat_list(item))
        else:
            L.append(item)
    return L

# En version one-liner
# f = lambda array : [array] if int==type(array) else sum( map(f,array) , [] )

# TEST
a1 = [1,4,3,2]
a2 = [1,[2,2,2],4]
a3 = [[[2]], [4, [5, 6, [6], 6, 6, 6], 7]]
a4 = [-1,[1,[-2,[3],[[5],[10,-11],[1,100,[-1000,[5000]]],[20,-10,[[[]]]]]]]]
a5 = [[7007],[6006],[7007]]
a6 = [[[[53],-43],33],23]
a7 = [20,[20,[20,[20]]]]
a8 = [1001,2001,3001,[4001,5001]]
a9 = [10,[20,[30]]]
a10 = [[[22]],[44,[55,66,[66],66,66,66],77]]
a11 = [13,[23,23,23],43]
a12 = [[7],[6],[7]]
a13 = [[[[3],3],3],3]
for array in [a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13]:
    print(f"{array}\n{flat_list(array)}\n")
```

```
[1, 4, 3, 2]
[1, 4, 3, 2]

[1, [2, 2, 2], 4]
[1, 2, 2, 2, 4]

[[[2]], [4, [5, 6, [6], 6, 6, 6], 7]]
```

```
[2, 4, 5, 6, 6, 6, 6, 6, 7]

[-1, [1, [-2, [3], [[5], [10, -11], [1, 100, [-1000, [5000]]], [20, -10, [[[]]]]]]]]
[-1, 1, -2, 3, 5, 10, -11, 1, 100, -1000, 5000, 20, -10]

[[7007], [6006], [7007]]
[7007, 6006, 7007]

[[[[53], -43], 33], 23]
[53, -43, 33, 23]

[20, [20, [20, [20]]]]
[20, 20, 20, 20]

[1001, 2001, 3001, [4001, 5001]]
[1001, 2001, 3001, 4001, 5001]

[10, [20, [30]]]
[10, 20, 30]

[[[22]], [44, [55, 66, [66], 66, 66, 66], 77]]
[22, 44, 55, 66, 66, 66, 66, 66, 77]

[13, [23, 23, 23], 43]
[13, 23, 23, 23, 43]

[[7], [6], [7]]
[7, 6, 7]

[[[[3], 3], 3], 3]
[3, 3, 3, 3]
```

☆ Exercice Bonus 6.78 (Plan cyclique)

On considère au plus 26 personnes notées A, B, C, D, E, F ... qui peuvent s'envoyer des messages. Le plan d'envoi doit respecter deux règles :

- chaque personne ne peut envoyer des messages qu'à une seule personne (éventuellement elle-même),
- chaque personne ne peut recevoir des messages qu'en provenance d'une seule personne (éventuellement elle-même).

Il s'agit donc d'une bijection de l'ensemble de personnes dans lui même.

Pour un plan d'envoi donné, s'il existe une suite de personnes dans laquelle la dernière est la même que la première, on dit que c'est un cycle. Lorsqu'un plan d'envoi comporte un unique cycle, on dit que le plan d'envoi est cyclique.

Plan a : Voici un exemple avec 6 personnes de «plan d'envoi des messages»

$$A \to E \to B \to F \to A \qquad C \to D \to C$$

C'est bien un plan car il respecte les deux règles (c'est une bijection). Pour le vérifier, il suffit d'écrire dans une matrice dans la première ligne les personnes qui envoient puis dans la deuxième celle qui reçoivent :

$$\begin{matrix} A & E & B & F & C & D \\ E & B & F & A & D & C \end{matrix}$$

on remarque chaque personne est présente une seule fois dans chaque ligne. De plus, il y a deux cycles distincts : un premier cycle avec A, E, B, F et un second cycle avec C et D.

Le dictionnaire correspondant à ce plan d'envoi est

```
plan_a = {'A':'E', 'B':'F', 'C':'D', 'D':'C', 'E':'B', 'F':'A'}
```

Plan b : Le plan d'envoi suivant

```
plan_b = {'A':'C', 'B':'F', 'C':'E', 'D':'A', 'E':'B', 'F':'D'}
```

comporte un unique cycle : A → C → E → B → F → D → A. Dans ce cas le plan d'envoi est cyclique.

Pour savoir si un plan d'envoi de messages comportant N personnes est cyclique, on peut utiliser l'algorithme ci-dessous :

- on part d'un expéditeur et on inspecte son destinataire dans le plan d'envoi,
- chaque destinataire devient à son tour expéditeur, selon le plan d'envoi, tant qu'on ne « retombe » pas sur l'expéditeur initial,
- le plan d'envoi est cyclique si on l'a parcouru en entier.

Correction

Sujet 38.2 : https://glassus.github.io/terminale_nsi/T6_6_Epreuve_pratique/BNS_2023/?s=03

```python
def est_cyclique(plan):
    '''
    Prend en paramètre un dictionnaire `plan` correspondant à un plan d'envoi de messages
    (ici entre les personnes A, B, C, D, E, F).
    Renvoie True si le plan d'envoi de messages est cyclique et False sinon.
    '''

    # si le plan ne respecte pas les règles
    if sorted(plan.keys()) != sorted(plan.values()):
        return False

    expediteur = 'A'
    destinataire = plan[expediteur]
    nb_destinaires = 1

    while destinataire != expediteur:
        destinataire = plan[destinataire]
        nb_destinaires += 1

    return nb_destinaires == len(plan)

#tests
print(est_cyclique({'A':'B', 'B':'C'})) # no plan
print(est_cyclique({'A':'B', 'B':'A'})) # plan cyclique
print(est_cyclique({'A':'B', 'B':'A', 'C':'C'})) # plan 2 cycles
print(est_cyclique({'A':'E', 'F':'A', 'C':'D', 'E':'B', 'B':'F', 'D':'C'})) # plan 2 cycles
print(est_cyclique({'A':'E', 'F':'C', 'C':'D', 'E':'B', 'B':'F', 'D':'A'})) # plan cyclique
print(est_cyclique({'A':'B', 'F':'C', 'C':'D', 'E':'A', 'B':'F', 'D':'E'})) # plan cyclique
print(est_cyclique({'A':'B', 'F':'A', 'C':'D', 'E':'C', 'B':'F', 'D':'E'})) # plan 2 cycles
```

```
False
True
False
False
True
True
False
```

Ajouter le calcul du nombre de cycles, il suffit d'éliminer au fur et à mesure les éléments du dico, si on termine le cycle car la clé n'est plus là mais coincide avec le premier element du cycle, on commence un nouveau cycle

CHAPITRE 7

Modules

En Python, un module est une collection de fonctions prédéfinies qui peuvent être importées dans un programme pour être utilisées selon les besoins. Cela permet d'éviter de réécrire du code pour des tâches courantes et de bénéficier de fonctionnalités avancées déjà implémentées. Grâce à ces modules, les développeurs peuvent importer des fonctions préexistants dans leur code et étendre facilement les fonctionnalités de leur programme.

Il y a différents types de modules : ceux qui sont inclus dans la version de Python comme `random` ou `math`, ceux que l'on peut rajouter comme `numpy` ou `matplotlib` et ceux que l'on peut faire soi-même (il s'agit dans les cas simples d'un fichier Python contenant un ensemble de fonctions).

Outre le module, un deuxième niveau d'organisation permet de structurer le code : les fichiers Python peuvent être organisés en une arborescence de répertoires appelée "package". Un package est donc un module contenant d'autres modules. Les modules d'un package peuvent être des sous-packages, ce qui donne une structure arborescente.

Le site pypi.python.org/pypi (The Python Package Index) recense des milliers de modules et de packages!

7.1. Importation des fonctions d'un module

Pour utiliser des fonctions faisant partie d'un module, il faut avant tout les importer.

Pour importer un module, on peut utiliser deux syntaxes :

1. La syntaxe générale est `import ModuleName`.
   Les fonctions s'utilisent sous la forme `ModuleName.FunctionName(parameters)`. Remarquons que la commande qui permet de calculer est précédée du module duquel elle vient.

2. On peut utiliser un alias pour le nom du module : on écrira alors `import ModuleName as Alias`.
   Les fonctions s'utilisent alors sous la forme `Alias.FunctionName(parameters)`.

3. Il est également possible d'importer seulement quelque fonction d'un module :
   `from ModuleName import function1, function2`.
   Dans ce cas les fonctions peuvent être utilisées directement par `function1(parameters)`.

En résumé :

Méthode 1.
```
import ModuleName
ModuleName.FunctionName(parameters)
```
Méthode 2.
```
import ModuleName as Alias
Alias.FunctionName(parameters)
```
Méthode 3.
```
from ModuleName import function1, function2
function1(parameters)
```

Il est possible d'obtenir une aide sur le module avec la commande `help(ModuleName)`.

La liste des fonctions définies dans un module peut être affichée par la commande `dir(ModuleName)`.

ATTENTION (POURQUOI NE PAS UTILISER `from ModuleName import *`)
Parfois, il est plus facile d'importer tout le contenu d'un module en utilisant un astérisque (`*`) au lieu de la liste de fonctions : `from ModuleName import *`. Dans ce cas les fonctions peuvent être utilisées directement par `FunctionName(parameters)`.

Cette méthode est sans danger si on importe un seul module. Cependant, de manière générale, on essaie d'éviter autant que possible de brutalement tout importer, afin d'éviter d'éventuels conflits. En effet, si on charge deux modules qui définissent tous les deux une même fonction, il vaut mieux opter pour la première syntaxe. Par exemple, si `module1` et `module2` définissent tous les deux la fonction `toto` alors il faudra écrire :

```
import module1
import module2

module1.toto(x)
module2.toto(x)
```

EXEMPLE
Nous allons nous intéresser aux fonctions mathématiques qui sont essentiellement rassemblées dans les modules `math` et `cmath`, le deuxième étant spécialisé pour les nombres complexes. Nous allons utiliser la fonction `sqrt` (racine carrée) :

```
>>> from cmath import *
>>> from math import *
>>> print('Racine carrée de -4 =', sqrt(-4))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

Les deux modules contiennent une fonction `sqrt`, mais Python va utiliser **celle du dernier module importé**, c'est-à-dire celle du module `math`. Par conséquent, une erreur d'exécution se produira puisqu'il faut la version complexe pour calculer la racine carrée de -4.

Pour résoudre ce problème, Python permet de juste importer un module avec l'instruction import suivie du nom du module à importer. Ensuite, lorsqu'on voudra utiliser une fonction du module, il faudra faire précéder son nom de celui du module :

```
>>> import cmath
>>> import math
>>> print('Racine carrée de -4 =', cmath.sqrt(-4))
Racine carrée de -4 = 2j
```

Il est maintenant explicite que la fonction `sqrt` appelée est celle provenant du module `cmath`, et il n'y aura donc plus d'erreurs d'exécution.

EXEMPLE (EXAMPLE OF NEED FOR NAMESPACES)
Le module `scipy` définit deux fonctions (informatiques) `gamma` : l'une est définie dans le module `scipy.special` est correspond à la fonction (mathématique) gamma, l'autre est définie dans le module `scipy.stats` est correspond à la distribution gamma.

```
scipy.special.gamma # gamma function
scipy.stats.gamma # gamma distribution
```

7.2. Quelques modules

Python dispose d'une bibliothèque standard qui comprend plus de deux cents modules prêts à l'emploi. Cette bibliothèque couvre une large gamme de domaines, tels que les mathématiques (fonctions mathématiques usuelles, calculs sur les nombres réels, les nombres complexes, la combinatoire, les matrices, la manipulation d'images, ...), l'administration système, la programmation réseau, la manipulation de fichiers, etc. L'utilisation de ces modules permet d'éviter de

devoir réinventer la roue à chaque fois que l'on souhaite écrire un programme, ce qui peut considérablement accélérer le développement et la maintenance du code.[1]

7.2.1. Le module `time`

Le module `time` fournit beaucoup de fonctions en rapport avec le temps : gestions de calendriers, des systèmes horaires, d'horloges... Il permet aussi une estimation des temps d'exécution des programmes. La qualité de cette estimation dépend des mises en oeuvre de ce module et des machines visées.[2]

La fonction `time()` est annoncée avec une précision de la seconde, sans que ce soit garanti pour toutes les machines. Une seconde est suffisamment long pour effectuer 2 giga = $2 \cdot 10^9$ opérations! Elle peut donc être utile pour mesurer des simulations ou des exécutions conséquentes. La fonction `perf_counter()` permet les mesures les plus fines possibles adaptées à la mesure des temps d'exécution de programmes (sans être pour autant exactes, ni reproductibles). Elle utilise des fonctions spécifiques aux architectures des machines qui exploitent les compteurs matériels.

La fonction `perf_counter()` renvoie une durée de temps en secondes et permet de mesurer des durées d'exécution par soustraction. On procède en deux appels qui encadrent la portion de code à mesurer :

- `t0 = time.perf_counter()` : le premier appel initialise une valeur initiale `t0`;
- `t = time.perf_counter()` : le second appel mesure `t` à l'issue de l'exécution de la portion de code,
- `t-t0` est la mesure de ce temps d'exécution (moyennant les réserves précédentes).

Voici un exemple qui permet de comparer le temps d'exécution pour ajouter des éléments à une liste avec trois techniques : l'opérateur +, la méthode append et la création par une liste en compréhension :

```
from time import perf_counter

N = 5*10**7

debut = perf_counter()
L = []
for i in range(N):
	L += [i]
fin = perf_counter()
print(f"Avec += temps d'exécution={fin-debut:g}s")

debut = perf_counter()
L = []
for i in range(N):
	L.append(i)
fin = perf_counter()
print(f"Avec append temps d'exécution={fin-debut:g}s")

debut = perf_counter()
L = [x for x in range(N)]
fin = perf_counter()
print(f"Avec comprehensions-list temps d'exécution={fin-debut:g}s")
```

1. À mon avis, des modules fondamentaux pour des étudiants en licence mathématique sont • `math`, • `random`, • `numpy`, • `matplotlib`, • `scipy`, • `sympy`, • `pandas`. Nous allons en voir quelques uns dans ce cours d'initiation à la programmation informatique avec Python, les autres seront rencontrés dans les ECUEs M25, M43 et M62 de la Licence Mathématiques de l'Université de Toulon.

2. Mesurer le temps d'exécution d'un programme sur un ordinateur moderne est un véritable défi. Ne vous laissez pas tromper par l'apparente rapidité des résultats, car l'ordinateur effectue en même temps de nombreuses tâches simultanées, telles que la gestion du système d'exploitation ou des périphériques. De plus, mesurer avec précision les temps d'exécution des programmes est compliqué en raison de l'organisation complexe de la mémoire et des unités de calculs. En pratique, plus ce que l'on veut mesurer est court, plus la mesure sera incertaine et non reproductible. Pour obtenir des résultats significatifs, il faut répéter l'exécution dans une boucle, mesurer le temps d'exécution de cette boucle et en retenir la moyenne. Les transferts de données entre la mémoire et les unités de calculs sont également un facteur important à considérer, et leur temps peut être jusqu'à dix fois plus long que celui des calculs eux-mêmes. À titre indicatif, il y a une facteur 10 entre ce temps de transfert et le temps d'un calcul arithmétique élémentaire. Ainsi, les mesures peuvent être surprenantes lorsqu'elles deviennent "dominées" par ces temps de transfert. Ces défis mettent en évidence la différence entre les propriétés théoriques d'un algorithme et sa mise en œuvre pratique.

```
Avec += temps d'exécution=5.24129s
Avec append temps d'exécution=3.96227s
Avec comprehensions-list temps d'exécution=1.92065s
```

On en conclut que l'utilisation des comprehensions-list est à privilégier et que la méthode `append` est plus efficace que l'instruction +=.

7.2.2. Le module `math`

Dans Python seulement quelque fonction mathématique est prédéfinie :

`abs(a)`	Valeur absolue de a
`max(suite)`	Plus grande valeur de la suite
`min(suite)`	Plus petite valeur de la suite
`round(a,n)`	Arrondi a à n décimales près
`pow(a,n)`	Exponentiation, renvoi a^n, équivalent à `a**n`
`sum(L)`	Somme des éléments de la suite
`divmod(a,b)`	Renvoie quotient et reste de la division de a par b
`cmp(a,b)`	Renvoie $\begin{cases} -1 & \text{si } a < b, \\ 0 & \text{si } a = b, \\ 1 & \text{si } a > b. \end{cases}$

Toutes les autres fonctions mathématiques sont définies dans le module `math`. Comme mentionné précédemment, on dispose de plusieurs syntaxes pour importer des fonctions d'un module :

```python
import math
print(math.sin(math.pi)) # sin(π)
```

1.2246467991473532e-16

```python
import math as mm
print(mm.sin(mm.pi))
```

1.2246467991473532e-16

```python
from math import sin, pi
print(sin(pi))
```

1.2246467991473532e-16

```python
from math import *
print(sin(pi))
```

1.2246467991473532e-16

Voici la liste des fonctions définies dans le module `math` :

```python
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',
↪ 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees',
↪ 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
↪ 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',
↪ 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter',
↪ 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
↪ 'tau', 'trunc', 'ulp']
```

Notons que le module définit les deux constantes π et e.

```python
import math
print(math.pi, math.e, math.exp(1))
```

```
3.141592653589793 2.718281828459045 2.718281828459045
```

Attention, les fonctions `prod` et `dist` ne sont disponibles que depuis la version 3.8 de Python.

```
import math
print(math.prod([1,2,3,4]))

p = [3, 3]
q = [6, 12]
# Calculate Euclidean distance
print(math.dist(p, q)) # = √((x_p − x_q)^2 + (y_p − y_q)^2)
```

```
24
9.486832980505138
```

Remarque
L'arithmétique à virgule flottante est sujette à un débordement (*overflow*) si une valeur devient trop grande. Cela provoque un type d'erreur (*exception*) qui, si elle n'est pas traitée, interrompt l'exécution du programme :

```
>>> import math
>>> math.exp(1000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: math range error
```

En revanche, la taille des nombres entiers n'est limitée que par la quantité de mémoire disponible, ainsi 1000! peut être évalué avec précision à l'aide de la fonction `math.factorial` :

```
>>> import math
>>> math.factorial(1000)
402387260077093773543702433923003985719374864210714632543799910429938512398629020592044208486969404800479998
```

7.2.3. Le module `fractions`

Ce module permet de manipuler des nombres rationnels. L'exemple suivant se passe de commentaires

```
from fractions import Fraction
print(1/3+2/5)
print(Fraction(1,3)+Fraction(2,5))
print(5/15)
print(Fraction(5,15))
```

```
0.7333333333333334
11/15
0.3333333333333333
1/3
```

Dans l'annexe A on verra des exemples d'utilisation de ce module.

7.2.4. Le module `random`

Ce module propose diverses fonctions permettant de générer des nombres (pseudo-)aléatoires qui suivent différentes distributions mathématiques. Il apparaît assez difficile d'écrire un algorithme qui soit réellement non-déterministe (c'est-à-dire qui produise un résultat totalement imprévisible). Il existe cependant des techniques mathématiques permettant de simuler plus ou moins bien l'effet du hasard. Voici quelques fonctions fournies par ce module : [3]

3. https://docs.python.org/fr/3/library/random.html

`randrange(p,n)`	choisit un entier aléatoirement dans la liste `range(p,n)`, *i.e.* dans la liste `[p,p+1,...,p+n-1]`
`randint(a,b)`	choisit un *entier* aléatoirement dans l'intervalle $[a;b]$ (alias pour `random.randrange(p,n+1))`
`randrange(p,n,h)`	choisit un éléments aléatoirement dans la liste `range(p,n,h)`
`random()`	renvoie un *décimal* aléatoire dans $[0;1[$
`uniform(a,b)`	choisit un *décimal* aléatoire dans $[a;b]$
`shuffle(seq)`	mélange la liste `seq` (en la modifiant)
`choice(seq)`	choisit un éléments aléatoirement dans la liste `seq`
`choices(seq,k=nb)`	tirage de `nb` éléments dans la liste `seq` avec répétition (on remet l'objet tiré avant de retirer)
`sample(seq,k=nb)`	tirage de `nb` (< `len(seq)`) éléments dans la liste `seq` sans répétition (comme si on tire `nb` éléments en même temps)

Tous les tirages sont réalisé selon une loi uniforme. Voir la documentation du module pour d'autres lois.

Voici quelques exemples :

1. Quelques tirages :

```
>>> import random
>>> random.randrange(50,100,5)
70
>>> random.randint(50,100)
88
>>> random.choice([1,7,10,11,12,25])
11
>>> random.random()
0.3983249272671874
>>> random.uniform(10,20)
13.049656233736348
```

2. Simulons le lancé d'un dé :

```
>>> from random import *

>>> dico = {1:0, 2:0, 3:0, 4:0, 5:0, 6:0}
>>> # avec une compréhension :
>>> # dico = { c:0 for c in range(1,6)}

>>> for i in range(1000):
...     n = randint(1,6)
...     dico[n]+=1
...
>>> print(dico)
{1: 153, 2: 158, 3: 164, 4: 174, 5: 173, 6: 178}
```

3. On veut simuler 20 tirages du jeu “Pierre Feuille Ciseaux” sous forme d'une liste.

```
>>> from random import *
>>> seq = ["Pierre", "Feuille", "Ciseaux"]
>>> # print([choice(seq) for _ in range(20)])
>>> print(choices(seq,k=20))
['Pierre', 'Ciseaux', 'Pierre', 'Ciseaux', 'Feuille', 'Feuille', 'Ciseaux', 'Ciseaux',
 ↪ 'Ciseaux', 'Pierre', 'Ciseaux', 'Pierre', 'Feuille', 'Ciseaux', 'Feuille', 'Ciseaux',
 ↪ 'Pierre', 'Feuille', 'Feuille', 'Pierre']
```

7.2.5. Le module `numpy` (algèbre matricielle)

Par convention, `numpy` est importé comme `np`.

Le module `numpy` est la boîte à outils indispensable pour faire du calcul scientifique avec Python. Pour modéliser les vecteurs, matrices, et plus généralement les tableaux à n dimensions, `numpy` fournit le type `array`. Il y a des différences

majeures avec les listes (resp. les listes de listes) qui pourraient elles aussi nous servir à représenter des vecteurs (resp. des matrices) :

- les tableaux `numpy` sont homogènes, c'est-à-dire constitués d'éléments du même type ;
- la taille des tableaux `numpy` est fixée à la création. On ne peut donc augmenter ou diminuer la taille d'un tableau comme le ferait pour une liste (à moins de créer un tout nouveau tableau, bien sûr).

Traditionnellement, on charge la totalité du module `numpy` de la manière suivante :

```
import numpy as np
```

On évitera de charger `numpy` par "`from numpy import *`" : le nombre de fonctions importées est en effet trop important et avec lui le risque d'homonymie avec les définitions déjà présentes au moment de l'importation.

Vous étudierez ce module dans l'ECUE M43. Dans ce cours nous nous limiterons à l'utiliser pour ses fonctions vectorisées en combinaison avec le module `matplotlib` (*cf.* chapitre 8).

Remarque (Fonction vectorisée)
Une fonction vectorisée ("universelle" ou «ufunc», abréviation de *universal function*, est le terme exacte) est une fonction qui peut s'appliquer terme à terme aux éléments d'un tableau. Si f est une ufunc et si $a = [a_0, a_1, \dots, a_{n-1}]$ est un tableau, alors $f(a)$ renvoie le tableau $[f(a_0), f(a_1), \dots, f(a_{n-1})]$. Un grand nombre de fonctions usuelles sont directement «universalisées» dans `numpy`. Les fonctions mathématiques gardent le même nom (préfixé par `np` si on a importé `numpy` par `import numpy as np`).

```
import numpy as np
xx = np.array(range(4))
yy = xx**2
print(xx,type(xx))
print(yy,type(yy))
```

```
[0 1 2 3] <class 'numpy.ndarray'>
[0 1 4 9] <class 'numpy.ndarray'>
```

https://perso.univ-perp.fr/langlois/images/pdf/mp/www.mathprepa.fr-une-petite-reference-numpy.pdf

7.2.6. Le module `scipy` (calcul approché)

Par convention, `scipy` est importé comme `sp`.

Cette librairie contient de nombreux algorithmes très utilisés par les personnes qui font du calcul scientifique : fft, algèbre linéaire (méthodes directes ou itératives pour résoudre des systèmes linéaires, ...), interpolation, intégration numérique, statistiques et autres algorithmes numériques. On peut voir ce module comme une extension de `Numpy` car **il contient toutes les fonctions de** `Numpy`.

Le site de la documentation en fournit la liste : http://docs.scipy.org/doc/scipy/reference

Voici deux exemples d'utilisation : le calcul approché d'une solution d'une équation et le calcul approché d'une intégrale.

`fsolve` Si on ne peut pas calculer analytiquement la solution d'une équation, on peut l'approcher numériquement. Tout d'abord on définit une fonction f telle que $f(x) = 0$ ssi x est solution de l'équation donnée, on se donne ensuite un point x_0 pas trop éloigné de la solution cherchée et on utilise la fonction `fsolve` du module `scipy.optimize` dont la syntaxe est `fsolve(f,x0)`. Cette fonction renvoie une liste contenant la solution (approchée) et une estimation de l'erreur.

Voici un exemple : on cherche à calculer la solution de l'équation $x = \cos(x)$. Une étude des fonction $x \mapsto x$ et $x \mapsto \cos(x)$ montre que la solution se trouve entre 0 et $\pi/2$. On peut donc poser $f(x) = x - \cos(x)$ et $x_0 = 1$:

```
from math import cos
from scipy.optimize import fsolve

f = lambda x: x-cos(x)
x0=1
sol=fsolve(f,x0)[0] # On ne garde que la solution approchée
print(sol)
```

0.7390851332151607

`integrate.quad` Pour approcher la valeur numérique d'une intégrale on peut utiliser la fonction quad du sous-module `integrate` du module `scipy` https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

```
from scipy import integrate # on importe toutes les fonctions du sous-module integrate

a = 0
b = 1
f = lambda x : x**2
integr = integrate.quad(f,a,b) # On approche ∫_a^b f(x) dx  avec f(x) = x^2
print("Integrale =",integr[0], " Erreur =",integr[1] )
```

```
Integrale = 0.33333333333333337  Erreur = 3.700743415417189e-15
```

```
from scipy import exp, sqrt, pi, inf
from scipy.integrate import quad

f = lambda x : exp(-x*x)
app = quad(f, 0,inf)[0] # On approche ∫_0^∞ e^{-x^2} dx
print(app)
print(sqrt(pi)/2) # Valeur exacte de cette intégrale
```

```
0.8862269254527579
0.8862269254527579
```

7.2.7. ☆Le module `SymPy` (calcul formel)

SymPy est une bibliothèque Python pour les mathématiques symboliques. Elle prévoit devenir un système complet de calcul formel ("CAS" en anglais : *Computer Algebra System*) tout en gardant le code aussi simple que possible afin qu'il soit compréhensible et facilement extensible.

- Quelques commandes : https://www.sympygamma.com/
- Pour tester en ligne : https://live.sympy.org/

Voici un petit exemple d'utilisation :

```
import sympy as sym
sym.var('x,y')
```

```
# Calculs
s = (x+y)+(x-y)
print(s)
```

$2x$

```
# Simplifications
expr = sym.sin(x)**2 + sym.cos(x)**2
print(expr,"=",sym.simplify(expr))

expr = (x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
print(expr,"=",sym.simplify(expr))
```

$\sin^2(x) + \cos^2(x) = 1$

$$\frac{x^3 + x^2 - x - 1}{x^2 + 2x + 1} = x - 1$$

```
# Fractions
expr = (x**3-y**3)/(x**2-y**2)
print(expr,"=",sym.cancel(expr))
```

$$\frac{x^3-y^3}{x^2-y^2}=\frac{x^2+xy+y^2}{x+y}$$

Encore un exemple :

```
import sympy as sym
sym.var('x')
functions = [sym.sin(x), sym.cos(x), sym.tan(x)]
for f in functions:
	d = sym.Derivative(f, x)
	i = sym.Integral(f, x)
	print(d, "=" , d.doit(), "\t" , i , "=" , i.doit())
```

$$\frac{d}{dx}\sin(x)=\cos(x) \qquad \int\sin(x)\,dx=-\cos(x)$$
$$\frac{d}{dx}\cos(x)=-\sin(x) \qquad \int\cos(x)\,dx=\sin(x)$$
$$\frac{d}{dx}\tan(x)=\tan^2(x)+1 \qquad \int\tan(x)\,dx=-\log(\cos(x))$$

Si vous utilisez spyder, vous pouvez remplacer `print(...)` par `display(...)`.

7.3. Exercices

Exercice 7.1 (Module `math` – sin, cos, tan, π, e)
Calculer avec le module `math` (resp. `numpy`) la valeur

$$\frac{\sin(3e^2)}{1+\tan\left(\frac{\pi}{8}\right)}.$$

Bonus : et avec le module `sympy` ?

Correction
Les quatre stratégies suivantes sont équivalentes dans ce cas (car il n'y a pas de risque de conflits avec d'autres modules) :

Méthode 1 :
```
import math
print( math.sin(3*math.e**2)/(1+math.tan(math.pi/8)) )

import numpy
print( numpy.sin(3*numpy.e**2)/(1+numpy.tan(numpy.pi/8)) )
```
```
-0.123823019762552
-0.123823019762552
```

Méthode 2 :
```
import math as mm
print( mm.sin(3*mm.e**2)/(1+mm.tan(mm.pi/8)) )

import numpy as np
print( np.sin(3*np.e**2)/(1+np.tan(np.pi/8)) )
```
```
-0.123823019762552
-0.123823019762552
```

Méthode 3 :
```
from math import pi, e, sin, tan
print( sin(3*e**2)/(1+tan(pi/8)) )

from numpy import pi, e, sin, tan
print( sin(3*e**2)/(1+tan(pi/8)) )
```
```
-0.123823019762552
-0.123823019762552
```

Méthode 4 :
```
from math import *   # version du paresseux
print( sin(3*e**2)/(1+tan(pi/8)) )

from numpy import *   # version du paresseux
print( sin(3*e**2)/(1+tan(pi/8)) )
```
```
-0.123823019762552
-0.123823019762552
```

Au lieu d'utiliser la constante `math.e` (resp. `numpy.e`), on aurait pu utiliser la fonction `math.exp` (resp. `numpy.exp`) et donc écrire `math.exp(2)` (resp. `numpy.exp(2)`) au lieu de `math.e**2` (resp. `numpy.e**2`) .

Bonus : en utilisant la deuxième méthode (on peut bien-sûr utiliser aussi les autres méthodes)

```
import sympy as sym
val = sym.sin(3*sym.E**2)/(1+sym.tan(sym.pi/8))
print( val )
print( val.evalf() )
```
```
sqrt(2)*sin(3*exp(2))/2
-0.123823019762553
```

Exercice Bonus 7.2 ($\sqrt{\tan(\pi)}$)
Calculer avec python $\sqrt{\tan(\pi)}$.

Correction
$\tan(\pi) = 0$ mais `math.pi` est juste une approximation de π et la tangent de cette valeur approchée est un nombre négatif dont on ne peut pas calculer la racine :

```
>>> from math import sqrt, tan,pi
>>> sqrt(tan(pi))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

On peut utiliser le calcul formel :

```
>>> from sympy import sqrt, tan,pi
>>> sqrt(tan(pi))
0
```

Source : https://scipython.com/book/chapter-9-general-scientific-programming/questions/evaluating-sqrttanpi/

Exercice 7.3 (Module `math` - Angles remarquables)
Soit $R = [0, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{\pi}{2}]$ une liste d'angles (en radians). Créer, par compréhension, une liste D avec les angles en degrés, une liste C avec l'évaluation du cosinus en ces angles et une liste S avec l'évaluation du sinus. Utiliser d'abord le module `math`, puis le module `numpy`.

Bonus : et avec le module `sympy` ?

Correction

1. Avec les module `math` et des listes en compréhension :

```
from math import pi, cos, sin
R = [0,pi/6,pi/4,pi/3,pi/2]
D = [r*180/pi for r in R]
C = [cos(r) for r in R]
S = [sin(r) for r in R]
print(f"{R = }")
print(f"{D = }")
print(f"{C = }")
print(f"{S = }")
```

```
R = [0, 0.5235987755982988, 0.7853981633974483, 1.0471975511965976, 1.5707963267948966]
D = [0.0, 29.999999999999996, 45.0, 59.99999999999999, 90.0]
C = [1.0, 0.8660254037844387, 0.7071067811865476, 0.5000000000000001,
 ↪ 6.123233995736766e-17]
S = [0.0, 0.49999999999999994, 0.7071067811865475, 0.8660254037844386, 1.0]
```

2. Avec les module `numpy` il n'est pas nécessaire d'utiliser des listes en compréhension car les fonctions sont vectorisées :

```
from numpy import array, pi, cos, sin
R = array([0,pi/6,pi/4,pi/3,pi/2])
D = R*180/pi # [r*180/pi for r in R]
C = cos(R) # [cos(r) for r in R]
S = sin(R) # [sin(r) for r in R]
print(f"{R = }")
print(f"{D = }")
print(f"{C = }")
print(f"{S = }")
```

```
R = array([0.        , 0.52359878, 0.78539816, 1.04719755, 1.57079633])
D = array([ 0., 30., 45., 60., 90.])
C = array([1.00000000e+00, 8.66025404e-01, 7.07106781e-01, 5.00000000e-01,
      6.12323400e-17])
S = array([0.        , 0.5       , 0.70710678, 0.8660254 , 1.        ])
```

3. Bonus (calcul formel) :

```python
import sympy as sym
R = [0,sym.pi/6,sym.pi/4,sym.pi/3,sym.pi/2]
D = [r*180/sym.pi for r in R]
C = [sym.cos(r) for r in R]
S = [sym.sin(r) for r in R]
print(f"{R = }")
print(f"{D = }")
print(f"{C = }")
print(f"{S = }")
```

```
R = [0, pi/6, pi/4, pi/3, pi/2]
D = [0, 30, 45, 60, 90]
C = [1, sqrt(3)/2, sqrt(2)/2, 1/2, 0]
S = [0, 1/2, sqrt(2)/2, sqrt(3)/2, 1]
```

Exercice 7.4 (Module `math` - Paper folding)

Une feuille de papier d'une épaisseur d'un dixième de millimètre est pliée 15 fois en deux : quelle est l'épaisseur du résultat après pliage ? Après combien de pliages l'épaisseur dépasse-t-elle la distance Terre-Lune (la distance Terre-Lune vaut approximativement 384 400 km) ?

Pour calculer $\log_2(x)$ utiliser la fonction `log(x,2)` du module `math`.

Correction

Épaisseur du résultat après pliage : $2^{15} \times 0{,}1\,\text{mm} = 3{,}2768\,\text{m}$.

L'épaisseur dépasse la distance Terre-Lune après n pliage avec n qui vérifie $2^n \times 10^{-4} \ge 3.844 \times 10^8$, *i.e.* $n \ge \log_2(3.844 \times 10^{12})$ ce qui correspond à 42 pliages.

```python
from math import log
d = 384400 * 1.e3       # distance to moon, m
t = 1.e-4               # paper thickness, m
d_over_t = d/t
x = log(d_over_t, 2)    # base-2 logarithm
print(int(x)+1)
```

```
42
```

Exercice 7.5 (Module `math` – Factorielle)

La factorielle d'un nombre n, notée $n!$, est définie par $n \times (n-1) \times \cdots \times 3 \times 2 \times 1$ et par convention $0! = 1$. Par exemple, $10! = 10 \times 9 \times \cdots \times 3 \times 2 \times 1 = 3628800$.

En remarquant que $n! = n \times (n-1)!$, créer une fonction qui calcule la factorielle d'un entier naturel n. Pour vérifier le calcul, on pourra comparer le résultat à la fonction `factorial` du module `math`.

Correction

- Version récursive

```python
factorielle = lambda n : n*factorielle(n-1) if n>1 else 1

# TESTS
from math import factorial
for n in [0,2,3,6,10,50]:
    print(factorial(n)==factorielle(n), end=" ")
```

```
True True True True True True
```

- Une version non récursive :

```
#def factorielle(n):
#    x=1
#    for i in range(2,n+1):
#        x*=i
#    return x

from math import prod # Python > 3.8
factorielle = lambda n : prod([i for i in range(2,n+1)])

# TESTS
from math import factorial
for n in [0,2,3,6,10,50]:
    print(factorial(n)==factorielle(n), end=" ")
```

```
True True True True True True
```

Exercice 7.6 (Module `math` – Stirling)
La formule de Stirling, du nom du mathématicien écossais James Stirling, donne un équivalent de la factorielle d'un entier naturel n quand n tend vers l'infini :

$$\ln(n!) \approx n\ln(n) - n.$$

1. Créez une fonction `exacte(n)` qui prend en entrée un entier n et renvoie $\ln(n!)$.
2. Créez une fonction `approche(n)` qui prend en entrée un entier n et renvoie $n\ln(n) - n$.
3. Écrire un script qui calcule la plus petite valeur de n telle que l'erreur relative
   `|exacte(n)-approche(n)|/exacte(n)`
   est inférieure à 1%.

Source : https://scipython.com/book/chapter-2-the-core-python-language-i/questions/problems/p24/stirlings-approximation/

Correction

```
from math import log, factorial

exacte   = lambda n : log(factorial(n))
approche = lambda n : n*log(n)-n
erreur   = lambda n : abs(exacte(n)-approche(n))/exacte(n)

n = 2
while erreur(n)>=0.01:
    n+=1

print(f"{n=}, {exacte(n)=:g}, {approche(n)=:g}, {erreur(n)=:g}")
```

```
n=90, exacte(n)=318.153, approche(n)=314.983, erreur(n)=0.00996305
```

Exercice 7.7 (Module `math` – Défi Turing n°6 et Projet Euler n°20 – somme de chiffres)
$10! = 3628800$ et la somme de ses chiffres vaut $3+6+2+8+8+0+0 = 27$. Trouver la somme des chiffres du nombre 2013! (défi Turing) et la somme des chiffres du nombre 100! (projet Euler).

Correction
Pour résoudre ce problème on commence par trouver la valeur de 2013! (par exemple en utilisant la fonction `factorial` du module `math`), convertir le nombre obtenu en chaîne de caractère avec `str`, lire les chiffres un par un, les convertir en entier avec `int` et enfin les additionner.

```
>>> import math
>>> print(sum([int(x) for x in str(math.factorial(10))]))
27
>>> print(sum([int(x) for x in str(math.factorial(2013))]))
24021
```

Exercice 7.8 (Module `math` – Nombres de Brown)
Les nombres de Brown sont des couples d'entiers m et n tels que $m^2 = n! + 1$. On peut montrer qu'ils sont tous inférieurs à 100. Calculer tous les nombres de Brown.

Correction

```
>>> from math import factorial
>>> print([ (m,n) for m in range(100) for n in range(100) if m**2==factorial(n)+1 ])
[(5, 4), (11, 5), (71, 7)]
```

Exercice 7.9 (Module `math` – Approximation de e)
Notons $f(i) = \frac{1}{i!}$ et $\tilde{e}(n) = \sum_{i=0}^{n} f(i)$. On sait que

$$e = \lim_{n\to\infty} \tilde{e}(n).$$

1. Créez un fonction `f(i)` qui prend en entrée un entier i et renvoie la valeur $f(i) = \frac{1}{i!}$ en utilisant la fonction `factorial` du module `math`.
2. Créez ensuite une fonction `approx_e(n)` qui prend en entrée un entier n et renvoie $\tilde{e}(n) = \sum_{i=0}^{n} f(i)$ (on construit la liste `[f(0), f(1), ..., f(n)]` en compréhension et on somme ses termes).
3. Écrire un script qui calcule la plus petite valeur de n telle que `approx_e(n) = math.e` à 10^{-15} près.

Correction

```
import math
f = lambda x : 1/math.factorial(x)
approx_e = lambda n : sum([f(i) for i in range(n+1) ])

print(f"Avec n = {10} on obtient e ≈ {approx_e(10)}")

# Méthode non optimisée
n = 0
while abs(approx_e(n)-math.e)>1e-15 and n<100:
	n+=1
print(f"Avec n = {n} on obtient\n{approx_e(n)=}\n{      math.e=}")

# Mieux: on ne recalcule pas toute la liste mais on ajoute juste un terme à chaque itération
n = 0
e_approx = f(n)
while abs(e_approx-math.e)>1e-15 and n<100:
	n += 1
	e_approx +=f(n)
print(f"Avec n = {n} on obtient\n{approx_e(n)=}\n{      math.e=}")
```

```
Avec n = 10 on obtient e ≈ 2.718281801146384
Avec n = 17 on obtient
approx_e(n)=2.7182818284590446
     math.e=2.718281828459045
Avec n = 17 on obtient
approx_e(n)=2.7182818284590446
     math.e=2.718281828459045
```

Exercice 7.10 (Module `math` – Polignac)
Vérifier la formule de Polignac pour le calcul du nombre de zéros de fin dans $n!$:

$$n! \text{ se termine avec } \sum_i \left\lfloor \frac{n}{5^i} \right\rfloor \text{ zero.}$$

Par exemple :

- 9! est égale à 362880, il se termine avec 1 zéros
- 10! est égale à 3628800, il se termine avec 2 zéros
- 20! est égale à 2432902008176640000, il se termine avec 4 zéros.

Source : https://scipython.com/book/chapter-2-the-core-python-language-i/questions/problems/p25/de-polignacs-formula/

Correction

```python
from math import factorial # Pour verification

for n in [9,10,20]:
		
		# POUR VERIFIER
    nf = math.factorial(n)
    print(f'TEST {n:d}! = {math.factorial(n):d}')
		
		# METHODE 1
    nzeros = 0
    term = n
    while True:
        term //= 5
        if term == 0:
            break
        nzeros += term
    print(f'{n:d}! ends in {nzeros:d} zeros.')

		# METHODE 2
    nzeros = 0
    for c in str(nf)[::-1]:
        if c != '0':
            break
        nzeros += 1
    print(f'{n:d}! ends in {nzeros:d} zeros.\n')
```

```
TEST 9! = 362880
9! ends in 1 zeros.
9! ends in 1 zeros.

TEST 10! = 3628800
10! ends in 2 zeros.
10! ends in 2 zeros.

TEST 20! = 2432902008176640000
20! ends in 4 zeros.
20! ends in 4 zeros.
```

Exercice 7.11 (Module `math` – Énoncer des chiffres)
Transforme un nombre n en une chaîne de caractères, sélectionner les x premières décimales et énoncer les chiffres (en français). Par exemple, si $n = \pi$ et $x = 2$, il devra afficher `trois virgule un quatre`. Pour simplifier, nous

considérons que des nombres positifs inférieurs à 10.

Source : https://scipython.com/book/chapter-2-the-core-python-language-i/questions/problems/p24/pi-read-aloud/

Correction

```
def enoncer(n,x):
    num = ('zero', 'un', 'deux', 'trois', 'quatre', 'cinq', 'six', 'sept', 'huit', 'neuf')
    print(f"{n} approché par {x} chiffres après la virgule vaut {round(n,x)}")
    n = str(n)
    print(f"{num[int(n[0])]} virgule ", end='')
    for i in n[2:x+2]:
        print(num[int(i)], end=' ')
    print()

import math
enoncer(math.pi,2)
enoncer(math.e,4)
```

```
3.141592653589793 approché par 2 chiffres après la virgule vaut 3.14
trois virgule un quatre
2.718281828459045 approché par 4 chiffres après la virgule vaut 2.7183
deux virgule sept un huit deux
```

Exercice 7.12 (Module `math` – Overflow)

Calculer l'hypoténuse d'un triangle rectangle dont les cotés mesurent 1.5×10^{200} et 3.5×10^{201}.

Correction

Calcul naif :

```
>>> from math import sqrt
>>> a,b = 1.5e200, 3.5e201
>>> sqrt(a**2+b**2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result out of range')
```

Le problème est que, bien que les valeurs numériques des longueurs des deux côtés puissent toutes être représentées par des float (*i.e.* ils ne sont pas trop grand), le calcul de la longueur de l'hypoténuse sous forme de $c=\sqrt{a^2+b^2}$ semblerait nécessiter les carrés des longueurs opposées et adjacentes, qui débordent. Deux stratégies possibles :

- on remarque que $c=b\sqrt{1+\left(\frac{a}{b}\right)^2}$; le carré ne peut pas déborder si $b>a$, sinon on factorise a ;

```
>>> from math import sqrt
>>> a,b = 1.5e200, 3.5e201
>>> b*sqrt(1+(a/b)**2)
3.5032128111206724e+201
```

- on peut utiliser la fonction `hypot`

```
>>> from math import hypot
>>> a,b = 1.5e200, 3.5e201
>>> hypot(a,b)
3.503212811120672e+201
```

Exercice 7.13 (Module `math` – Distance)

① Soit `P=[Px,Py]` la liste contenant les coordonnées d'un point du plan $P=(P_x,P_y)$. On définit la distance entre deux points P et Q par

$$d(P,Q)=\sqrt{(P_x-Q_x)^2+(P_y-Q_y)^2}.$$

Écrire une fonction `distance(P,Q)` qui retourne $d(P,Q)$.

Exemple : Si P = [0, 0] et Q = [0, 1] alors distance(P,Q) = 1.0

② Soit P un point et L une liste de points. Écrire une fonction `plus_proche(P,L)` qui renvoie le point de la liste L se trouvant à la plus courte distance du point P.

③ Soit `T=[P,Q,R]` une liste contenant trois points du plan.

- Écrire une fonction `perimetre(T)` qui retourne le périmètre du triangle de sommets P, Q et R (en utilisant la fonction `distance`).
- Écrire une fonction `IsEquilateral(T)` qui retourne `True` si le triangle est équilatère, `False` sinon.

Exemple : Si T = [[0, 0], [0, 1], [1, 0]] alors perimetre(T) = 3.41421 et IsEquilateral(T) = False

④ Soit `C=[P,r]` une liste contenant un point du plan et une valeur positive, représentant un cercle de centre P et rayon r.

- Écrire une fonction `superpose(C1,C2)` qui retourne `True` si les deux cercles C1 et C2 se chevauchent, `False` sinon.

Correction

① Distance entre deux points

```
import math
distance = lambda P,Q : math.sqrt((P[0]-Q[0])**2+(P[1]-Q[1])**2)
# distance = lambda P,Q : math.sqrt( sum(  (p-q)**2 for p,q in zip(P,Q)  )  )
# distance = lambda P,Q : math.dist(P,Q) # si Python > 3.8

# TEST
P, Q = [0,0] , [0,1]
print(f"{distance(P,Q) = }")
```

```
distance(P,Q) = 1.0
```

② Point d'une liste le plus proche d'un point donné

```
plus_proche = lambda P,L : min( L , key=lambda ell:distance(ell,P) )

# TEST
P = [0,0]
L = [ [1,0] , [0,1] , [1,1] , [-0.5,-0.5]   ]
N = plus_proche(P,L)
print(f"plus_proche(P,L) renvoie {N}, en effet la distance vaut {distance(P,N)}")
```

```
plus_proche(P,L) renvoie [-0.5, -0.5], en effet la distance vaut 0.7071067811865476
```

③ Triangles

```
perimetre = lambda T : distance(T[0],T[1])+distance(T[0],T[2])+distance(T[1],T[2])
# perimetre = lambda T : sum( [ distance(T[i%3],T[(i+1)%3]) for i in range(3) ] )
IsEquilateral = lambda T: abs(distance(T[0],T[1])-distance(T[0],T[2]))<1.e-10 and \
                                   abs(distance(T[0],T[1])-distance(T[1],T[2]))<1.e-1

# TEST 1
P, Q, R = [0,0] , [0,1] , [1,0]
T = [P,Q,R]
print(f"{T = }")
print(f"{perimetre(T) = }")
print(f"{IsEquilateral(T) = }")

# TEST 2
P, Q, R = [0,0] , [1,0] , [0.5,math.sqrt(1-0.25)]
T = [P,Q,R]
print(f"{T = }")
print(f"{perimetre(T) = }")
print(f"{IsEquilateral(T) = }")
```

```
T = [[0, 0], [0, 1], [1, 0]]
perimetre(T) = 3.414213562373095
IsEquilateral(T) = False
T = [[0, 0], [1, 0], [0.5, 0.8660254037844386]]
perimetre(T) = 3.0
IsEquilateral(T) = True
```

Remarque : lorsqu'il s'agit juste de comparer deux distances, il suffit de comparer les carrés (en évitant l'extraction de la racine carré).

```
d2 = lambda P,Q : (P[0]-Q[0])**2+(P[1]-Q[1])**2
IsEquilateral = lambda T: abs(d2(T[0],T[1])-d2(T[0],T[2]))<1.e-10 and
↪ abs(d2(T[0],T[1])-d2(T[1],T[2]))<1.e-10

# TEST 1
P, Q, R = [0,0] , [0,1] , [1,0]
T = [P,Q,R]
print(f"{T = }")
print(f"{IsEquilateral(T) = }")

# TEST 2
P, Q, R = [0,0] , [1,0] , [0.5,(1-0.25)**0.5]
T = [P,Q,R]
print(f"{T = }")
print(f"{IsEquilateral(T) = }")
```

```
T = [[0, 0], [0, 1], [1, 0]]
IsEquilateral(T) = False
T = [[0, 0], [1, 0], [0.5, 0.8660254037844386]]
IsEquilateral(T) = True
```

④ Cercles.
Pour savoir si deux cercles se chevauchent, il suffit de vérifier que la distance entre les deux centres est inférieure à la somme de leurs rayons. On peut à nouveau éviter le calcul de la racine carrée : $d(P_1,P_2) \le r_1 + r_2$ ssi $(d(P_1,P_2))^2 \le (r_1+r_2)^2$:

```
d2 = lambda P,Q : (P[0]-Q[0])**2+(P[1]-Q[1])**2
superpose = lambda C1,C2 : d2(C1[0],C2[0]) <= (C1[1]+C2[1])**2

# TEST 1
C1, C2 = [[0,0],1] , [[1,1],1]
print(f"Cercle 1: centre ({C1[0][0]},{C1[0][1]}), rayon {C1[1]}")
print(f"Cercle 2: centre ({C2[0][0]},{C2[0][1]}), rayon {C2[1]}")
print(f"Carré de la distance des deux centres: {d2(C1[0],C2[0])}")
print(f"Carré de la somme des deux rayons: {(C1[1]+C2[1])**2}")
print(f"{superpose(C1,C2) = }")

# TEST 2
C1, C2 = [[0,0],1] , [[3,0],1]
print(f"Cercle 1: centre ({C1[0][0]},{C1[0][1]}), rayon {C1[1]}")
print(f"Cercle 2: centre ({C2[0][0]},{C2[0][1]}), rayon {C2[1]}")
print(f"Carré de la distance des deux centres: {d2(C1[0],C2[0])}")
print(f"Carré de la somme des deux rayons: {(C1[1]+C2[1])**2}")
print(f"{superpose(C1,C2) = }")
```

```
Cercle 1: centre (0,0), rayon 1
Cercle 2: centre (1,1), rayon 1
Carré de la distance des deux centres: 2
Carré de la somme des deux rayons: 4
superpose(C1,C2) = True
Cercle 1: centre (0,0), rayon 1
```

```
Cercle 2: centre (3,0), rayon 1
Carré de la distance des deux centres: 9
Carré de la somme des deux rayons: 4
superpose(C1,C2) = False
```

Exercice 7.14 (Module `math` – Distance point droite)

On se donne trois points V, W et P dans un plan cartésien. Écrire une fonction `dist_point_droite` qui prend en entrée trois listes correspondantes aux coordonnées des trois points et renvoi la distance de P de la droite qui passe par les points V et W.

Rappel : la distance du point $P = (x_p, y_p)$ de la droite r d'équation $ax + by + c = 0$ est

$$d(P, r) = \frac{|ax_p + by_p + c|}{\sqrt{a^2 + b^2}}.$$

Correction

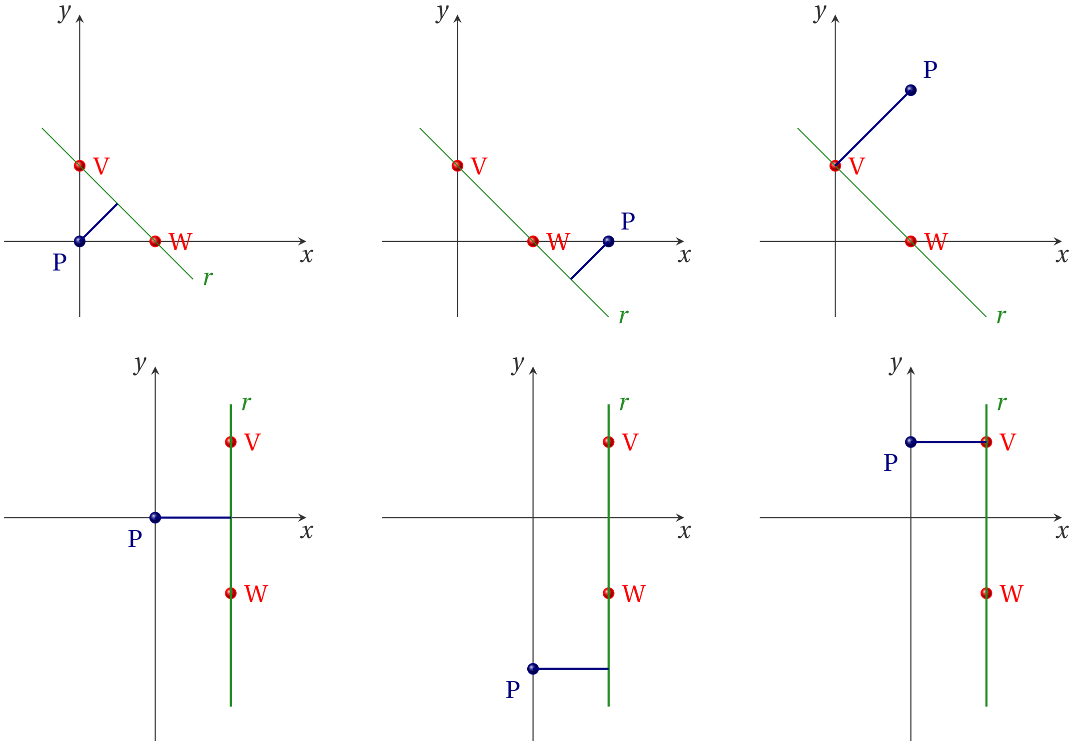
On doit alors calculer tout d'abord l'équation de la droite qui passe par les points $V = (x_v, y_v)$ et $W = (x_w, y_w)$:

$$\begin{cases} ax_v + by_v + c = 0, \\ ax_w + by_w + c = 0, \end{cases} \quad \text{ce qui équivaut à} \begin{cases} a(x_v - x_w) + b(y_v - y_w) = 0, \\ ax_w + by_w + c = 0, \end{cases}$$

Un choix possible pour a et b est $a = -(y_v - y_w)$ et $b = (x_v - x_w)$ ainsi $c = -ax_v - by_v = (x_v y_w - y_v x_w)$.

```
from math import sqrt

def dist_point_droite(V,W,P):
    a = V[1]-W[1]
    b = W[0]-V[0]
    c = V[0]*W[1]-V[1]*W[0]
    return abs(a*P[0]+b*P[1]+c)/sqrt(a**2+b**2)

TESTS = []
TESTS.append( [(0,1), (1,0), (0,0)] )
TESTS.append( [(0,1), (1,0), (2,0)] )
TESTS.append( [(0,1), (1,0), (1,2)] )
TESTS.append( [(1,1), (1,-1), (0, 0)] )
TESTS.append( [(1,1), (1,-1), (0,-2)] )
TESTS.append( [(1,1), (1,-1), (0, 1)] )
TESTS.append( [(-1,1), (1,1), ( 0,0)] )
TESTS.append( [(-1,1), (1,1), (-3,0)] )
TESTS.append( [(-1,1), (1,1), ( 3,0)] )
for V, W, P in TESTS :
    print(f"{V=}, {W=}, {P=}, d_r={dist_point_droite(V,W,P)}")
```

```
V=(0, 1), W=(1, 0), P=(0, 0), d_r=0.7071067811865475
V=(0, 1), W=(1, 0), P=(2, 0), d_r=0.7071067811865475
V=(0, 1), W=(1, 0), P=(1, 2), d_r=1.414213562373095
V=(1, 1), W=(1, -1), P=(0, 0), d_r=1.0
V=(1, 1), W=(1, -1), P=(0, -2), d_r=1.0
V=(1, 1), W=(1, -1), P=(0, 1), d_r=1.0
V=(-1, 1), W=(1, 1), P=(0, 0), d_r=1.0
V=(-1, 1), W=(1, 1), P=(-3, 0), d_r=1.0
V=(-1, 1), W=(1, 1), P=(3, 0), d_r=1.0
```



☆ Exercice Bonus 7.15 (Module `math` – Distance point segment)
On se donne trois points V, W et P dans un plan cartésien. On va complexifier l'exercice 7.14 : écrire une fonction `dist_point_segment` qui a les même paramètres en entrée et renvoi la distance de P du segment d'extrémités V et W.

Correction
Pour calculer la distance d_s du point P au segment, il faut d'abord calculer le point M projection de P sur r, c'est à dire l'intersection de la droite r avec la droite perpendiculaire qui passe par P. Cette droite a pour équation

$$-\frac{x_w - x_v}{y_w - y_v} \iff \underbrace{(x_w - x_v)}_{a_1} x + \underbrace{(y_w - y_v)}_{b_1} y + \underbrace{-\left(y_p(y_w - y_v) - x_p(x_w - x_v)\right)}_{c_1} = 0.$$

donc le point M a pour coordonnées :

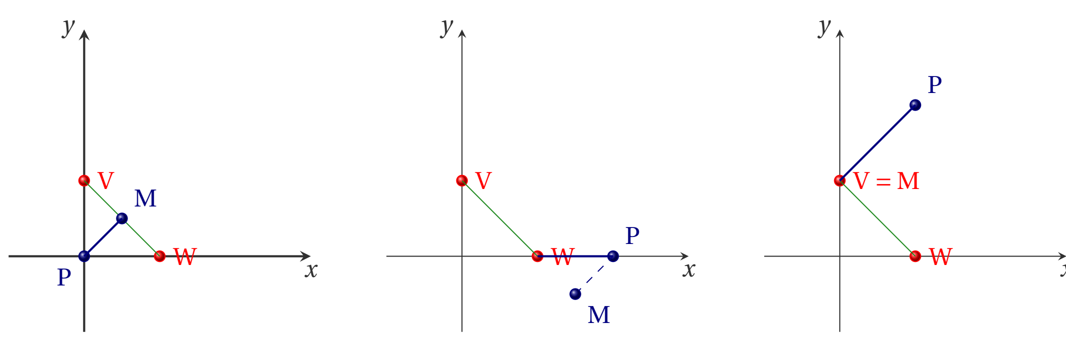
$$\begin{cases} ax + by + c = 0, \\ a_1 x + b_1 y + c_1 = 0, \end{cases} \iff (x, y) = \left(\frac{bc_1 - b_1 c}{ab_1 - a_1 b}, \frac{ca_1 - c_1 a}{ab_1 - a_1 b}\right).$$

Si M appartient au segment alors $d_s = d_r$ (distance de P de la droite r comme à l'exercice 7.14) ; sinon $d_s = \min\{d(\mathrm{P},\mathrm{W}), d(\mathrm{P},\mathrm{V})\}$.

```
from math import dist

def dist_point_segment(V,W,P):
    a = V[1]-W[1]
    b = W[0]-V[0]
    c = V[0]*W[1]-V[1]*W[0]
    # droite perpendiculaire
    a1 = W[0]-V[0]
    b1 = W[1]-V[1]
    c1 = -P[1]*(W[1]-V[1])-P[0]*(W[0]-V[0])
    # projection de P = intersection des deux droites
    M = (b*c1 - b1*c)/(a*b1 - a1*b), (-a*c1 + a1*c)/(a*b1 - a1*b)
    # M dans le segment ?
    is_bet_x = V[0]<=M[0]<=W[0] or W[0]<=M[0]<=V[0]
```

```
    is_bet_y = V[1]<=M[1]<=W[1] or W[1]<=M[1]<=V[1]
    if is_bet_x and is_bet_y :
        return  dist(P,M)
    else :
        return min( [dist(P,V),dist(P,W)] )

TESTS = []
TESTS.append( [(0,1), (1,0), (0,0)] )
TESTS.append( [(0,1), (1,0), (2,0)] )
TESTS.append( [(0,1), (1,0), (1,2)] )
TESTS.append( [(1,1), (1,-1), (0, 0)] )
TESTS.append( [(1,1), (1,-1), (0,-2)] )
TESTS.append( [(1,1), (1,-1), (0, 1)] )
TESTS.append( [(-1,1), (1,1), ( 0,0)] )
TESTS.append( [(-1,1), (1,1), (-3,0)] )
TESTS.append( [(-1,1), (1,1), ( 3,0)] )
for V, W, P in TESTS :
    print(f"{V=}, {W=}, {P=}, d_s={dist_point_segment(V,W,P)}")
```

```
V=(0, 1), W=(1, 0), P=(0, 0), d_s=0.7071067811865476
V=(0, 1), W=(1, 0), P=(2, 0), d_s=1.0
V=(0, 1), W=(1, 0), P=(1, 2), d_s=1.4142135623730951
V=(1, 1), W=(1, -1), P=(0, 0), d_s=1.0
V=(1, 1), W=(1, -1), P=(0, -2), d_s=1.4142135623730951
V=(1, 1), W=(1, -1), P=(0, 1), d_s=1.0
V=(-1, 1), W=(1, 1), P=(0, 0), d_s=1.0
V=(-1, 1), W=(1, 1), P=(-3, 0), d_s=2.23606797749979
V=(-1, 1), W=(1, 1), P=(3, 0), d_s=2.23606797749979
```



☆ Exercice Bonus 7.16 (Module `math` – Longueur d'une courbe)

Considérons le graphe de la fonction $f\colon [a,b] \to \mathbb{R}$. On peut approcher la courbe par une succession de n segments d'extrémités $(x_k, f(x_k))$ et $(x_{k+1}, f(x_{k+1}))$. La longueur de la courbe peut alors être approchée par la somme des longueurs des segments :

$$\ell_{[a,b]}(f) \approx \sum_{k=0}^{n-1} \sqrt{\left(x_{k+1}-x_k\right)^2 + \left(f(x_{k+1})-f(x_k)\right)^2}$$

Écrire une fonction `longueur_courbe` qui prend en entrée $n \geq 1$, a, b et f et renvoi la longueur $\ell_{[a,b]}(f)$.

On sait que pour $a=0$, $b=1$, $f(x)=\sqrt{1-x^2}$, alors $\ell_{[a,b]}(f)=\frac{\pi}{2}$ (c'est un quart de la circonférence d'un cercle de rayon 1).

Si on teste la fonction sur cet exemple, pour $n=1$ elle doit renvoyer $\sqrt{2}$, et pour $n\to\infty$ elle doit renvoyer une valeur qui tend vers $\frac{\pi}{2}$.

Correction

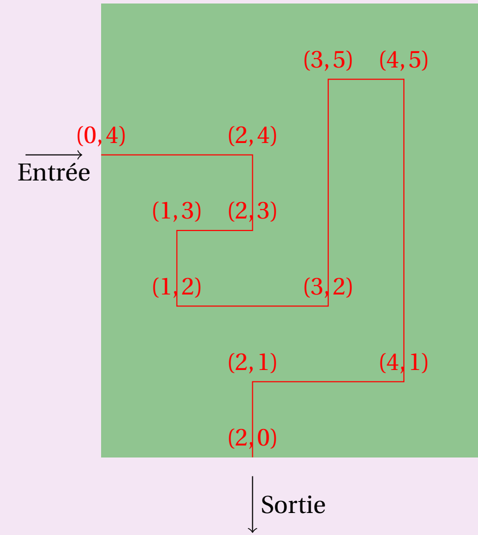
```
from math import dist, sqrt, pi
```

```
def longueur_courbe(n,a,b,f):
    h  = (b-a)/n
    xx = [ a+k*h for k in range(n+1) ]
    yy = [ f(x) for x in xx ]
    dd = [ dist( [xx[k],yy[k]] , [xx[k+1],yy[k+1]] ) for k in range(n) ]
    return sum(dd)

# TESTS
f = lambda x : sqrt(1-x**2)
for n in range(1,62,10):
    print(f"{n = }, longueur = {longueur_courbe(n,0,1,f)}")

print(f"Longueur exacte = {pi/2}")
```

```
n = 1, longueur = 1.4142135623730951
n = 11, longueur = 1.5667549487614463
n = 21, longueur = 1.569266499956974
n = 31, longueur = 1.5699437966219634
n = 41, longueur = 1.5702359675382154
n = 51, longueur = 1.5703924759972325
n = 61, longueur = 1.5704876263084087
Longueur exacte = 1.5707963267948966
```

Exercice Bonus 7.17 (Pydéfis – La biche de Cyrénée)

Histoire : pour son troisième travail, Eurysthée demanda à Hercule de capturer la biche aux pieds d'airain qui s'était enfuie de l'attelage d'Artémis. La difficulté pour Hercule était de capturer la biche sans la blesser, sous peine d'essuyer la colère d'Artémis.



Il décida donc de l'épuiser en la poursuivant dans les bois d'Oénoée. Son plan était clair : il commença par aménager les carrefours afin que la biche, à chaque embranchement, n'ait qu'un seul choix de parcours. Il construisit un plan qui ressemblait un peu à celui-ci, encore que le vrai plan était beaucoup plus grand :

Les chemins étaient tous parfaitement orthogonaux, direction Nord-Sud ou Est-Ouest. Puis il nota les positions des carrefours, dans l'ordre où la biche devait les parcourir. Dans l'exemple qui précède, il aurait noté les coordonnées de tous les points ainsi : $(0,4)$, $(2,4)$, $(2,3)$, $(1,3)$, $(1,2)$, $(3,2)$, $(3,5)$, $(4,5)$, $(4,1)$, $(2,1)$, $(2,0)$. Enfin, il dût choisir ses chaussures. Hercule étant un athlète très méthodique, sa paire de chaussures devait être choisie en fonction de la distance précise qu'il aurait à parcourir.

Problème : dans l'exemple qui précède, en suivant le parcours indiqué par les positions des carrefours, notées en kilomètres, Hercule aurait parcouru 18 kilomètres, ce que l'on peut voir sur le dessin, mais aussi calculer à partir du relevé de coordonnées. Le bois d'Oénoée était en réalité beaucoup plus grand que ce qui est indiqué précédemment. L'entrée du problème est le relevé des positions des carrefours, tous les nombres ayant été mis à la suite, sans les parenthèses, et les valeurs étant données en kilomètres. Aide Hercule en calculant pour lui la distance qu'il aura à parcourir dans le bois, à la poursuite de la biche.

Source : https://pydefis.callicode.fr/defis/Herculito03Biche/txt

Exercice 7.18 (Réduire une fraction : module `math` puis module `fractions`)

Compléter la fonction `reduce_fraction(numerator, denominator)` qui prend comme paramètres deux entiers strictement positifs représentant le numérateur et le dénominateur d'une fraction et qui renvoie le numérateur et le dénominateur de la fraction réduite. Par exemple, si les paramètres transmis à la fonction sont 6 et 63, la fonction doit renvoyer 2 et 21.

On pourra utiliser la fonction `gcd` du module `math`.

Bonus : comparer votre fonction à la fonction `Fraction` du module `fractions`.

Correction

```
# Avec le module math
from math import gcd
def reduce_fraction(numerator, denominator):
    # trouver le plus grand diviseur commun des deux nombres
    NDgcd = gcd(numerator,denominator)
    # diviser le numérateur et le dénominateur par le plus grand diviseur commun
    numerator //= NDgcd
    denominator //= NDgcd
    return (numerator, denominator)

# Avec le module fractions
from fractions import Fraction
def reduce_fraction_bis(numerator, denominator):
    return Fraction(numerator, denominator)

from random import *
TESTS = [ (6,63) , (1,2) , (25,5),  (randrange(1,100),randrange(1,100)) ]
for N,D in TESTS:
    reduced_numerator,reduced_denominator = reduce_fraction(N,D)
    print(f'{N}/{D} = {reduced_numerator}/{reduced_denominator} (avec le module math) et =
     {reduce_fraction_bis(N,D)} avec le module fractions')
```

```
6/63 = 2/21 (avec le module math) et = 2/21 avec le module fractions
1/2 = 1/2 (avec le module math) et = 1/2 avec le module fractions
25/5 = 5/1 (avec le module math) et = 5 avec le module fractions
64/90 = 32/45 (avec le module math) et = 32/45 avec le module fractions
```

Exercice Bonus 7.19 (Module `Fraction` – Triangles semblables)

Considérons deux listes contenant chacune trois tuples représentant les sommets d'un triangle. Écrire une fonction qui renvoie `True` si les deux triangles sont semblables, `False` sinon.

Source : https://py.checkio.org/en/mission/similar-triangles/

Correction

Tester `x == flottant` est presque toujours une erreur, on utilisera plutôt `abs(x-flottant)<1.e-10` par exemple.

Si on suppose que les coordonnées sont des valeurs dans $\mathbb{Q}$, on peut comparer le carré des distances (en évitant les racines carrées) et utiliser le module `Fraction` pour comparer les rapports, comme dans l'exemple ci-dessous :

```
from fractions import Fraction

def similar_triangles(T,V):
    dist2 = lambda p1,p2 : (p1[0]-p2[0])**2+(p1[1]-p2[1])**2
    a1,b1,c1 = sorted( [dist2(T[0],T[1]) , dist2(T[1],T[2]) , dist2(T[2],T[0])]  )
    a2,b2,c2 = sorted( [dist2(V[0],V[1]) , dist2(V[1],V[2]) , dist2(V[2],V[0])]  )
    return Fraction(a2,a1)==Fraction(b2,b1)==Fraction(c2,c1)
    # sans le module Fraction on écrira
    # toll = 1.e-12
```

```
    # return abs(a2*b1-a1*b2)<toll and abs(b2*c1-b1*c2)<toll and abs(a2*c1-a1*c2)<toll

# TESTS
TEST=[]
TEST.append( ( [(0, 0), (1, 2), (2, 0)] , [(3, 0), (4, 2), (5, 0)] ) )
TEST.append( ( [(0, 0), (1, 2), (2, 0)] , [(3, 0), (4, 3), (5, 0)] ) )
TEST.append( ( [(0, 0), (1, 2), (2, 0)] , [(2, 0), (4, 4), (6, 0)] ) )
TEST.append( ( [(0, 0), (0, 3), (2, 0)] , [(3, 0), (5, 3), (5, 0)] ) )
TEST.append( ( [(1, 0), (1, 2), (2, 0)] , [(3, 0), (5, 4), (5, 0)] ) )
TEST.append( ( [(1, 0), (1, 3), (2, 0)] , [(3, 0), (5, 5), (5, 0)] ) )

print(*[similar_triangles(T,V) for T,V in TEST],sep=", ")
```

```
True, False, True, True, True, False
```

Exercice 7.20 (Module `random` - Jeu de dé)
On lance un dé. Si le numéro est 1, 5 ou 6, alors c'est gagné, sinon c'est perdu. Écrire un programme simulant ce jeu 8 fois et qui affiche "gagné" ou "perdu".

Simuler le jeu 10000 fois et compter combien de fois on a gagné.

Correction
On simule 10 tentatives :

```
from random import randint
T = 8
Tirages = [randint(1,6) for t in range(T)]
for t in Tirages:
    if t in [1,5,6] :
        print('Gagné!')
    else:
        print('Perdu')
```

```
Perdu
Perdu
Perdu
Perdu
Gagné!
Gagné!
Perdu
Perdu
```

On simule 10000 tentatives et on affiche seulement combien de fois on a gagné :

```
from random import randint
T = 10000
Tirages = [randint(1,6) for t in range(T)]

#Gagne=0
#for t in Tirages:
#    if t in [1,5,6] :
#        Gagne+=1

# ou de manière équivalente
# Gagne = sum([1 for t in Tirages if t in [1,5,6]])
# qu'on peut réécrire comme
Gagne = sum([t in [1,5,6] for t in Tirages])

print(f"Sur {T} tirages on a gagné {Gagne} fois.")
```

```
Sur 10000 tirages on a gagné 5088 fois.
```

One-liner :

```
from random import randint
T = 10000
Gagne=sum( [ randint(1,6) in [1,5,6]   for t in range(T) ] )
print(f"Sur {T} tirages on a gagné {Gagne} fois.")
```

```
Sur 10000 tirages on a gagné 5018 fois.
```

Exercice 7.21 (Module `random` - Calcul de fréquences)
On tire 10000 nombres au hasard dans l'intervalle $[0,1]$. À chaque tirage, on se demande si l'événement A : «le nombre tiré au hasard appartient à l'intervalle $[1/7,5/6]$» est réalisé.

1. Combien vaut la probabilité p de A ?
2. Calculer la fréquence de réalisation de A.

Correction
L'énoncé ne précise pas selon quel hasard les nombres sont tirés dans l'intervalle $[0,1]$. En fait, il est sous-entendu qu'ils sont tirés selon la loi uniforme sur $[0,1]$. Dans ces conditions, par définition de cette loi, la probabilité de l'événement A est la longueur de $[1/7,5/6]$, soit $p = 5/6 - 1/7 = 29/42$.

```
from random import random
T = 10000
Tirages = [random() for _ in range(T)] # _ car variable muette
OuiA = [1 for t in Tirages if 1/7<t<5/6]
print(f"Fréquence théorique = {29/42}, Fréquence moyenne = {sum(OuiA)/T}")
```

```
Fréquence théorique = 0.6904761904761905, Fréquence moyenne = 0.6886
```

Exercice 7.22 (Module `random` - Puce)
Une puce fait des bonds successifs indépendants les uns des autres en ligne droite. La longueur de chaque bond est aléatoire. On considère que c'est un nombre choisi au hasard dans l'intervalle $[0,5[$, l'unité de longueur étant le cm. On note la distance totale parcourue au bout de m bonds. On répète cette expérience n fois. Calculer la distance moyenne parcourue au cours de ces n expériences.

Correction
Avec le script

```
from random import  randint
n = 1000 # nombre de tirage
m = 20 # nombre de bond par tirage
L = [ sum([randint(0,5) for i in range(m)]) for j in range(n)]
print(sum(L)/n)
```

on trouve

```
49.799
```

Explications :

- `[randint(0,5) for i in range(m)]` est une liste qui contient les m bonds d'une expérience. En faisant la somme, on obtient la distance parcourue lors de cette expérience.
- `L[j]` contient donc la distance parcourue à la j-ème expérience.
- On voit que la distance moyenne parcourue tend (*i.e.* si on augment n) vers $\frac{(5-0)}{2}m$.

Exercice Bonus 7.23 (Module `random` - Le nombre mystère)
Écrire un script où l'ordinateur choisit un nombre mystère entre 0 et 100 (inclus) au hasard et le joueur doit deviner ce nombre en suivant les indications «plus grand» ou «plus petit» données par l'ordinateur. Le joueur a sept tentatives pour trouver la bonne réponse. Programmer ce jeu.

Pour que l'ordinateur choisisse un entier aléatoirement dans l'intervalle $[0;100]$ on écrira

```
import random
N = random.randint(0,100)
```

Pour affecter à la variable `j` la valeur que le joueur tape au clavier on écrira

```
j=int(input('Quel nombre proposes-tu ?'))
```

Correction

```
import random
N=random.randint(0,100)

j=-1
i=0
while j!=N and i<7:
    i+=1
    j=int(input('Quel nombre proposes-tu ? '))
    if j>N:
        print('Le nombre à trouver est plus petit')
    elif j<N:
        print('Le nombre à trouver est plus grand')
    else:
        print('Bravo')
        break
    if i==7:
        print('Dommage')
        break
print(f"N={N}, J={j}, Tentatives={i}")
```

La techniques optimale pour jouer contre l'ordinateur est d'utiliser une méthode dichotomique :

- on pose $a=0$, $b=100$ et on propose $c=\mathrm{E}\left(\frac{a+b}{2}\right)=50$,
- si le nombre mystère est plus petit que c on posera $b=c$, sinon $a=c$ et on recommence avec $c=\mathrm{E}\left(\frac{a+b}{2}\right)$.

Cette stratégie se termine lorsque $b=a$. Comme à chaque étape l'intervalle de recherche est divisé en deux parties, si initialement on a $b-a=d$, après n étapes on aura $b-a=d\times 2^{-n}$ ainsi $d\times 2^{-n}\le 1$ pour $n\ge \log_2(d)$. Ici $d=100$ et $\log_2(d)=6.643856189774725$: en 7 étapes (n de 0 à 6) on trouvera forcement le nombre mystère.

☆ Exercice Bonus 7.24 (Module `random` - Yahtzee)
Écrire un script où l'ordinateur simule le lancé de trois dés en même temps et il continue tant que il obtient un "yahtzee" (les trois dés obtiennent tous les trois 6) ou si le nombre de lancés sans obtenir de yahtzee est supérieur à 100.

Correction

```
from random import randint
d1, d2, d3 = 0, 0, 0
count = 1
while (d1+d2+d3 < 18 and count <= 100):
    d1 = randint(1,6)
    d2 = randint(1,6)
    d3 = randint(1,6)
    count += 1
    # print(f"{d1}, {d2}, {d3}")

if d1+d2+d3 == 18:
    print (f"Yahtzee avec {count-1} tentatives")
else:
    print( "no Yahtzee :( ")
```

Bonus : on peut imprimer les dés :

```
print([chr(0x2680+i) for i in range(6)])
```

Exercice 7.25 (Module `math` – Cylindre)
Fabriquer une fonction qui calcule le volume d'un cylindre de révolution de hauteur h et dont la base est un disque de rayon r.

Correction
On écrit la fonction soit avec `def` soit avec une `lambda` function, puis on valide la fonction avec un test dont on connaît le résultat :

```
>>> import math
>>> volume = lambda h,r : (r**2 * h * math.pi)
>>> h,r = 1,1
>>> print(f'h = {h} cm, r = {r} cm, v = {volume(h,r):.5f} cm2')
h = 1 cm, r = 1 cm, v = 3.14159 cm2
```

Exercice 7.26 (Module `math` – Formule d'Héron)
Pour le calcul de l'aire $\mathscr{A}(\mathrm{T})$ d'un triangle T de coté a, b et c, on peut utiliser la formule d'Héron :

$$\mathscr{A}(\mathrm{T}) = \sqrt{p(p-a)(p-b)(p-c)}$$

où p est le demi-périmètre de T. Écrire une fonction qui implémente cette formule. La tester en la comparant à la solution exacte (calculée à la main).

Correction
On choisi d'utiliser la fonction `sqrt()` du module `math`. Il est cependant possible d'utiliser `**0.5` sans utiliser d'autres modules.

```
import math
def Heron(a,b,c):
	p = (a+b+c)/2
	return math.sqrt(p*(p-a)*(p-b)*(p-c))

# TEST-1
(a,b,c) = (5,4,3)
He = Heron(a,b,c)
Ex = b*c/2
print(f"Erone = {He:.5f}, Exacte = {Ex:.5f}")

# TESTS
# on choisit des cas où le calcul de la solution exacte est simple
for a,b,c in [(5,2.5,2.5),(5,3,3),(5,5,5)]:
    He = Heron(a,b,c)
    Ex = a/2*math.sqrt(b**2-(a/2)**2)
    Err = He-Ex
    print(f"Erone = {He:.5f}, Exacte = {Ex:.5f}, Erreur = {Err:.5f}")
```

```
Erone = 6.00000, Exacte = 6.00000
Erone = 0.00000, Exacte = 0.00000, Erreur = 0.00000
Erone = 4.14578, Exacte = 4.14578, Erreur = 0.00000
Erone = 10.82532, Exacte = 10.82532, Erreur = -0.00000
```

Exercice 7.27 (Module `math` – Formule de Kahan)
Pour le calcul de l'aire $\mathscr{A}(\mathrm{T})$ d'un triangle T de coté a, b et c avec $a \ge b \ge c$, William Kahan a proposé une formule plus stable que celle d'Héron :

$$\mathrm{S} = \frac{1}{4}\sqrt{\left[a+(b+c)\right]\left[c-(a-b)\right]\left[c+(a-b)\right]\left[a+(b-c)\right]}.$$

Écrire une fonction qui implémente cette formule. La tester en la comparant à la solution exacte (calculée à la main).

Correction

```
import math
def Khan(a,b,c):
    c,b,a = sorted([a,b,c])
    return 0.25*math.sqrt( (a+(b+c))*(c-(a-b))*(c+(a-b))*(a+(b-c))  )

# TEST-1
(a,b,c) = (5,4,3)
He = Khan(a,b,c)
Ex = b*c/2
print(f"Khan={He:.5f}, Exacte={Ex:.5f}")

# TESTS
# on choisit des cas où le calcul de la solution exacte est simple
for a,b,c in [(2.5,5,2.5),(5,3,3),(5,5,5)]:
    He = Khan(a,b,c)
    Ex = a/2*math.sqrt(b**2-(a/2)**2)
    Err = He-Ex
    print(f"Khan = {He:.5f}, Exacte = {Ex:.5f}, Erreur = {Err:.5f}")
```

```
Khan=6.00000, Exacte=6.00000
Khan = 0.00000, Exacte = 6.05154, Erreur = -6.05154
Khan = 4.14578, Exacte = 4.14578, Erreur = 0.00000
Khan = 10.82532, Exacte = 10.82532, Erreur = -0.00000
```

Exercice 7.28 (Module `math` – Cercle circonscrit)

Écrire (et tester) une fonction qui prend en entrée la longueur des trois cotés d'un triangle et renvoie le rayon du cercle circonscrit.

Rappel : si on note a, b et c les longueurs des trois cotés du triangle et $\mathscr{A}$ l'aire, alors le rayon du cercle circonscrit est donné par

$$r = \frac{abc}{4\mathscr{A}}.$$

Correction

On va écrire d'abord une fonction qui calcule l'aire du triangle à partir des longueurs des trois cotés (par exemple avec la formule d'Héron, *cf.* exercice 7.26), puis écrire une fonction qui calcule le rayon.

```
import math

def Heron(a,b,c):
    p = (a+b+c)/2
    return math.sqrt(p*(p-a)*(p-b)*(p-c))
    
def circumscribed(a,b,c):
    A = Heron(a,b,c)
    return a*b*c/(4*A)

# TEST-1
(a,b,c) = (1,1,1)
r = circumscribed(a,b,c)
Ex = 1/math.sqrt(3)
print(f"r = {r:.5f}, Exacte = {Ex:.5f}")

# TEST-2
(a,b,c) = (3,4,5)
```

```
r = circumscribed(a,b,c)
Ex = 2.5
print(f"r = {r:.5f}, Exacte = {Ex:.5f}")

# TEST-3
(a,b,c) = (1,1,math.sqrt(3))
r = circumscribed(a,b,c)
Ex = 1
print(f"r = {r:.5f}, Exacte = {Ex:.5f}")


# TEST-4
(a,b,c) = (2,2,math.sqrt(7))
r = circumscribed(a,b,c)
Ex = 4/3
print(f"r = {r:.5f}, Exacte = {Ex:.5f}")
```

```
r = 0.57735, Exacte = 0.57735
r = 2.50000, Exacte = 2.50000
r = 1.00000, Exacte = 1.00000
r = 1.33333, Exacte = 1.33333
```

☆ Exercice Bonus 7.29 (Centre et rayon d'un cercle pour un arc donné)

L'objectif de cet exercice est d'écrire une fonction qui renvoie le centre et le rayon d'un cercle, à partir de deux points sur un arc et l'angle entre eux.

```
def find_center(p1, p2, angle):
    # Votre code ici
    return xc, yc, r
```

Entrées : `p1` et `p2` sont deux liste contenantes les coordonnées $[x, y]$ des deux points sur l'arc du cercle, `angle` est l'angle entre les deux points (en degrés).

Sortie : `xc` et `yc` sont les coordonnées du centre du cercle, `r` son rayon.

Correction

```
import math

def find_center(p0, p1, angle):

    x0, y0 = p0
    x1, y1 = p1
    a = math.radians(angle)

    # Distance entre les points p1 et p2 et calcul du rayon
    d_p1p0 = math.sqrt((x1 - x0)**2 + (y1 - y0)**2)
    r = d_p1p0 / (2 * math.sin(a / 2))

    # Pente de la droite passant par la corde
    slope = (y1 - y0) / (x1 - x0) if x1 != x0 else float('inf')

    # Pente de la droite perpendiculaire à la corde
    new_slope = -1 / slope if slope != 0 else float('inf')

    # Point sur la droite perpendiculaire à la corde
    # passant par le centre du cercle
    xm, ym = (x1 + x0) / 2 , (y1 + y0) / 2

    # Distance entre (xm,ym) et le centre du cercle (xc, yc)
```

```
    d_cm = d_p1p0 / (2 * math.tan(a))

    if math.isinf(new_slope):
        xc = xm
        yc = ym - d_cm
    else:
        xc = xm - (d_cm) / math.sqrt(new_slope**2 + 1)
        yc = new_slope * (xc - xm) + ym

    return xc, yc, r

# TESTS
TESTS = [ ([1,0],[0,1],90) , ([0,1],[1,0],270) , ([1,1],[-1,1],45)  ]

for p1,p2,angle in TESTS:
    xc,yc,r = find_center(p1, p2, angle)
    print( f"{p1=}, {p2=}, {angle=} ⇝ {xc=:g}, {yc=:g}, {r=:g} " )
```

```
p1=[1, 0], p2=[0, 1], angle=90 ⇝ xc=0.5, yc=0.5, r=1
p1=[0, 1], p2=[1, 0], angle=270 ⇝ xc=0.5, yc=0.5, r=1
p1=[1, 1], p2=[-1, 1], angle=45 ⇝ xc=0, yc=-2.22045e-16, r=2.61313
```

Exercice 7.30 (Module `random` - Distance moyenne entre deux points aléatoires d'un carré)
Considérons un carré de côté 1 et plaçons en son intérieur deux points de manière aléatoire. Quelle est la distance moyenne entre deux points ?

Correction

```
from random import random
N = 1000
points = [ [random(),random(),random(),random()] for i in range(N) ]
distan = [ ((points[i][0]-points[i][1])**2+(points[i][2]-points[i][3])**2)**0.5 for i in
    range(N) ]
print(f"Avec {N} couples de points, en moyenne les deux points de chaque couple sont à une
    distance de {sum(distan)/len(distan)} l'un de l'autre.")
```

```
Avec 1000 couples de points, en moyenne les deux points de chaque couple sont à une distance
    de 0.521012560254331 l'un de l'autre.
```

Depuis la version 3.8 de python, la fonction `dist()` a été ajoutée dans le module `math`. On pourra alors écrire

```
from random import random
from math import dist
N = 1000
points = [ [random(),random(),random(),random()] for i in range(N) ]
distan = [ dist( (points[i][0],points[i][1]) , (points[i][2],points[i][3]) ) for i in range(N)
    ]
print(f"Avec {N} couples de points, en moyenne les deux points de chaque couple sont à une
    distance de {sum(distan)/len(distan)} l'un de l'autre.")
```

```
Avec 1000 couples de points, en moyenne les deux points de chaque couple sont à une distance
    de 0.5110808380929007 l'un de l'autre.
```

Exercice 7.31 (Module `random` - Kangourou)
Un kangourou fait habituellement des bonds de longueur aléatoire comprise entre 1 et 9 mètres.[a]

- Combien de sauts devra-t-il faire pour parcourir 2000 mètres ?
- Fabriquer une fonction qui à toute distance d exprimée en mètres associe le nombre aléatoire N de sauts nécessaires pour la parcourir.

- Calculer le nombre moyen de sauts effectués par le kangourou quand il parcourt T fois la distance d.

cf. http://gradus-ad-mathematicam.fr/documents/300_Directeur.pdf

a. Il est sous-entendu que la longueur de chaque bond est la valeur prise par une variable aléatoire qui suit la loi uniforme entre 0 et 9. Il est aussi sous-entendu, comme d'habitude, que si on considère des sauts successifs, les variables aléatoires qui leur sont associées sont indépendantes.

Correction

```
from random import randint

def distanceTOpas(d):
	l = 0 # distance parcourue avant le premier saut
	b = 0 # nombre de sauts effectues avant le premier saut
	while l<=d:
		l += randint(1,9)
		b += 1
	return b,l

d = 2000
N,L = distanceTOpas(d)
print("Pour parcourir d =",d," mètres une fois il faut N =",N," bonds.")
print("La distance réelle parcourue est de ",L," mètres.")

T = 100
NN = [distanceTOpas(d)[0] for N in range(T)]
print("Pour parcourir d =",d," mètres ",T," fois il faut en moyenne N =",sum(NN)/T," bonds")
```

```
Pour parcourir d = 2000  mètres une fois il faut N = 403  bonds.
La distance réelle parcourue est de  2002  mètres.
Pour parcourir d = 2000  mètres  100  fois il faut en moyenne N = 400.47  bonds
```

Exercice 7.32 (Module math – sin cos)

En important seulement les fonctions nécessaires du module `math`, écrire un script qui vérifie la formule suivante pour $n = 10$:

$$\sum_{k=0}^{n} \cos(kx) = \frac{1}{2} + \frac{\sin\left(\frac{2n+1}{2}x\right)}{2\sin\left(\frac{x}{2}\right)}$$

Comparer en un point au choix.

Correction

```
>>> from math import sin, cos, pi
>>> n = 10
>>> L = lambda x : sum([cos(k*x) for k in range(n+1)])
>>> R = lambda x : 0.5*(1+sin(0.5*(2*n+1)*x)/sin(0.5*x))
>>> print(L(1),R(1))
-0.4174477464559059 -0.417447746455906
>>> # On peut évaluer la valeur absolue de la différence en plusieurs points:
>>> print([abs(L(x)-R(x)) for x in range(1,5)])
[1.1102230246251565e-16, 0.0, 1.1102230246251565e-16, 2.220446049250313e-16]
```

Exercice 7.33 (Approximations de ln)

L'algorithme de Briggs est une méthode qui permet de calculer de manière approchée le logarithme d'un nombre.

Soit x un réel strictement positif. Pour approcher $\ln(x)$ on utilise la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$\begin{cases} u_0 = x, \\ u_{n+1} = \sqrt{u_n}. \end{cases}$$

On peut montrer que la suite $(w_n)_{n\in\mathbb{N}}$ définie par $w_n = 2^n(u_n - 1)$ converge vers $\ln(x)$.

Écrire une fonction qui, pour x donné, renvoie la valeur w_N ; N doit être le plus petit entier tel que $|u_N - 1| > 10^{-12}$. Comparer ensuite pour différentes valeurs de x la valeur approchée obtenue par cette fonction et la valeurs `math.log(x)` du module `math`.

https://www.mathoutils.fr/grand-oral-mathematiques/grand-oral-logarithme/

Correction

```python
def briggs(x,epsilon):
    n = 0
    u = x
    w = u-1
    while abs(u-1)>epsilon:
        n +=1
        u = u**0.5
        w = 2**n*(u-1)
    return w

# TESTS
from math import log
from tabulate import tabulate
T = []
T.append(["x","Briggs","math.log","erreur"])
for n in range(1,10):
    b = briggs(n,1.e-12)
    m = log(n)
    e = b-m
    T.append([n,b,m,e])

print(tabulate(T, headers="firstrow", floatfmt=".15f"))
```

x	Briggs	math.log	erreur
1	0.000000000000000	0.000000000000000	0.000000000000000
2	0.693115234375000	0.693147180559945	-0.000031946184945
3	1.098388671875000	1.098612288668110	-0.000223616793110
4	1.386230468750000	1.386294361119891	-0.000063892369891
5	1.609375000000000	1.609437912434100	-0.000062912434100
6	1.791503906250000	1.791759469228055	-0.000255562978055
7	1.945800781250000	1.945910149055313	-0.000109367805313
8	2.079101562500000	2.079441541679836	-0.000339979179836
9	2.196777343750000	2.197224577336220	-0.000447233586220

Exercice 7.34 (Approximations de π)

Dans cet exercice nous allons étudier et mettre en œuvre certaines méthodes qui peuvent être utilisées pour calculer les premiers chiffres du nombre irrationnel π. Des méthodes sont basées sur des tirages aléatoires, les autres sont des algorithmes itératifs.

La valeur de π considérée comme exacte sera celle du module `math`.

Méthode 1 : Fractions

Évaluer les approximations suivantes de π et en donner la précision

$$\frac{22}{7},\qquad \frac{355}{113},\qquad 3+\frac{8}{60}+\frac{29}{60^2}+\frac{44}{60^3}.$$

Bonus : vérifier que l'erreur entre l'approximation $\frac{22}{7}$ et la valeur exacte π vaut

$$\int_0^1 \frac{x^4(1-x)^4}{1+x^2}\,\mathrm{d}x.$$

Méthode 2 : Racines
Évaluer les approximations suivantes de π et en donner la précision

$$\sqrt{2}+\sqrt{3},\qquad \sqrt[3]{31},\qquad \sqrt{63}-\sqrt{23},\qquad \sqrt{51}-\sqrt{16},\qquad \sqrt[15]{28658146},\qquad \sqrt[4]{\frac{2143}{22}}.$$

Méthode 3 : Formule de Liu Hui

$$\pi \approx 768\sqrt{2-\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+1}}}}}}}}}$$

Méthode 4 : Logarithme

$$\pi \approx \frac{\ln(640320^3+744)}{\sqrt{163}}$$

Méthode 5 : Formule de Basel https://en.wikipedia.org/wiki/Basel_problem

$$\lim_{N\to+\infty} s(N)=\frac{\pi^2}{6},\qquad \text{avec}\qquad s(N)\stackrel{\text{déf}}{=}\sum_{n=1}^{N}\frac{1}{n^2}$$

- Écrire une fonction qui, pour N donné, renvoie $s(N)$.
- Écrire un script qui calcule pour quelles valeurs de N on obtient une approximation de $\frac{\pi^2}{6}$ à 10^{-5} près.

Méthode 6 : Formule de Madhava–Leibniz

$$\pi = 4\lim_{N\to+\infty} s(N)\qquad \text{avec}\qquad s(N)\stackrel{\text{déf}}{=}\sum_{n=0}^{N}\frac{(-1)^n}{2n+1}.$$

- Écrire une fonction qui, pour N donné, renvoie $s(N)$.
- Écrire un script qui calcule pour quelles valeurs de N on obtient une approximation de π à 10^{-3} près. Même exercice à 10^{-4} près puis 10^{-5} près.

Méthode 7 : Formule de Madhava

$$\pi = \sqrt{12}\lim_{N\to+\infty} s(N)\qquad \text{avec}\qquad s(N)\stackrel{\text{déf}}{=}\sum_{n=0}^{N}\frac{1}{(-3)^n(2n+1)}.$$

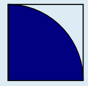
- Écrire une fonction qui, pour N donné, renvoie $s(N)$.
- Écrire un script qui calcule pour quelles valeurs de N on obtient une approximation de π à 10^{-3} près. Même exercice à 10^{-4} près puis 10^{-5} près.

Méthode 8 : Formule de Bailey-Borwein-Plouffe

$$\pi = \lim_{N\to+\infty} s(N)\qquad \text{avec}\qquad s(N)\stackrel{\text{déf}}{=}\sum_{n=0}^{N}16^{-n}\left(\frac{4}{8n+1}-\frac{2}{8n+4}-\frac{1}{8n+5}-\frac{1}{8n+6}\right).$$

- Écrire une fonction qui, pour N donné, renvoie $s(N)$.
- Écrire un script qui calcule pour quelles valeurs de N on obtient une approximation de π aussi précise que celle fournie par la variable π du module `math`.

Méthode 9 : Aire d'un cercle (Monte-Carlo)
Il est possible de calculer les premières décimales de π avec l'aide du hasard.
On considère un carré de coté 1 et un cercle de rayon 1 centré à l'origine. Si on divise l'aire de la portion de disque par celle du carré on trouve $\frac{\pi}{4}$. Si on tiré au hasard un point dans le carré, on a une probabilité de $\frac{\pi}{4}$ que le point soit dans la portion de disque.



On considère l'algorithme suivant pour approcher π : on génère N couples $\{(x_k, y_k)\}_{k=0}^{N-1}$ de nombres aléatoires (avec une distribution uniforme) dans l'intervalle $[0,1]$, puis on calcule le nombre $m \le N$ de ceux qui se trouvent dans le premier quart du cercle unité.

Écrire un programme pour calculer cette suite et observer comment évolue l'erreur quand N augmente sachant que π est la limite de la suite $4m/N$ lorsque $N \to +\infty$.

Méthode 10 : Intégrale double
On peut montrer que

$$\pi = \int_0^1 \left(\int_0^{\sqrt{1-x^2}} 4 \right) \mathrm{d}y \, \mathrm{d}x$$

Utiliser `scipy.integrate` pour approcher cette intégrale.

Méthode 11 : Suite de Borwein
Considérons les suites $\{a_n\}_{n\in\mathbb{N}}$, $\{s_n\}_{n\in\mathbb{N}}$ et $\{r_n\}_{n\in\mathbb{N}}$ ainsi construites :

$$\begin{cases} s_0 = \frac{\sqrt{3}-1}{2}, \\ r_0 = \frac{3}{1+2\sqrt[3]{1-s_n^3}}, \\ a_0 = \frac{1}{3}; \end{cases} \qquad \begin{cases} r_{n+1} = \frac{3}{1+2\sqrt[3]{1-s_n^3}}, \\ s_{n+1} = \frac{r_{n+1}-1}{2}, \\ a_{n+1} = r_{n+1}^2 a_n - 3^n (r_{n+1}^2 - 1). \end{cases}$$

On peut montrer que

$$\pi = \lim_{n\to\infty} 1/a_n$$

Écrire un script qui affiche a_n pour $n = 0, \dots, 3$ et calcule l'erreur.

Méthode 12 : Suite de Jonathan-Borwein
Soit $g(x) = \sqrt[4]{1-x^4}$ et considérons les deux suites $\{y_n\}_{n\in\mathbb{N}}$, et $\{a_n\}_{n\in\mathbb{N}}$ ainsi construites :

$$\begin{cases} y_0 = \sqrt{2} - 1, \\ a_0 = 6 - 4\sqrt{2}; \end{cases} \qquad \begin{cases} y_{n+1} = \frac{1-f(y_n)}{1+f(y_n)}, \\ a_{n+1} = a_n (1 + y_{n+1})^4 - 2^{2n+3} y_{n+1} (1 + y_{n+1} + y_{n+1}^2). \end{cases}$$

On peut montrer que

$$\pi = \lim_{n\to\infty} 1/a_n$$

Écrire un script qui affiche a_n pour $n = 0, \dots, 3$ et calcule l'erreur.

Méthode 13 : Suite de Gauss-Legendre
Considérons les quatre suites $\{a_n\}_{n\in\mathbb{N}}$, $\{b_n\}_{n\in\mathbb{N}}$,$\{p_n\}_{n\in\mathbb{N}}$, $\{t_n\}_{n\in\mathbb{N}}$ ainsi construites :

$$\begin{cases} a_0 = 1, \\ b_0 = 1/\sqrt{2}, \\ p_0 = 1, \\ t_0 = 1/4; \end{cases} \qquad \begin{cases} a_{n+1} = \frac{a_n + b_n}{2}, \\ b_{n+1} = \sqrt{a_n b_n}, \\ p_{n+1} = 2 p_n, \\ t_{n+1} = t_n - p_n (a_n - a_{n+1})^2. \end{cases}$$

On peut montrer que

$$\pi = \lim_{n\to\infty} f(n) \qquad \text{avec} \qquad f(n) = \frac{(a_{n+1} + b_{n+1})^2}{4 t_{n+1}}.$$

Écrire un script qui affiche $f(n)$ pour $n = 0, \dots, 3$ et calcule l'erreur.

Méthode 14 : Le module `decimal`
Les méthodes précédentes utilisent des nombres flottants pour toutes les variables et la somme partielle, ainsi la précision finale de l'approximation de π sera extrêmement limitée (et ne sera pas meilleure que d'importer `math.pi`). On peut essayer d'utiliser des nombres décimaux à la place, en important le module `decimal`. Comprendre le code suivant :

© G. Faccanoni

```
from decimal import *
mypi = Decimal (22) / Decimal (7)
print(mypi)
```

Lire la documentation du module pour améliorer la précision.

Source : https://en.wikipedia.org/wiki/Approximations_of_%CF%80

Correction

Pour une meilleur affichage, on pourra utiliser la fonction `tabulate` du module `tabulate`.

Méthode 1 : Fractions

```
from tabulate import tabulate
T = []
T.append( ["Formule" , "Approximation", "Erreur"] )

from math import pi
for formule in [ "22/7" , "355/113", "3 + 8/60 + 29/60**2 + 44/60**3" ]:
	approx = eval(formule)
	T.append( [formule, approx, approx-pi] )

print(tabulate(T,headers="firstrow",floatfmt="0.9f"))
```

```
Formule                           Approximation        Erreur
------------------------------  ---------------  ------------
22/7                                3.142857143   0.001264489
355/113                             3.141592920   0.000000267
3 + 8/60 + 29/60**2 + 44/60**3      3.141592593  -0.000000061
```

Bonus : on peut calculer une primitive (et ensuite l'intégrale) analytiquement (à la main ou avec le module sympy) ou utiliser une valeur approchée de l'intégrale obtenue avec la fonction `scipy.integrate.quad` :

```
func = lambda x: x**4*(1-x)**4/(1+x**2)

# VALEURE EXACTE  DE L'INTEGRALE
import sympy as sp
x = sp.Symbol('x')
primitive = sp.integrate(func(x), x)
print(f"Une primitive est {primitive}")
integrale_exacte = primitive.subs(x, 1) - primitive.subs(x, 0)
print(f"donc l'integrale vaut {integrale_exacte} ≈  {integrale_exacte.evalf()}")

# VALEURE APPROCHÉE   DE L'INTEGRALE
import numpy as np
from scipy.integrate import quad
val, err = quad(func, 0, 1)
print(f"L'integrale vaut approximativement {val}")
```

```
Une primitive est x**7/7 - 2*x**6/3 + x**5 - 4*x**3/3 + 4*x - 4*atan(x)
donc l'integrale vaut 22/7 - pi ≈  0.00126448926734962
L'integrale vaut approximativement 0.0012644892673496185
```

Méthode 2 : Racines

```
from tabulate import tabulate
T = []
T.append( ["Formule" , "Approximation", "Erreur"] )

from math import pi, sqrt
for formule in [ "sqrt(2)+sqrt(3)" , "31**(1/3)", "sqrt(63)-sqrt(23)",
 ↪ "sqrt(51)-sqrt(16)", "28658146**(1/15)", "(2143/22)**(1/4)" ]:
	approx = eval(formule)
```

```
    T.append( [formule, approx, approx-pi] )

print(tabulate(T,headers="firstrow",floatfmt="0.12f"))
```

```
Formule                  Approximation           Erreur
-----------------  ---------------  ---------------
sqrt(2)+sqrt(3)     3.146264369942   0.004671716352
31**(1/3)           3.141380652391  -0.000212001198
sqrt(63)-sqrt(23)   3.141422409881  -0.000170243709
sqrt(51)-sqrt(16)   3.141428428543  -0.000164225047
28658146**(1/15)    3.141592653814   0.000000000224
(2143/22)**(1/4)    3.141592652583  -0.000000001007
```

Méthode 3 : Formule de Liu Hui

```
from math import sqrt
pi_approx = 768 * sqrt(2 - sqrt(2 + sqrt(2 + sqrt(2 + sqrt(2 + sqrt(2 + sqrt(2 +
    sqrt(2 + sqrt(2+1)))))))))
print(f"pi_approx={pi_approx:0.6f}, erreur={pi_approx-pi:0.6f}")
```

```
pi_approx=3.141590, erreur=-0.000002
```

Méthode 4 : Logarithme

```
from math import pi, sqrt, log
formule = "log(640320**3+744)/sqrt(163)"
approx = eval(formule)
print(f"{approx=}\n{    pi=}")
```

```
approx=3.141592653589793
    pi=3.141592653589793
```

Méthode 5 : Formule de Basel

```
u = lambda n : 1/n**2

from math import pi
n, somme = 0, 0
while abs(somme-pi**2/6)>1e-5:
    n += 1
    somme += u(n)
print(f"N={n}, somme={somme:0.6f}, pi^2/6={pi**2/6:0.6f}")
```

```
N=100000, somme=1.644924, pi^2/6=1.644934
```

Méthode 6 : Formule de Madhava–Leibniz

```
u = lambda n : (-1)**n/(2*n+1)

from math import pi
n, somme = 0, 4
while abs(somme-pi)>1e-5:
    n += 1
    somme += 4*u(n)
print(f"N={n}, somme={somme:0.6f}, pi/4={pi:0.6f}")
```

```
N=100000, somme=3.141603, pi/4=3.141593
```

Méthode 7 : Formule de Madhava

```
u = lambda n : (-3)**(-n)/(2*n+1)

from math import pi
n, somme = 0, (12)**(0.5)
while abs(somme-pi)>1e-5:
```

```
    n += 1
    somme += (12)**(0.5)*u(n)
print(f"N={n}, somme={somme:0.6f}, pi/4={pi:0.6f}")
```

```
N=8, somme=3.141600, pi/4=3.141593
```

Méthode 8 : Formule de Bailey-Borwein-Plouffe
On note

$$u(n) = 16^{-n}\left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\right) \qquad \pi_{\text{approché}}(\mathrm{N}) = \sum_{n=0}^{\mathrm{N}} u(n)$$

```
from math import pi

u = lambda n :  ( 4/(8*n+1) - 2/(8*n+4) - 1/(8*n+5) - 1/(8*n+6) )*(1/16)**n
pi_approche = lambda N :  sum( [ u(n) for n in range(N+1)] )

N = 0
while abs(pi-pi_approche(N))>0:
    N += 1

print(f"N = {N}")
print("pi approx =",pi_approche(N))
print(" math.pi  =",pi)
```

```
N = 15
pi approx = 3.141592653589793
 math.pi  = 3.141592653589793
```

Pour $\mathrm{N} = 10$ on obtient une approximation de π qui coïncide (à la précision Python) avec la variable interne `math.pi`. Cet algorithme est en effet extrêmement efficace et permet le calcul rapide de centaines de chiffres significatifs de π.
Remarque : on peut améliorer cette implémentation en calculant, à chaque itération, juste $u(\mathrm{N}+1)$ et en posant $\pi_{\text{approché}}(\mathrm{N}+1) = \pi_{\text{approché}}(\mathrm{N}) + u(\mathrm{N}+1)$ (dans le script précédent on recalcule tous les termes de la suite $\{u(n)\}_n$ à chaque appelle à la fonction `pi_approx`).
Pour bien visualiser la convergence on affichera, pour chaque N, la valeur approchée et l'erreur. La mise en forme sera effectuée grâce à la fonction `tabulate` du module `tabulate` :

```
from tabulate import tabulate
table = []
table.append(["N", "Approximation", "Erreur"])

u = lambda n :  ( 4/(8*n+1) - 2/(8*n+4) - 1/(8*n+5) - 1/(8*n+6) )*(1/16)**n
N = 0
pi_approx = u(N)
from math import pi
table.append( [N, pi_approx, pi-pi_approx] )
while abs(pi-pi_approx)>0:
    N += 1
    pi_approx += u(N)
    table.append( [N, pi_approx, pi-pi_approx] )

print(tabulate(table,headers="firstrow",floatfmt=".15f"))
```

```
  N       Approximation               Erreur
---   -----------------   -----------------
  0   3.133333333333333   0.008259320256460
  1   3.141422466422466   0.000170187167327
  2   3.141587390346582   0.000005263243211
  3   3.141592457567436   0.000000196022357
  4   3.141592645460336   0.000000008129457
```

```
 5  3.141592653228088  0.000000000361705
 6  3.141592653572881  0.000000000016912
 7  3.141592653588973  0.000000000000820
 8  3.141592653589752  0.000000000000041
 9  3.141592653589791  0.000000000000002
10  3.141592653589793  0.000000000000000
```

Méthode 9 : Aire d'un cercle (Monte-Carlo)

```python
from math import pi
import random

N = 1000 # On tirera N points
m = 0    # Compteur pour les points qui tombent dans le disque
for k in range(N):
    # On tire au hasard un point [x,y] dans [0,1[ x [0,1[
    P = [ random.uniform(0,1), random.uniform(0,1) ]
    m += (P[0]**2+P[1]**2<=1) # Si ==True alors le point est dans le disque

myPi = 4*m/N # notre approximation de pi
err = abs(pi-myPi) # on compare à la valur de pi donnée par le module math
print(f"pi={pi:g}, myPi={myPi:g}, error={err:g}")
```

Pour N = 1000 on obtient

```
pi=3.14159, myPi=3.16, error=0.0184073
```

Naturellement, comme les nombres sont générés aléatoirement, les résultats obtenus pour une même valeur de N peuvent changer à chaque exécution.

Plus N est grand, meilleure est l'approximation de π. Cependant, cette méthode n'est pas très efficace, il faut beaucoup de tirs pour obtenir le deux premières décimales de π (pour obtenir chaque nouvelle décimale, nous devons augmenter le nombre de points aléatoires d'un facteur 10).

Méthode 10 : Intégrale double

```python
# VALEURE EXACTE  DE L'INTEGRALE
import sympy as sp
x = sp.Symbol('x')
y = sp.Symbol('y')
exacte = sp.integrate(  sp.integrate(4, (y,0,sp.sqrt(1-x**2))) , (x,0,19) )
print(f"L'intégrale vaut exactement {exacte }")


# VALEURE APPROCHÉE   DE L'INTEGRALE
from scipy.integrate import quad
from numpy import sqrt
approx = quad(lambda x: quad(lambda y: 4, 0, sqrt(1-x**2))[0], 0, 1)[0]
print(f"L'integrale vaut approximativement {approx = }")

from numpy import sqrt
from scipy.integrate import dblquad
# NB l'ordre des variables dans la fonction lambda: d'abord l'intégrale interne
# ∫_a^b (∫_c^{b(x)} f(x,y) dy) dx dblquad(lambda y, x: f(x,y), a, b, lambda x: c, lambda x:
↪  b(x))
approx = dblquad(lambda y, x: 4, 0, 1, lambda x: 0, lambda x: sqrt(1-x**2))[0]
print(f"L'integrale vaut approximativement {approx = }")
```

```
L'intégrale vaut exactement 2*asin(19) + 228*sqrt(10)*I
L'integrale vaut approximativement approx = 3.1415926535897922
L'integrale vaut approximativement approx = 3.1415926535897922
```

Méthode 11 : Suite de Borwein

```python
from math import pi,sqrt
s,a = (sqrt(3)-1)/2 , 1/3
for n in range(4):
    r = 3/(1+2*(1-s**3)**(1/3))
    s = (r-1)/2
    a = r**2*a-3**n*(r**2-1)
    pi_approx = 1/a
    print(f"{n=}, {pi_approx=:0.15f}, erreur={pi_approx-pi:0.15f}")
```

```
n=0, pi_approx=3.141590585205897, erreur=-0.000002068383896
n=1, pi_approx=3.141592653589804, erreur=0.000000000000011
n=2, pi_approx=3.141592653589804, erreur=0.000000000000011
n=3, pi_approx=3.141592653589804, erreur=0.000000000000011
```

Méthode 12 : Suite de Jonathan-Borwein

```python
from math import pi,sqrt
f = lambda x : (1-x**4)**(1/4)
y,a = sqrt(2)-1, 6-4*sqrt(2)
for n in range(4):
    y = (1-f(y))/(1+f(y))
    a = a*(1+y)**4-2**(2*n+3)*y*(1+y+y**2)
    pi_approx = 1/a
    print(f"{n=}, {pi_approx=:0.15f}, erreur={pi_approx-pi:0.15f}")
```

```
n=0, pi_approx=3.141592646213547, erreur=-0.000000007376246
n=1, pi_approx=3.141592653589793, erreur=-0.000000000000000
n=2, pi_approx=3.141592653589793, erreur=-0.000000000000000
n=3, pi_approx=3.141592653589793, erreur=-0.000000000000000
```

Méthode 13 : Suite de Gauss-Legendre

```python
from math import pi, sqrt
a,b,t,p = 1, 1/sqrt(2), 1/4, 1
for n in range(4):
    a_new = (a+b)/2
    a,b,t,p = a_new, sqrt(a*b), t-p*(a-a_new)**2, 2*p
    pi_approx = (a+b)**2/(4*t)
    print(f"{n=}, {pi_approx=:0.15f}, erreur={pi_approx-pi:0.15f}")
```

```
n=0, pi_approx=3.140579250522169, erreur=-0.001013403067625
n=1, pi_approx=3.141592646213543, erreur=-0.000000007376250
n=2, pi_approx=3.141592653589794, erreur=0.000000000000001
n=3, pi_approx=3.141592653589794, erreur=0.000000000000001
```

Méthode 14 : Le module decimal

```python
from decimal import *
mypi = Decimal (22) / Decimal (7)
print(mypi)
```

```
3.142857142857142857142857143
```

https://docs.python.org/fr/3/library/decimal.html

```python
from decimal import *
getcontext().prec = 50
mypi = Decimal (22) / Decimal (7)
print(mypi)
```

```
3.1428571428571428571428571428571428571428571428571
```

Exercice 7.35 (Module `scipy` - Calcul approché d'une intégrale (méthode Monte-Carlo))
La méthode de Monte-Carlo (du nom des casinos, pas d'une personne) est une approche probabiliste permettant d'**approcher la valeur de l'intégrale**

$$J = \int_a^b f(x)\,dx.$$

L'idée de base est que l'intégrale J peut être vue comme l'espérance d'une variable aléatoire uniforme X sur l'intervalle $[a,b]$. Par la loi des grands nombres cette espérance peut être approchée par la moyenne empirique

$$J \approx J(N) = \frac{b-a}{N}\sum_{i=0}^{N-1} f(x_i),$$

où les x_i sont tirés aléatoirement dans l'intervalle $[a,b]$ avec une loi de probabilité uniforme.

- Écrire une fonction `montecarlo(f,a,b,N)` qui calcule J_N.
- Valider la fonction (*i.e.* on considère des cas dont on connaît la solution exacte et on écrit un test unitaire). Quelle valeur obtient-on avec le module `scipy.integrate`?

Correction

```
import random
montecarlo = lambda f,a,b,N : (b-a)*sum([f(random.uniform(a,b)) for i in range(N)])/N

# TESTS
from scipy import integrate

g = lambda x: 1
print("Calcul approché par Montecarlo =",montecarlo(g,0,1,100))
print("Calcul approché par ScyPy =", integrate.quad(g,0,1)[0])

g = lambda x: x**3
print("Calcul approché par Montecarlo =",montecarlo(g,0,1,100))
print("Calcul approché par ScyPy =", integrate.quad(g,0,1)[0])
```

```
Calcul approché par Montecarlo = 1.0
Calcul approché par ScyPy = 1.0
Calcul approché par Montecarlo = 0.24246108741142147
Calcul approché par ScyPy = 0.25
```

Exercice 7.36 (Module `numpy` - Statistique)
Soit L une liste de n valeurs. On rappelle les définitions suivantes :

Moyenne arithmétique $\overline{m} = \frac{\sum L_i}{n}$

Variance $V = \frac{\sum (L_i - \overline{m})^2}{n-1}$

Écart-type $\sigma = \sqrt{V}$

Médiane C'est la valeur qui divise l'échantillon en deux parties d'effectifs égaux. Soit C la liste qui contient les composantes de L ordonnées, alors

$$\text{médiane} = \begin{cases} \dfrac{C_{\frac{n}{2}} + C_{\frac{n}{2}+1}}{2} & \text{si } n \text{ est pair,} \\ C_{\frac{n}{2}+1} & \text{si } n \text{ est impair.} \end{cases}$$

Attention, dans cette définition les indices commencent à 1.

Écrire trois fonctions qui renvoient respectivement la moyenne, l'écart type et la médiane d'une liste donnée en entré. Vérifier l'implémentation sur deux listes bien choisies et comparer avec les calculs effectués par les fonctions `mean`, `std` et `median` prédéfinies dans le module `numpy` (à importer).

Correction
Pour vérifier l'implémentation, il faut pouvoir tester les deux cas possibles pour le calcul de la médiane, à savoir une liste qui a un nombre pair d'éléments puis une liste qui en a un nombre impair.

```
from numpy import *

moyenne = lambda L : sum(L)/len(L)

def variance(L):
    m = moyenne(L)
    return sum((ell-m)**2 for ell in L)/(len(L)-1) # cas d'un échantillon

ecart_type = lambda L : (variance(L))**0.5

def mediane(L):
    C = sorted(L)
    n = len(L)
    if n%2==0 :
        return (C[n//2-1]+C[n//2])/2
    else:
        return C[n//2]

# TESTS
for L in ( [0, 0, 8, 1, 1, 2, 2] , [2, 3, 3, 2] ):
    print(f"{L = }")
    print(f"{moyenne(L) = }, avec numpy on trouve {mean(L) = }")
    print(f"{ecart_type(L) = }, avec numpy on trouve {std(L,ddof=1) = }")
    print(f"{mediane(L) = }, avec numpy on trouve {median(L) = }\n")
```

```
L = [0, 0, 8, 1, 1, 2, 2]
moyenne(L) = 2.0, avec numpy on trouve mean(L) = 2.0
ecart_type(L) = 2.7688746209726918, avec numpy on trouve std(L,ddof=1) = 2.7688746209726918
mediane(L) = 1, avec numpy on trouve median(L) = 1.0

L = [2, 3, 3, 2]
moyenne(L) = 2.5, avec numpy on trouve mean(L) = 2.5
ecart_type(L) = 0.5773502691896257, avec numpy on trouve std(L,ddof=1) = 0.5773502691896257
mediane(L) = 2.5, avec numpy on trouve median(L) = 2.5
```

Exercice 7.37 (Module numpy - Statistique et suites)

Soit $\mathbf{x}$ une liste de $n \in \mathbb{N}^*$ valeurs et définissons la fonction $\mathrm{F}(\mathbf{x}) = (\mathrm{mean}(\mathbf{x}), \mathrm{geometric_mean}(\mathbf{x}), \mathrm{median}(\mathbf{x}))$.

On veut vérifier que, quel que soit la liste $\mathbf{x}$, il existe une valeur ℓ telle que $\mathrm{F}(\mathrm{F}(\mathrm{F}(\dots(\mathrm{F}(\mathbf{x}))))) = (\ell, \ell, \ell)$, c'est-à-dire que la suite $\{\mathbf{y}_n\}_n$ définie par récurrence

$$\begin{cases} \mathbf{y}_0 = \mathrm{F}(\mathbf{x}) \\ \mathbf{y}_{n+1} = \mathrm{F}(\mathbf{y}_n) \end{cases}$$

converge vers un triplet (ℓ, ℓ, ℓ).

On pourra utiliser les fonctions `mean`, `geometric_mean`, `median` du module `statistics`.

Correction
Pour une meilleur affichage, on utilise le module `tabulate`.

```
from tabulate import tabulate
T = []
T.append(["i","yy[0]","yy[1]","yy[2]"])
```

```
from statistics import mean, geometric_mean, median
f = lambda xx : (mean(xx),geometric_mean(xx),median(xx))

from random import *
n = randint(20,50)
xx = [randint(1,100) for _ in range(n)]

i = 0
yy = f(xx)
T.append([f"{i}",f"{yy[0]}",f"{yy[1]}",f"{yy[2]}"])

while abs(yy[1]-yy[0])>1.e-12 or abs(yy[1]-yy[2])>1.e-12:
    yy = f(yy)
    i += 1
    T.append([f"{i}",f"{yy[0]}",f"{yy[1]}",f"{yy[2]}"])
print(tabulate(T,headers="firstrow",floatfmt=".13f"))
```

```
  i             yy[0]             yy[1]             yy[2]
---  ----------------  ----------------  ----------------
  0  43.5000000000000  34.6371094074723  42.0000000000000
  1  40.0457031358241  39.8498535410944  42.0000000000000
  2  40.6318522256395  40.6203799350556  40.0457031358241
  3  40.4326450988397  40.4317160932825  40.6203799350556
  4  40.4949137090593  40.4948166237650  40.4326450988397
  5  40.4741251438880  40.4741145124938  40.4948166237650
  6  40.4810187600490  40.4810175844396  40.4741251438880
  7  40.4787204961255  40.4787203656986  40.4810175844396
  8  40.4794861487546  40.4794861342703  40.4787204961255
  9  40.4792309263835  40.4792309247744  40.4794861342703
 10  40.4793159951427  40.4793159949639  40.4792309263835
 11  40.4792876388300  40.4792876388102  40.4793159949639
 12  40.4792970908680  40.4792970908658  40.4792876388300
 13  40.4792939401880  40.4792939401877  40.4792970908658
 14  40.4792949904138  40.4792949904138  40.4792939401880
 15  40.4792946403385  40.4792946403385  40.4792949904138
 16  40.4792947570303  40.4792947570303  40.4792946403385
 17  40.4792947181330  40.4792947181330  40.4792947570303
 18  40.4792947310988  40.4792947310988  40.4792947181330
 19  40.4792947267769  40.4792947267769  40.4792947310988
 20  40.4792947282175  40.4792947282175  40.4792947267769
 21  40.4792947277373  40.4792947277373  40.4792947282175
 22  40.4792947278974  40.4792947278974  40.4792947277373
 23  40.4792947278440  40.4792947278440  40.4792947278974
 24  40.4792947278618  40.4792947278618  40.4792947278440
 25  40.4792947278559  40.4792947278559  40.4792947278618
 26  40.4792947278578  40.4792947278578  40.4792947278559
 27  40.4792947278572  40.4792947278572  40.4792947278578
```

☆ Exercice Bonus 7.38 (Module numpy - Représentation et manipulation de polynômes)
Résoudre l'exercice 6.38 en utilisant le sous-module polynomial du module numpy.

Correction
Première méthode :

```
import numpy as np
from numpy.polynomial import polynomial

p = np.array([1, 2, 3])
```

```
x = np.array([-1,0,1,2])
y = polynomial.polyval(x,p)
print( f"p(x)={p} => p({x})={y}" )

p = np.array([1, 2, 3])
q = np.array([1, -2])
somme = polynomial.polyadd(p,q)
print(f"p(x)={p}, q(x)={q} => (p+q)(x)={somme}")

p = np.array([1, 0, 3])
q = np.array([1, -2])
mul = polynomial.polymul(p,q)
print(f"p(x)={p}, q(x)={q} => (pq)(x)={mul}")

p = np.array([1, 2, 6])
d = polynomial.polyder(p)
print(f"p(x)={p} => p'(x)={d}")

p = np.array([1, 2, 6])
q = polynomial.polyint(p)
print(f"q'(x)={p} => q(x)={q}")
```

```
p(x)=[1 2 3] => p([-1  0  1  2])=[ 2.  1.  6. 17.]
p(x)=[1 2 3], q(x)=[ 1 -2] => (p+q)(x)=[2. 0. 3.]
p(x)=[1 0 3], q(x)=[ 1 -2] => (pq)(x)=[ 1. -2.  3. -6.]
p(x)=[1 2 6] => p'(x)=[ 2. 12.]
q'(x)=[1 2 6] => q(x)=[0. 1. 1. 2.]
```

Deuxième méthode

```
from numpy.polynomial import Polynomial

p = Polynomial([1, 2, 3])
x = np.array([-1,0,1,2])
y = p(x)
print( f"p(x)={p} => p({x})={y}" )

p = Polynomial([1, 0, 3])
q = Polynomial([1, -2])
s = p+q
print(f"p(x)={p}, q(x)={q} => (p+q)(x)={s}")

p = Polynomial([1, 0, 3])
q = Polynomial([1, -2])
m = p*q
print(f"p(x)={p}, q(x)={q} => (pq)(x)={m}")

p = Polynomial([1, 2, 6])
d = p.deriv()
print(f"p(x)={p} => p'(x)={d}")

p = Polynomial([1, 2, 6])
q = p.integ()
print(f"q'(x)={p} => q(x)={q}")
```

```
p(x)=1.0 + 2.0·x¹ + 3.0·x² => p([-1  0  1  2])=[ 2.  1.  6. 17.]
p(x)=1.0 + 0.0·x¹ + 3.0·x², q(x)=1.0 - 2.0·x¹ => (p+q)(x)=2.0 - 2.0·x¹ + 3.0·x²
p(x)=1.0 + 0.0·x¹ + 3.0·x², q(x)=1.0 - 2.0·x¹ => (pq)(x)=1.0 - 2.0·x¹ + 3.0·x² - 6.0·x³
p(x)=1.0 + 2.0·x¹ + 6.0·x² => p'(x)=2.0 + 12.0·x¹
q'(x)=1.0 + 2.0·x¹ + 6.0·x² => q(x)=0.0 + 1.0·x¹ + 1.0·x² + 2.0·x³
```

☆ Exercice Bonus 7.39 (Module sympy - Représentation et manipulation de polynômes)

Résoudre l'exercice 6.38 en utilisant le module sympy.

Correction

```
import sympy
x = sympy.symbols('x')

p = 1+2*x+3*x**2
xx = [-1,0,1,2]
yy = [p.subs(x,xi) for xi in xx]
print(f"p(x)={p} => p({xx})={yy}")

p = 1+2*x+3*x**2
q = 1-2*x
somme = p+q
print(f"p(x)={p}, q(x)={q} => (p+q)(x)={somme}")

p = 1+2*x+3*x**2
q = 1-2*x
mul = p*q
print(f"p(x)={p}, q(x)={q} => (pq)(x)={mul}")

p = 1+2*x+6*x**2
q = p.diff()
print(f"q'(x)={p} => q(x)={q}")

p = 1+2*x+6*x**2
q = p.integrate()
print(f"q'(x)={p} => q(x)={q}")
```

```
p(x)=3*x**2 + 2*x + 1 => p([-1, 0, 1, 2])=[2, 1, 6, 17]
p(x)=3*x**2 + 2*x + 1, q(x)=1 - 2*x => (p+q)(x)=3*x**2 + 2
p(x)=3*x**2 + 2*x + 1, q(x)=1 - 2*x => (pq)(x)=(1 - 2*x)*(3*x**2 + 2*x + 1)
q'(x)=6*x**2 + 2*x + 1 => q(x)=12*x + 2
q'(x)=6*x**2 + 2*x + 1 => q(x)=2*x**3 + x**2 + x
```

☆ Exercice Bonus 7.40 (Permutations avec itertools)

Soit un nombre à 10 chiffres tous différents. On pourra écrire ce nombre sous la forme $\overline{abcdefghij}$ ayant noté avec une barre la suite des chiffres qui le composent dans l'écriture décimale. Trouver le seul nombre tel que $\overline{a}$ est divisible par 1, $\overline{ab}$ est divisible par 2, $\overline{abc}$ est divisible par 3, etc. $\overline{abcdefghij}$ est divisible par 10.

Correction

On considère les permutations de la liste de chaînes de caractères `C = ["0","1","2","3","4","5","6","7","8","9"]`. La concatenation des éléments d'une permutation de `C` donne un nombre à 10 chiffres tous différents. Pour chaque liste `P` qui est une permutation de la liste `C`, on vérifie toutes les conditions. Dès qu'elles sont toutes satisfaites, on sort de la fonction.

```
import itertools
def f():
	C = [str(i) for i in range(10)]
	for P in list(itertools.permutations(C)):
		cond = [ int(''.join(P[:j]))%j==0   for j in range(1,11) ]
		if all(cond):
			return int(''.join(P))

print(f())
```

```
3816547290
```

Une autre stratégie consiste à tester tous les cas possibles en aidant la construction de ce nombre :

- $j = 0$ car $\underline{abcdefghij}$ est divisible par 10,
- $e = 5$ car $\underline{abcde}$ est divisible par 5 et 0 a été déjà utilisé,
- $b, d, f, h \in 2, 4, 6, 8$ car $\underline{ab}$, $\underline{abcd}$, $\underline{abcdef}$ et $\underline{abcdefgh}$ sont tous divisibles par 2,
- $a, c, g, i \in \{1, 3, 7, 9\}$ car les autres chiffres ont été utilisés,

Notons que le code devient presque illisible.

```python
def f():
    j = 0
    e = 5
    for a in [1,3,7,10]:
        for b in range(2,9,2):
            for c in [1,3,7,9]:
                abc = int(str(a)+str(b)+str(c))
                if abc%3==0:
                    for d in range(2,9,2):
                        abcd = int(str(abc)+str(d))
                        if d!=b and abcd%4==0 :
                            for f in range(2,9,2):
                                abcdef = int(str(abcd)+str(5)+str(f))
                                if f!=b and f !=d and abcdef%6==0:
                                    for g in [1,3,7,9]:
                                        abcdefg = int(str(abcdef)+str(g))
                                        if g!=c and abcdefg%7==0:
                                            for h in range(2,9,2):
                                                abcdefgh = int(str(abcdefg)+str(h))
                                                if h!=b and h!=d and h!=f and abcdefgh%8==0:
                                                    for i in [1,3,7,9]:
                                                        abcdefghi = int(str(abcdefgh)+str(i))
                                                        if i!=a and i!=c and i!=5 and i!=g
                                                        ↪   and abcdefghi%9==0:
                                                            return int(str(abcdefghi)+str(i))

print(f())
```

Exercice Bonus 7.41 (Permutations avec itertools)

Lors du premier tome de Harry Potter les trois héros doivent résoudre une énigme afin d'accéder à la salle où est cachée la pierre philosophale. Ce problème, dont l'auteur serait le professeur Rogue, consiste à trouver deux potions parmi les sept qui se trouvent devant eux : celles permettent d'avancer et de reculer. Ils sont aidés de quelques indices :

- Il y a trois fioles de poison, deux fioles de vin d'ortie, une fiole permettant d'avancer et une fiole permettant de reculer.
- Immédiatement à gauche de chacune des deux fioles de vin se trouve une fiole de poison.
- La première et la dernière fiole ont des contenus différents ; ni l'une ni l'autre n'est la fiole qui permet d'avancer.
- Les contenus des fioles en position 1 et 5 sont identiques et ne contient pas du poison.

Trouver le contenu des 7 fioles. Pour générer toutes les permutations possibles, on pourra utiliser le module `itertools`. Voici un exemple d'utilisation

```python
import itertools
for L in list(itertools.permutations(["A","B","C"])):
    print(L)
```

```
('A', 'B', 'C')
('A', 'C', 'B')
('B', 'A', 'C')
```

```
('B', 'C', 'A')
('C', 'A', 'B')
('C', 'B', 'A')
```

Source : http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx/specials/hermionne.html

Correction

On considère les permutations de la liste F=["p","p","p","v","v","a","r"] où "p" indique une fiole qui contient du poison, "v" du vin d'ortie, "a" si elle permet d'avancer et "r" de reculer. Pour chaque liste L qui est une permutation de la liste F, on vérifie toutes les conditions. Dès qu'elles sont toutes satisfaites, on sort de la fonction (sans continuer dans la boucle car on cherche un cas qui marche, pas tous les cas).

```python
import itertools
def f():
	F = ["p","p","p","v","v","a","r"]
	for L in list(itertools.permutations(F)):
		if L[0]!=L[6]  and L[6]!="a" and L[0]!="a" and L[5]!="p" and L[2]!="p" and
		↪ L[1]==L[5]:
			idx = [ i for i in range(1,7) if L[i]=="v"]
			if sum([L[i]=="v" and L[i-1]=="p" for i in idx])==2:
				return L

print(f())
```

```
('p', 'v', 'a', 'p', 'p', 'v', 'r')
```

☆ Exercice Bonus 7.42 (Module sympy – devine le résultat)

Tester les codes suivants :

1. ```python
   from sympy import *
   var('x')
   expr=(x**3+x**2-x-1)/(x**2+2*x+1)
   expr2=simplify(expr)
   print(expr,"=",expr2)
   ```
2. ```python
   from sympy import *
   var('x')
   expr=x**4/2+5*x**3/12-x**2/3
   expr2=factor(expr)
   print(expr,"=",expr2)
   ```
3. ```python
   from sympy import *
   var('x')
   expr=x**4-1
   sol=solve(expr)
   print(sol)
   ```
4. ```python
   from sympy import *
   var('x,a,b,c')
   expr = a*x**2 + b*x + c
   sol=solve(expr, x)
   print(sol)
   ```
5. ```python
   from sympy import *
   var('x,y')
   print(limit(cos(x),x,0))
   print(limit(x,x,oo))
   print(limit(1/x,x,oo))
   print(limit(x**x,x,0))
   print(diff(x*y**2,y))
   print(integrate(sin(x),x))
   ```

Correction

1. $\dfrac{x^3+x^2-x-1}{x^2+2x+1} = x-1$
2. $\dfrac{x^4}{2}+\dfrac{5x^3}{12}-\dfrac{x^2}{3} = \dfrac{x^2(2x-1)(3x+4)}{12}$
3. $[-1,\ 1,\ -i,\ i]$
4. $\left[\dfrac{-b+\sqrt{-4ac+b^2}}{2a},\ -\dfrac{b+\sqrt{-4ac+b^2}}{2a}\right]$
5. $1, \infty, 0, 1, 2xy, -\cos(x)$
   ```

Exercice Bonus 7.43 (Module sympy – calcul formel d'une dérivée)
Calculer $\partial_x f(x,y)$ pour
$$f(x,y)=\frac{ay-1}{(bx-1)^c}.$$

Correction

```python
from sympy import *
var('x,y,a,b,c')
f=(a*y-1)/(b*x-1)**c
g=diff(f,x,1)
print(g)
```

$$-\frac{bc\left(ay-1\right)(bx-1)^{-c}}{bx-1}$$

Exercice 7.44 (Approximation valeur ponctuelle dérivées)
Écrire une fonction `derivatives` qui approche la valeur des dérivées première et seconde d'une fonction f en un point x par les formules

$$f'(x)\simeq\frac{f(x+h)-f(x-h)}{2h},\qquad f''(x)\simeq\frac{f(x+h)-2f(x)+f(x-h)}{h^2}.$$

Comparer la valeur exacte avec la valeur approchée pour la fonction $x\mapsto\cos(x)$ en $x=\frac{\pi}{2}$ (à vous de choisir une valeur de h satisfaisante).

Correction
Si $f(x)=\cos(x)$ alors $f'(x)=-\sin(x)$ et $f''(x)=-\cos(x)$ donc $f'\left(\frac{\pi}{2}\right)=1$ et $f''\left(\frac{\pi}{2}\right)=0$.

```python
import math

def derivatives(f,x,h):
	df = (f(x+h)-f(x-h))/(2*h)
	ddf = (f(x+h)-2*f(x)+f(x-h))/h**2
	return df,ddf

df, ddf = derivatives(f=math.cos, x=math.pi/2, h=1.0e-5)
print('First derivative =', df)
print('Second derivative =', ddf)
```

```
First derivative = -0.9999999999898845
Second derivative = 1.6940658945086004e-11
```

Exercice Bonus 7.45 (Module sympy – composition de fonctions)
Calculer $f(m,f(n,m))$ si
$$f(a,b)=a^2-b.$$

Correction

```python
import sympy as sym
sym.var('a,b,n,m')
f = lambda a,b :a**2-b
print( f(m,f(n,m)) )
```

$$m^2+m-n^2$$

Exercice Bonus 7.46 (Module sympy – calcul des paramètres)
La courbe d'équation $y=ax^2+bx+c$ passe par le point $(1;P_1)$ et la droite tangente à son graphe au point $(2;P_2)$ a pente égale à 1. Calculer a,b,c.

Correction

```
import sympy as sym
sym.var('x,a,b,c,P_1,P_2')
f = a*x**2+b*x+c
eq1 = sym.Eq(f.subs(x,1),P_1)
eq2 = sym.Eq(f.subs(x,2),P_2)
fp = sym.diff(f,x)
eq3 = sym.Eq(fp.subs(x,2),1)
sol = sym.solve([eq1,eq2,eq3],[a,b,c])
print( sol )
```

$$\{a: P_1 - P_2 + 1,\ b: -4P_1 + 4P_2 - 3,\ c: 4P_1 - 3P_2 + 2\}$$

☆ Exercice Bonus 7.47 (Module `sympy` – calcul des paramètres)

Soit $f\colon \mathbb{R} \to \mathbb{R}$ la fonction définie par $f(x) = \dfrac{ax+b}{x^2+1}$. Calculer a, b, c si f a un maximum local en $x_M = 1$ et $f(x_M) = c$.

Correction

```
import sympy as sym
sym.var('x,a,b,c')
f=(a*x+b)/(x**2+1)
eq1=f.subs(x,1)-c
fp=sym.diff(f,x)
eq3=fp.subs(x,1)
sol=sym.solve([eq1,eq3],[a,b])
print( sol )
```

$$\{a: 2c,\ b: 0\}$$

☆ Exercice Bonus 7.48 (Module `sympy` – calcul de paramètres)

Soit $f\colon \mathbb{R} \to \mathbb{R}$ la fonction définie par $f(x) = x^4 - ax^3 + bx^2 + cx + d$. Si $b = -3a^2$, sur quel intervalle cette fonction est concave ?

Correction

```
import sympy as sym
sym.var('x,a,b,c,d')
b=-3*a**2
f=x**4-a*x**3+b*x**2+c*x+d
fp=sym.diff(f,x)
fs=sym.diff(fp,x)
sol=sym.solve(fs,x)
print( sol )
```

$$\left[-\frac{a}{2},\ a\right]$$

CHAPITRE 8

Tracé de courbes

8.1. Importation des modules `matplotlib` et `numpy`

Le tracé de courbes scientifiques peut se faire à l'aide du package `matplotlib` et du module `numpy` (dont on a parlé à la page 220). On peut importer les deux modules de deux façons.

- La "bonne" façon consiste en l'utilisation des instructions

  ```
  import matplotlib.pyplot as plt
  import numpy as np
  ```

  Avec ce type de import il faudra faire précéder toutes les instruction `matplotlib` par `plt` et celles `numpy` par `np`, comme dans l'exemple qui suit

  ```
  import matplotlib.pyplot as plt
  import numpy as np
  xx = np.linspace(-np.pi,np.pi,101)
  plt.plot(xx,np.cos(xx))
  plt.show()
  ```

On peut importer les deux modules ensemble en important le sous-module `matplotlib.pylab` qui associe les fonctions de `pyplot` (pour les tracés) avec les fonctionnalités du module `numpy` (pour la gestion des tableaux et les fonctions mathématiques vectorisées).

- On peut combiner l'importation des deux modules avec

  ```
  import matplotlib.pylab as pyl
  ```

  qui importe aussi le module `numpy` avec le même alias.
  Avec ce type de import il faudra faire précéder toutes les instruction (de `matplotlib` ou de `numpy`) par `pyl`, comme dans l'exemple qui suit

  ```
  import matplotlib.pylab as pyl
  xx = pyl.linspace(-pyl.pi,pyl.pi,101)
  pyl.plot(xx,pyl.cos(xx))
  pyl.show()
  ```

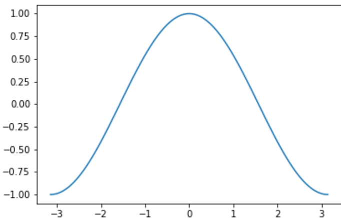
- La méthode du "paresseux" consiste en l'utilisation de l'instruction

  ```
  from matplotlib.pylab import *
  ```

  qui importe aussi les deux modules sans alias. On obtient un environnement très proche de celui de MATLAB/Octave, très répandu en calcul scientifique.

  ```
  from matplotlib.pylab import *
  xx = linspace(-pi,pi,101)
  plot(xx,cos(xx))
  show()
  ```

Dans tous les cas on obtiendra l'image suivante :



Deux remarques :

- quelle qu'elle soit la méthode choisie, il est inutile d'importer le module `math` car `numpy` redéfinit les même fonctions ;

- les fonctions de `numpy` sont vectorisées, *i.e.* si on les applique à une liste **x** (en réalité un `array numpy`), elles génèrent une liste **y** qui contient les évaluations en chaque point de **x**. Dans notre exemple l'écriture `np.cos(xx)` équivaut à `[np.cos(x) for x in xx]` car la fonction cosinus utilisée est celle définie par le module `numpy`.

8.2. Tracé d'une courbe sur un repère

Pour tracer le graphe d'une fonction $f: [a, b] \to \mathbb{R}$, il faut tout d'abord générer une liste de points x_i où évaluer la fonction f, puis la liste des valeurs $f(x_i)$ et enfin, avec la fonction `plot`, Python reliera entre eux les points $(x_i, f(x_i))$ par des segments. Plus les points sont nombreux, plus le graphe est proche du graphe de la fonction f.

Pour générer les points x_i on peut utiliser l'une des deux instructions suivantes fournies par le module `numpy` :

- soit l'instruction `np.linspace(a,b,n+1)` qui construit la liste de $n+1$ éléments

$$[a, a+h, a+2h, \dots, b = a+nh] \quad \text{avec } h = \frac{b-a}{n}$$

- soit l'instruction `np.arange(a,b,h)` qui construit la liste de $n = \mathrm{E}\left(\frac{b-a}{h}\right) + 1$ éléments

$$[a, a+h, a+2h, \dots, a+nh]$$

  Dans ce cas, attention au dernier terme : avec des `float` les erreurs d'arrondis pourraient faire en sort que b ne soit pas pris en compte.

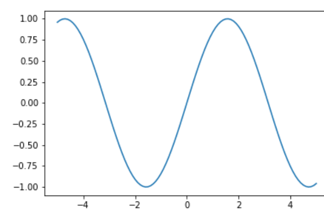
Canevas

Voici un canevas pour afficher le graphe de la fonction $f: [a; b] \to \mathbb{R}$ définie par $y = f(x)$ avec $n+1$ points :

```
import matplotlib.pyplot as plt
import numpy as np

f = lambda x : # à compléter
a,b,n = # à compléter

xx = np.linspace(a,b,n+1) # n+1 points, n sous-intervalles de largeur (b-a)/n
yy = [f(x) for x in xx] # si f est une fonction numpy, on écrira yy = np.f(xx)
plt.plot(xx,yy)
plt.show()
```

Un exemple avec une sinusoïde :

```
import matplotlib.pyplot as plt
import numpy as np
xx = np.linspace(-5,5,101) # x = [-5,-4.9,-4.8,...,5]
# xx = np.arange(-5,5,0.1) # idem
yy = np.sin(xx)  # fonction vectorisee
plt.plot(xx,yy)
plt.show()
```



On obtient une courbe sur laquelle on peut zoomer, modifier les marges et sauvegarder dans différents formats.

On peut spécifier la couleur et le type de trait, changer les étiquettes des axes, donner un titre, ajouter une grille, une légende etc.

 © G. Faccanoni

```
import matplotlib.pyplot as plt
import numpy as np
xx = np.linspace(-5,5,101)
yy = np.sin(xx)
plt.plot(xx,yy,color='r',ls=":",lw=2,label=r'$y=\sin(t)$')
plt.legend()
plt.grid()
plt.xlabel('$t$')
plt.ylabel('$y$')
plt.title('Mon premier tracé')
plt.show()
```

8.3. Plusieurs courbes sur le même repère et options

On peut tracer plusieurs courbes dans le même repère.

Par exemple, dans la figure suivante, on a tracé la même fonction : la courbe bleu correspond à la grille la plus grossière, la courbe rouge correspond à la grille la plus fine :

```
import matplotlib.pyplot as plt
import numpy as np
x1 = np.linspace(-5,5,5)
y1 = np.sin(a)
plt.plot(x1,y1,'b-',label="5 points")
x2 = np.linspace(-5,5,101)
y2 = np.sin(b)
plt.plot(x2,y2,'r-.',label="101 points" )
plt.legend()
plt.show()
```

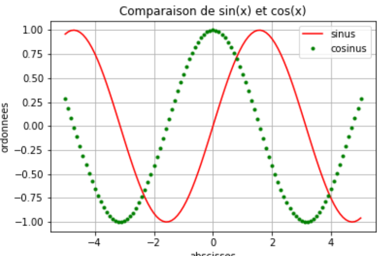
Pour **tracer plusieurs courbes sur le même repère**, on peut les mettre les unes à la suite des autres en spécifiant la couleur et le type de trait, changer les étiquettes des axes, donner un titre, ajouter une grille, une légende etc.

8.3.1. Personnalisation

Quasiment tous les aspects d'une figure peuvent être configurés par l'utilisateur soit pour y ajouter des données, soit pour améliorer l'aspect esthétique. Plutôt que de vous faire une liste des fonctions qui permettent de faire ces actions, voici des exemples.

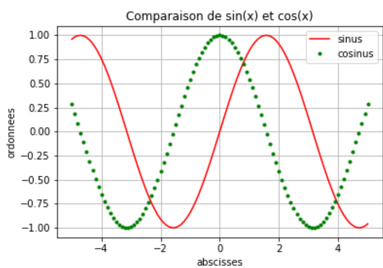
Par exemple, dans le code ci-dessous `"r-"` indique que la première courbe est à tracer en rouge (red) avec un trait continu, et `"g."` que la deuxième est à tracer en vert (green) avec des points.

```
import matplotlib.pyplot as plt
import numpy as np
x  = np.linspace(-5,5,101)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x,y1,"r-",x,y2,"g.")
plt.legend(['sinus','cosinus'])
plt.xlabel('abscisses')
plt.ylabel('ordonnees')
plt.title('Comparaison de np.sin(x) et np.cos(x)')
plt.grid(True)
plt.show()
```



soit encore

```
import matplotlib.pyplot as plt
import numpy as np
x  = np.linspace(-5,5,101)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x,y1,"r-",label=('sinus'))
plt.plot(x,y2,"g.",label=('cosinus'))
plt.legend()
plt.xlabel('abscisses')
plt.ylabel('ordonnees')
plt.title('Comparaison de np.sin(x) et np.cos(x)')
plt.grid(True)
plt.show()
```



On peut être plus explicite et au lieu de donner un string du type `'r--'` indiquer explicitement qu'il s'agit d'une couleur et d'un style de ligne.

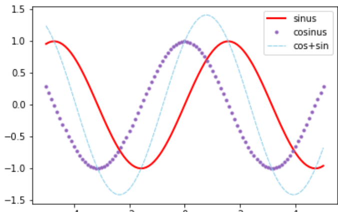
- Pour la couleur, on écrira `plt.plot(x,y,'r')` ou `plt.plot(x,y,color='red')` ou encore `plt.plot(x,y,c='red')`. Dans le tableau on a indiqué les couleurs de base. On peut cependant utiliser aussi une string html hex comme `plt.plot(x,y,color="#f44265")` ou un tuple RGB comme `plt.plot(x,y,color=(0.9569, 0.2588, 0.3891) )`
- Pour définir le type de ligne, on écrira `plt.plot(x,y,'--')` ou `plt.plot(x,y,linestyle='dashed')` ou encore `plt.plot(x,y,ls='dashed')`. Dans le tableau on a indiqué les 4 styles de base.
- Pour définir l'épaisseur de la ligne, on écrira par exemple `plt.plot(x,y,linewidth=2)` ou encore l'abréviation `plt.plot(x,y,lw=2)` (l'unité est le point).
- Pour définir un marker, on écrira `plt.plot(x,y,'o')` ou `plt.plot(x,y,marker='o')`. Dans le tableau on a indiqué les marker disponibles. Il est possible d'en fixer la taille par les commandes `plt.plot(x,y,markersize=5)` ou encore `plt.plot(x,y,ms=5)` (l'unité est le point).

Quelques options de `pyplot` :

	linestyle=		color=	marker=	
-	solid	r	red	.	points
- -	dashed	g	green	,	pixel
:	dotted	b	blue	o	filled circles
-.	dashdot	c	cyan	v	triangle down
(0,(5,1))	densely dashed	m	magenta	^	triangle up
		y	yellow	>	triangle right
		w	white	<	triangle left symbols
		k	black	*	star
				+	plus
				s	square
				p	pentagon
				x	x
				X	x filled
				d	thin diamond
				D	diamond

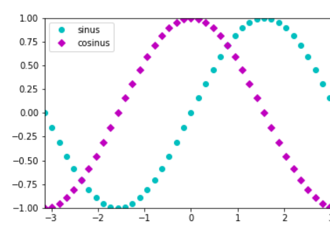
Voici un exemple d'utilisation :

```
import matplotlib.pyplot as plt
import numpy as np
x  = np.linspace(-5,5,101)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.sin(x)+np.cos(x)
plt.plot(x,y1,color='r'      ,ls='-'      ,linewidth=2            ,label='sinus')
plt.plot(x,y2,color='purple' ,ls=' '      ,lw=1       ,marker='.',label='cosinus')
plt.plot(x,y3,color='skyblue',ls=(0,(5,1)),lw=1                   ,label='cos+sin')
plt.legend()
plt.show()
```



On peut déplacer la légende en spécifiant l'une des valeurs suivantes : `best`, `upper right`, `upper left`, `lower right`, `lower left`, `center right`, `center left`, `lower center`, `upper center`, `center` :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-np.pi,np.pi,0.05*np.pi)
plt.plot(x,np.sin(x),'co', x,np.cos(x),'mD')
plt.legend(['sinus','cosinus'],loc='upper left')
# plt.axis([xmin, xmax, ymin, ymax])
plt.axis([-np.pi, pi, -1, 1]);
plt.show()
```



On peut personnaliser l'aspect des étiquettes des axes, par exemple

```
plt.xlabel("Time (s)", size = 16, family='serif', color='r', weight='normal', labelpad = 6)
```

- Pour `size`, il s'agit de la taille (en points).
- Pour `family`, on peut choisir parmi `'serif'`, `'sans-serif'`, `'cursive'`, `'fantasy'`, `'monospace'`
- Pour la couleur, on a les mêmes possibilités que pour les lignes.
- Pour `weight`, on peut choisir parmi `'light'`, `'normal'`, `'medium'`, `'semibold'`, `'bold'`, `'heavy'` et `'black'`.
- Le `labelpad` est la distance entre l'axe et l'étiquette.

On peut personnaliser le titre du repère, par exemple

```
plt.title("Title Example", loc='right', fontdict={'family': 'serif', 'color' : 'darkblue',
↪ 'weight': 'bold', 'size': 18})
```

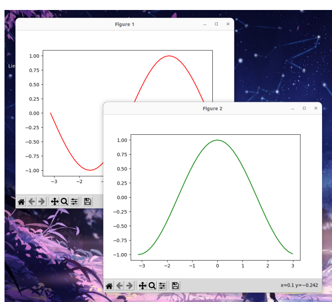
Voir aussi

- https://matplotlib.org/api/markers_api.html
- https://matplotlib.org/examples/lines_bars_and_markers/marker_reference.html
- https://matplotlib.org/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py
- https://matplotlib.org/examples/lines_bars_and_markers/linestyles.html
- https://www.delftstack.com/tutorial/matplotlib/

8.4. Plusieurs “fenêtres” graphiques

Avec `plt.figure()` on génère une nouvelle fenêtre contenant les graphes qui suivent :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-np.pi,np.pi,0.05*np.pi)
plt.figure(1)
plt.plot(x, np.sin(x), 'r')
plt.figure(2)
plt.plot(x, np.cos(x), 'g')
plt.show()
```



Pour modifier la taille de la figure on utilisera `figsize`, comme par exemple `plt.figure(1,figsize=(3,3))`.

8.5. Une grille de repères dans la même fenêtre

La fonction `fig,ax=plt.subplots(r,c)` subdivise la fenêtre (appelée `fig`) en une grille (appelée `ax`) de `r` lignes et `c` colonnes. Par exemple, si on écrit `plt.subplots(4,3)`, cela signifie que la fenêtre principale a été subdivisée en $4 \times 3 = 12$ cases.

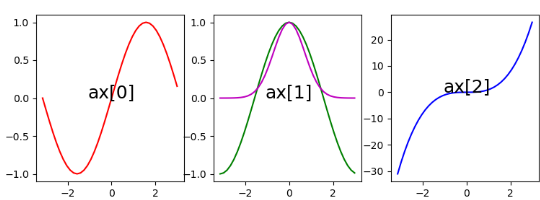
8.5.1. Approche I

Chaque case est numérotée avec un seul indice si la grille contient une seule ligne ($r = 1$) et avec deux indices si $r > 1$. Pour pouvoir utiliser un seul indice dans tous les cas, nous utiliserons la commande `reshape`.

On peut alors parcourir les indices de la matrice comme un simple vecteur : les cases sont numérotées de gauche à droite, du haut vers le bas, à partir de l'indice 0. Par exemple, la case qui dans la matrice correspond à la ligne d'indice 0 et à la colonne d'indice 2 (celle en haut à droite) sera numéroté 2.

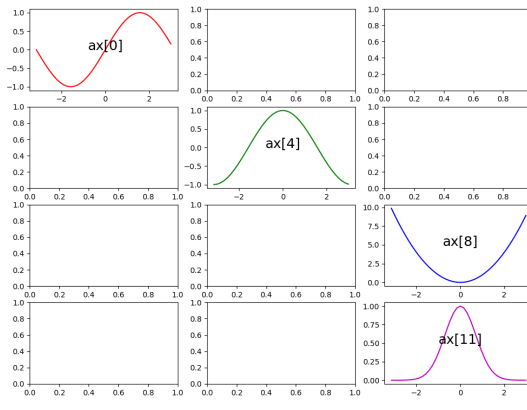
① Exemple avec une grille qui contient une seule ligne de repères (notons qu'on peut afficher plusieurs courbes sur le même repère) :

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-np.pi, np.pi, 0.05*np.pi)

fig,ax = plt.subplots(1,3,figsize=(9,3))

ax[0].plot(x, np.sin(x), 'r')
ax[1].plot(x, np.cos(x), 'g')
ax[1].plot(x, np.exp(-x*x), 'm')
ax[2].plot(x, x**3, 'b')

plt.show()
```



② Exemple avec une grille de repères :

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-np.pi, np.pi, 0.05*np.pi)

fig,ax = plt.subplots(4,3,figsize=(12,9))
ax = ax.reshape(-1)

ax[0].plot(x, np.sin(x), 'r')
ax[4].plot(x, np.cos(x), 'g')
ax[8].plot(x, x*x, 'b')
ax[11].plot(x, np.exp(-x*x), 'm')

plt.show()
```



On peut même effacer les repères inutilisés :

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-np.pi, np.pi, 0.05*np.pi)

fig,ax = plt.subplots(4,3,figsize=(12,9))
ax = ax.reshape(-1)

ax[0].plot(x, np.sin(x), 'r')
ax[4].plot(x, np.cos(x), 'g')
ax[8].plot(x, x*x, 'b')
ax[11].plot(x, np.exp(-x*x), 'm')

for i in [1,2,3,5,6,7,9,10]:
    fig.delaxes(ax[i])

plt.show()
```

8.5.2. Approche II (à la MATLAB)

Une autre approche est la suivante : la fonction `plt.subplot(r,c,i)` (sans "**s**") subdivise la fenêtre sous forme d'une matrice de `r` lignes et `c` colonnes où chaque case est numérotée et `i` est le numéro de la case où afficher le graphe. La numérotation se fait de gauche à droite, puis de haut en bas, en commençant par 1 (bien noter la différence avec l'approche précédente où la numérotation commençait par 0). Par exemple, si on écrit `plt.subplot(4,3,3)`, cela signifie que la fenêtre principale a été subdivisée en $4 \times 3 = 12$ cases ; la case qui a pour indice 3 est celle en haut à droite.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-np.pi, np.pi, 0.05*np.pi)

plt.subplot(4,3,1)
plt.plot(x, np.sin(x), 'r')

plt.subplot(4,3,5)
plt.plot(x, np.cos(x), 'g')

plt.subplot(4,3,9)
plt.plot(x, x*x, 'b')

plt.subplot(4,3,12)
plt.plot(x, np.exp(-x*x), 'm')

plt.show()
```

8.6. ☆ Animations

`FuncAnimation` est une fonction de la classe `matplotlib.animation`. Elle permet de créer une animation à partir d'une fonction.

Nous allons construire un exemple simple d'animation d'un point le long d'une fonction sinusoïdale. On commence par importer les modules nécessaires :

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

Nous allons ensuite tracer la fonction sinus :

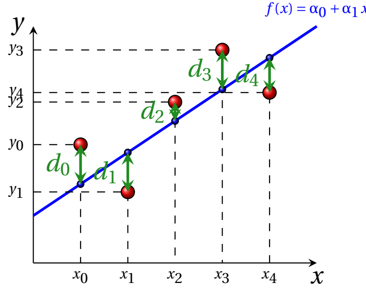
```
my_fig = plt.figure()
xx = np.arange(0, 2*np.pi, 0.001)
yy = np.sin(xx)
plt.plot(xx,yy)
```

Ensuite, nous allons créer le point rouge que nous allons ensuite animer. Pour créer le point rouge, nous commençons par $(0, \sin(0))$. Nous utilisons `dot,` car `plt.plot` renvoie un tuple (et nous ajoutons la virgule après la variable pour décompresser le tuple).

```
dot, = plt.plot([0], [np.sin(0)], 'ro') # NB la virgule après dot!!!
def my_func(i):
    dot.set_data(i, np.sin(i))
    return dot, # NB la virgule !!!

_animation = FuncAnimation(fig=my_fig, func=my_func, frames=np.arange(0, 2*np.pi, 0.1),
    interval=10)
plt.show()
```

8.7. ☆ Régression : polynôme de meilleure approximation

Supposons que deux grandeurs x et y sont liées approximativement par une relation affine, *i.e.* y est peu différent de $f(x) = \alpha_0 + \alpha_1 x$ (autrement dit, lorsqu'on affiche ces points dans un plan cartésien, les points ne sont pas exactement alignés mais cela semble être dû à des erreurs de mesure). On souhaite alors trouver les constantes α_0 et α_1 pour que la droite d'équation $y = \alpha_0 + \alpha_1 x$ s'ajuste *le mieux possible* aux points observés. Pour cela, introduisons $d_i(\alpha_0, \alpha_1) \equiv y_i - (\alpha_0 + \alpha_1 x)$ l'écart vertical du point (x_i, y_i) par rapport à la droite :



La méthode des moindres carrés est celle qui choisit α_0 et α_1 de sorte que *la somme des carrés de ces écarts soit minimale*. La droite d'équation $y = \alpha_1 x + \alpha_0$ ainsi calculée s'appelle *droite de régression de y par rapport à x*.

On peut généraliser cette approche en supposant que les deux grandeurs x et y sont liées par une relation polynomiale, c'est-à-dire de la forme $y = \sum_{j=0}^{m} a_j x^j$ pour certaines valeurs de a_j. On souhaite alors trouver les $m+1$ constantes a_j pour que le polynôme d'équation $f(x) = \sum_{j=0}^{m} a_j x^j$ s'ajuste le mieux possible aux points observés.

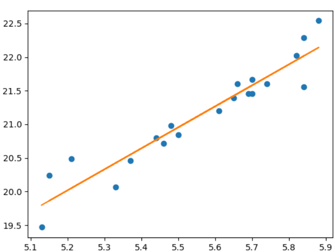
La fonction au cœur de la régression est `polyfit` du module `numpy`. Elle s'utilise de la façon suivante :

```
import numpy as np
coeff = np.polyfit(xx,yy,n)
```

où `xx` et `yy` désignent respectivement la liste des abscisses et des ordonnées des points du nuage de points et n est le degré du polynôme de meilleure approximation. `coeff` est un tuple qui contient les coefficients du polynôme cherché, dans l'ordre décroissant des puissances. Par exemple, si on veut approcher le nuage de points par un polynôme de la forme $ax^2 + bx + c$, on écrira `a,b,c = np.polyfit(xx,yy,2)`.

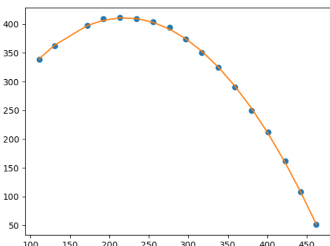
Exemple de régression linéaire :

```
import matplotlib.pyplot as plt
import numpy as np

# Les listes des abscisses et ordonnées :
xx =[ 5.13 ,  5.7,  5.48,  5.7,  5.66, 5.84,  5.5,  5.69,  5.44,  5.84, 5.82,  5.88,  5.61,
↪ 5.65,  5.21, 5.37,  5.46,  5.33,  5.15,  5.74]
yy =[ 19.47,  21.67,  20.98,  21.46 , 21.6,  22.29,  20.84,  21.46, 20.8,  21.56,  22.02 ,
↪ 22.54, 21.2,  21.39,  20.49,  20.46, 20.72,  20.07 ,  20.24,  21.6]

a,b = np.polyfit(xx,yy,1)
plt.plot(xx,yy,"o")
plt.plot(xx,a*xx+b)
plt.show()
```



Exemple de régression d'ordre 2 :

```
import matplotlib.pyplot as plt
import numpy as np

# Les listes des abscisses et ordonnées :
xx = [111, 131, 172, 192, 213, 234, 255, 276, 297, 317, 338, 359, 380, 401, 423, 442, 462]
yy = [339, 362, 398, 409, 411, 409, 404, 394, 374, 350, 325, 290, 250, 212, 162, 108, 51]

a,b,c = np.polyfit(xx,yy,2)
plt.plot(xx,yy,"o")
plt.plot(xx,[a*x**2+b*x+c for x in xx])
plt.show()
```



Même si la relation entre deux quantités n'est pas linéaire, il est parfois possible d'appliquer une transformation pour trouver une relation linéaire.

***Fitting* linéaire après transformation d'un exponentiel** Soit $a > 0$ et considérons la fonction $f(x) = ae^{kx}$: elle est non-linéaire mais si on prend son logarithme on obtient $\ln(f(x)) = \ln(a) + kx$ qui est linéaire et a la forme $\alpha_0 + \alpha_1 x$ avec $\alpha_1 = k$ et $\alpha_0 = \ln(a)$.
On peut alors calculer l'équation de la droite de régression sur l'ensemble $\{ (x_i, \ln(y_i)) \}_{i=0}^{n}$ et obtenir ainsi k et $\ln(a)$.

***Fitting* linéaire après transformation d'une puissance** Soit $a > 0$ et considérons la fonction $f(x) = ax^k$: elle est non-linéaire mais si on prend son logarithme on obtient $\ln(f(x)) = \ln(a) + k\ln(x)$ qui est linéaire et a la forme $\alpha_0 + \alpha_1 x$ avec $\alpha_1 = k$ et $\alpha_0 = \ln(a)$.
On peut alors calculer l'équation de la droite de régression sur l'ensemble $\left\{ (\ln(x_i), \ln(y_i)) \right\}_{i=0}^n$ et obtenir ainsi k et $\ln(a)$.

Exemple d'étude de complexité.
Considérons un algorithme qui dépend d'un entier n. On veut savoir comment le temps t d'exécution évolue lorsqu'on n augmente. Par exemple, si la croissance est polynomiale, on cherchera à calculer p tel que $n \mapsto t(n) \approx cn^p$. Pour cela on calcule d'abord le temps d'exécution pour différents valeurs de n, puis on affiche $n \mapsto t(n)$ en échelle logarithmique. On obtient une droite ($\ln(t) = \ln(c) + p\ln(n)$) de pente p. Pour estimer cette pente on utilisera `polyfit`.

Dans l'exemple suivant on estime la complexité de l'ajout d'un terme dans une liste par la methode `append`. Pour chaque valeur de N, on calcule combien de temps est nécessaire pour remplir la liste et on le sauvegarde. À la fin on aura un ensemble de points $\{(\mathrm{N}_i, \mathrm{T}_i)\}_{i=0}^4$ et on pourra chercher la droite de meilleur appproximation de l'ensemble $\{(\ln(\mathrm{N}_i), \ln(\mathrm{T}_i)\}_{i=0}^4$.

```
import numpy as np
from time import perf_counter

NN = [ 10**2, 10**3, 3*10**3, 5*10**3 ]
TT = []

for N in NN:
    debut=perf_counter()
    L = []
    for i in range(N):
        for j in range(N):
            L.append(i+j)
    fin = perf_counter()
    TT.append(fin-debut)

p,b = np.polyfit(np.log(NN),np.log(TT),1)
print(f"Pente = {p}")

# # AFFICHAGE
# import matplotlib.pyplot as plt
# plt.loglog(NN,TT,"o")
# plt.loglog(NN,[np.exp(b)*n**p for n in NN])
# plt.show()
```

8.7.1. Quelque référence

- Référence complète de `matplotlib`
- http://jeffskinnerbox.me/notebooks/matplotlib-2d-and-3d-plotting-in-ipython.html
- http://apprendre-python.com/page-creer-graphiques-scientifiques-python-apprendre
- https://www.courspython.com/introduction-courbes.html
- https://jakevdp.github.io/PythonDataScienceHandbook/04.08-multiple-subplots.html
- Latest release 3.3 : cool features of Matplotlib https://towardsdatascience.com/latest-cool-features-of-matplotlib-c7a1e2c060c1

8.8. Exercices

Canevas

Affichage du graphe de la fonction $f\colon [a;b] \to \mathbb{R}$ définie par $y = f(x)$ avec $n+1$ points :

```
import matplotlib.pyplot as plt
import numpy as np

f = lambda x : # à compléter
a,b,n = # à compléter

xx = np.linspace(a,b,n+1) # n+1 points, n sous-intervalles de largeur (b-a)/n
yy = [f(x) for x in xx] # si f est une fonction numpy, on écrira yy = np.f(xx)
plt.plot(xx,yy)
plt.grid()
plt.title("Ma jolie figure")
plt.show()
```

Exercice 8.1 (Bhaskara I)

Tracer dans le même repère le graphe représentatif des fonctions

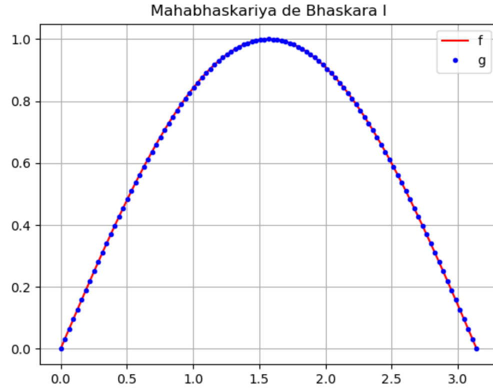
$$f(x) = \sin(x), \qquad g(x) = \frac{16(\pi - x)x}{5\pi^2 - 4(\pi - x)x}, \qquad x \in [0, \pi].$$

Ajouter une grille, la légende, un titre.

Correction

Bhāskara I est un mathématicien indien du VII siècle. Il donna cette unique et remarquable approximation rationnelle de la fonction sinus.

Source : https://scholarworks.umt.edu/cgi/viewcontent.cgi?article=1313&context=tme




```
import matplotlib.pyplot as plt
import numpy as np

f = lambda x : np.sin(x)
g = lambda x :
↪ 16*x*(np.pi-x)/(5*np.pi**2-4*(np.pi-x)*x)

xx = np.linspace(0,np.pi,101)
yy = [f(x) for x in xx]
zz = [g(x) for x in xx]

plt.plot(xx,yy,'r-',label=('f'))
plt.plot(xx,zz,'b.',label=('g'))
plt.title("Mahabhaskariya de Bhaskara I")
plt.grid()
plt.legend(loc="best")
plt.show()
```

Notons qu'on peut même écrire `yy = f(xx)` et `zz = g(xx)` car xx est un vecteur numpy (les opérations élémentaires sont vectorisées).

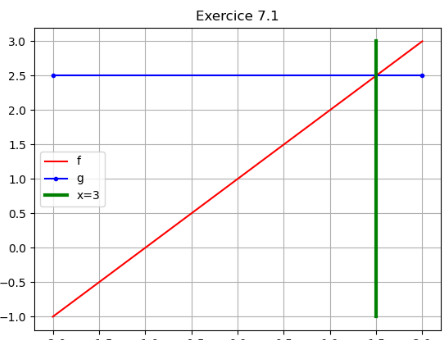
Exercice 8.2 (Tracer des droites)

Tracer dans le même repère les droites suivantes sur $[-2;2]$ (il s'agit donc bien-sûr de segments) :

- $f(x) = x + 1$

- $g(x) = 2.5$
- $x = 1.5$

Ajouter une grille, la légende, un titre.




```
import matplotlib.pyplot as plt
import numpy as np

xx = np.linspace(-2,2,101)

f  = lambda x : x+1
ff = [f(x) for x in xx]
plt.plot(xx,ff,'r-',label=('f'))

# horizontale
g  = lambda x : 2.5
gg = [g(x) for x in xx]
plt.plot(xx,gg,'r-',label=('g'))

# verticale
yy = np.linspace(-2,2,101)
h  = lambda y : 1.5
hh = [h(y) for y in yy]
plt.plot(hh,yy,'g-',lw=3,label=('x=3'))

plt.title("Exercice 8.1")
plt.grid()
plt.legend(loc="best")
plt.show()
```

Notons qu'on peut même écrire `yy = xx+1` car `xx` est un vecteur numpy (les opérations élémentaires sont vectorisées).

Étant donné que pour définir un segment il suffit d'imposer le passage par deux points, on peut tracer les mêmes graphes comme ci-dessous :

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([-2,2],[-1,3],'r-',label=('f')) # segment d'extrémités (-2,-1) et (2,3)
plt.plot([-2,2],[2.5,2.5],'b.-',label=('g')) # segment d'extrémités (-2,2.5) et (2,2.5)
plt.plot([1.5,1.5],[-1,3],'g-',lw=3,label=('x=3')) # segment d'extrémités (1.5,-1) et (1.5,3)
plt.show()
```

Exercice 8.3 (Tracer une fonction définie par morceaux)

Tracer le graphe de la fonction

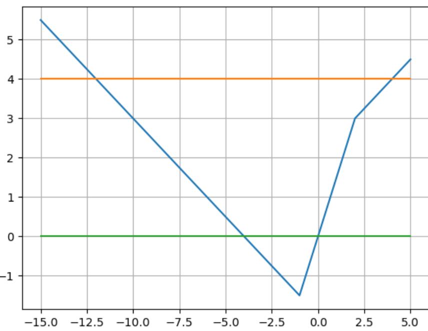
$$f\colon \mathbb{R} \to \mathbb{R}$$
$$x \mapsto |x+1| - \left|1 - \frac{x}{2}\right|$$

Résoudre d'abord graphiquement puis en utilisant la fonction `fsolve` du module `scipy.optimize` l'équation $f(x) = 0$ et l'inéquation $f(x) \le 4$.

Bonus : même question avec le module pour le calcul formel `sympy`.

Correction

En affichant une grille et en utilisant le zoom, on peut conjecturer que $f(x) = 0$ pour $x \approx -4$ et $x \approx 0$. On affiche alors $f(-4)$ et $f(0)$. De même, on peut conjecturer que $f(x) \le 4$ pour $-12 \lesssim x \lesssim 4$. On affiche alors $f(-12)$ et $f(4)$.



```
import matplotlib.pyplot as plt
import numpy as np
f  = lambda x : abs(x+1)-abs(1-x/2)
xx = np.linspace(-15,5,101)
yy = [f(x) for x in xx]
plt.plot(xx,yy)
plt.plot([-15,5],[4,4])
plt.plot([-15,5],[0,0])
plt.grid()
plt.show()
print(f"f(-4)={f(-4)}, f(0)={f(0)}")
print(f"f(-12)={f(-12)}, f(4)={f(4)}")
```

f(-4)=0.0, f(0)=0.0 f(-12)=4.0, f(4)=4.0

Variante : il s'agit d'une fonction affine par morceaux, on peut donc juste relier des segments :

```
import matplotlib.pyplot as plt
import numpy as np
f  = lambda x : abs(x+1)-abs(1-x/2)
xx = [-15,-1,2,5]
yy = [f(x) for x in xx]
plt.plot(xx,yy)
plt.plot([-15,5],[4,4])
plt.plot([-15,5],[0,0])
plt.grid()
plt.show()
print(f"f(-4)={f(-4)}, f(0)={f(0)}")
print(f"f(-12)={f(-12)}, f(4)={f(4)}")
```

Bonus :

```
import sympy
x = sympy.Symbol('x', real=True)
sol = sympy.solve(abs(x+1)-abs(1-x/2),x)
print(f"f(x)=0 ssi x appartient à l'ensemble {sol}")
print("(NB ce n'est pas un intervalle mais l'ensemble des solutions de l'équation)")
sol = sympy.solve(abs(x+1)-abs(1-x/2)<4,x)
print(f"f(x)<0 ssi {sol}")
```

```
f(x)=0 ssi x appartient à l'ensemble [-4, 0]
(NB ce n'est pas un intervalle mais l'ensemble des solutions de l'équation)
f(x)<0 ssi (-12 < x) & (x < 4)
```

☆ Exercice Bonus 8.4 (Approximations de deux fonctions polynomiales)

Les fonctions

$$f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1, \qquad \text{et} \qquad g(x) = (x-1)^7$$

sont égales mais tracées dans l'intervalle $0.988 \le x \le 1.012$ montrent une différence significative. Expliquer l'origine de cette différence.

Correction

L'expression `f(x)` est calculée de façon approchée à proximité de $x = 1$ avec des + et des − avec des valeurs supérieurs à 35×1024^4, ce qui entraîne une perte de précision dans le tracé de f.

```
import numpy as np

f = lambda x: x**7-7*x**6+21*x**5-35*x**4+35*x**3-21*x**2+7*x-1
g = lambda x: (x-1)**7
```

```
# Affichage des valeurs de f et g pour x ≈ 1
from tabulate import tabulate
T = []
T.append(['x', 'f(x)', 'g(x)'])
for x in np.linspace(0.988, 1.012, 11):
    T.append( [x, f(x) , g(x)] )
print(tabulate(T, headers='firstrow', floatfmt='1.7e'))

# Affichage des graphes de f et g pour x ≈ 1
import matplotlib.pyplot as plt
x = np.linspace(0.988, 1.012, 10000)
plt.plot(x, f(x), label=r'$f$')
plt.plot(x, g(x), label=r'$g$')
# plt.ylim(0.99995, 1.00005)
plt.legend()
plt.savefig("Images/exo-poly.png")
plt.show()

# Vérifions l'égalité des deux fonctions
import sympy as sp
sp.var('x')
print("Avec sympy: f(x)-g(x) = ", sp.simplify(f(x)-g(x)))
```

```
            x            f(x)            g(x)
-------------  --------------  --------------
9.8800000e-01  -3.5527137e-14  -3.5831808e-14
9.9040000e-01  -1.1546319e-14  -7.5144748e-15
9.9280000e-01  -3.5527137e-15  -1.0030613e-15
9.9520000e-01   3.5527137e-15  -5.8706834e-17
9.9760000e-01   7.1054274e-15  -4.5864714e-19
1.0000000e+00   0.0000000e+00   0.0000000e+00
1.0024000e+00  -1.4210855e-14   4.5864714e-19
1.0048000e+00   0.0000000e+00   5.8706834e-17
1.0072000e+00   8.8817842e-16   1.0030613e-15
1.0096000e+00   4.4408921e-15   7.5144748e-15
1.0120000e+00   4.6185278e-14   3.5831808e-14
Avec sympy: f(x)-g(x) =  0
```

Exercice Bonus 8.5 (Approximations de deux fonctions trigonométriques)

Les fonctions

$$f(x) = \frac{1-\cos^2(x)}{x^2}, \qquad \text{et} \qquad g(x) = \frac{\sin^2(x)}{x^2}$$

sont égales mais tracées dans l'intervalle $-0.001 \le x \le 0.001$ montrent une différence significative. Expliquer l'origine de cette différence.

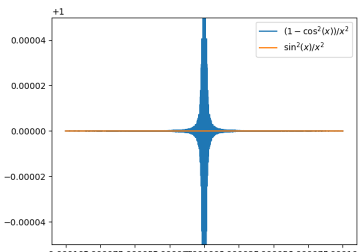
Correction

L'expression `1-np.cos(x)**2` est calculée de façon approchée à proximité de $x = 0$, ce qui entraîne une perte de précision et des oscillations sauvages dans le tracé de f.

```
import numpy as np

f = lambda x: (1 - np.cos(x)**2)/x**2
g = lambda x: (np.sin(x)/x)**2

# Affichage des valeurs de f et g pour x ≈ 0
from tabulate import tabulate
```

```
T = []
T.append(['x', '1 - np.cos(x)**2' , 'np.sin(x)**2' ,  'f(x)', 'g(x)'])
for x in np.linspace(-1.e-7, 1.e-7, 10):
    T.append( [x, 1 - np.cos(x)**2 , np.sin(x)**2, f(x) , g(x)] )
print(tabulate(T, headers='firstrow', floatfmt='1.10e'))

# Affichage des graphes de f et g pour x ≈ 0
import matplotlib.pyplot as plt
x = np.linspace(-1.e-4, 1.e-4, 10000)
plt.plot(x, f(x), label=r'$(1-\cos^2(x))/x^2$')
plt.plot(x, g(x), label=r'$\sin^2(x)/x^2$')
plt.ylim(0.99995, 1.00005)
plt.legend()
plt.savefig("Images/exo-PB.png")
plt.show()
```

```
                x     1 - np.cos(x)**2      np.sin(x)**2              f(x)              g(x)
-----------------  -------------------  ----------------  ----------------  ----------------
-1.0000000000e-07    9.9920072216e-15  1.0000000000e-14  9.9920072216e-01  1.0000000000e+00
-7.7777777778e-08    5.9952043330e-15  6.0493827160e-15  9.9104398157e-01  1.0000000000e+00
-5.5555555556e-08    3.1086244690e-15  3.0864197531e-15  1.0071943279e+00  1.0000000000e+00
-3.3333333333e-08    1.1102230246e-15  1.1111111111e-15  9.9920072216e-01  1.0000000000e+00
-1.1111111111e-08    2.2204460493e-16  1.2345679012e-16  1.7985612999e+00  1.0000000000e+00
 1.1111111111e-08    2.2204460493e-16  1.2345679012e-16  1.7985612999e+00  1.0000000000e+00
 3.3333333333e-08    1.1102230246e-15  1.1111111111e-15  9.9920072216e-01  1.0000000000e+00
 5.5555555556e-08    3.1086244690e-15  3.0864197531e-15  1.0071943279e+00  1.0000000000e+00
 7.7777777778e-08    5.9952043330e-15  6.0493827160e-15  9.9104398157e-01  1.0000000000e+00
 1.0000000000e-07    9.9920072216e-15  1.0000000000e-14  9.9920072216e-01  1.0000000000e+00
```



Source : https://scipython.com/book/chapter-9-general-scientific-programming/questions/floating-point-approximations-to-two-trigonometric-functions/

Exercice 8.6 (Module `scipy` - Calcul approché d'une intégrale)
Utiliser `scipy.integrate.quad` pour évaluer l'intégrale

$$\int_0^6 f(x)\,\mathrm{d}x.$$

avec

$$f(x) = \lfloor x \rfloor - 2\left\lfloor \frac{x}{2} \right\rfloor.$$

Après avoir tracé le graphe de la fonction f sur l'intervalle $[0,6]$, calculer la valeur exacte de l'intégrale et la comparer à la valeur approchée obtnue.

Source : https://scipython.com/book2/chapter-8-scipy/questions/numerical-integration-of-a-simple-function/

Correction
La fonction f se réécrit comme

$$f(x)=\begin{cases}0 & \text{si } 2k<x<2k+1\\ 1 & \text{sinon}\end{cases}$$

pour $k\in\mathbb{Z}$. L'intégrale cherchée vaut donc 3.

```
import numpy as np
func = lambda x: np.floor(x) - 2*np.floor(x/2)

from scipy.integrate import quad
val, err = quad(func, 0, 6)
print(f"L'integrale vaut approximativement {val:.2f}. L'erreur est inférieure à {err:.2f}.")

# import matplotlib.pyplot as plt
# xx = np.linspace(0,6,100)
# plt.plot(xx,func(xx))
# plt.show()
```

```
L'integrale vaut approximativement 3.00. L'erreur est inférieure à 0.00.
```

Exercice 8.7 (Dépréciation ordinateur)

On achète un ordinateur portable à 430 €. On estime qu'une fois sorti du magasin sa valeur u_n en euro après n mois est donnée par la formule

$$u_n=40+300\times(0.95)^n.$$

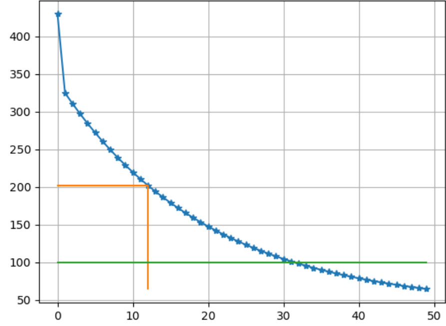
- Que vaut l'ordinateur à la sortie du magasin ? (afficher u_0)
- Que vaut après un an de l'achat ? (afficher u_{12})
- À long terme, à quel prix peut-on espérer revendre cet ordinateur ?
- Déterminer le mois à partir duquel l'ordinateur aura une valeur inférieure à 100 €.

Correction

- À la sortie du magasin $u_0=340$
- Après un an de l'achat on a $u_{12}=40+300\times(0.95)^{12}=202.11$
- À long terme, on peut espérer revendre cet ordinateur à $\lim_{n\to+\infty}u_n=40$.
- À partir du 32-ème mois l'ordinateur aura une valeur inférieure à 100 € car :

$$40+300\times(0.95)^n<100 \iff (0.95)^n<\frac{100-40}{300}=\frac{1}{5}=5^{-1}$$
$$\iff n\ln(0.95)<-\ln(5) \iff n>-\frac{\ln(5)}{\ln(0.95)}\simeq 31.377$$

Vérifions nos calculs :




```
import matplotlib.pyplot as plt
import numpy as np
nn = range(50)
uu = [430]+[40+300*0.95**n for n in nn[1:]]
i = ([u>100 for u in uu]).count(True)
print(f"u[{i-1}]={uu[i-1]} et u[{i}]={uu[i]}")
plt.plot(nn,uu,'*-')
plt.plot([nn[0],nn[12],nn[12]],
         [uu[12],uu[12],uu[-1]])
plt.plot([nn[0],nn[-1]],[100,100])
plt.grid()
plt.show()
```

u[31]=101.17204772373711 et u[32]=98.11344533755026

Exercice 8.8 (Mercato)

C'est le mercato et un attaquant est convoité par deux clubs F et G qui lui proposent le même salaire mensuel mais des systèmes différents pour les primes par but marqué :

- le club F lui propose une prime de 8 000 euros par but marqué pour les dix premiers buts marqués puis de 11 300 euros par but marqué à partir du onzième but ;
- le club G lui propose une prime de 10 000 euros par but marqué quel que soit le nombre de buts inscrits.

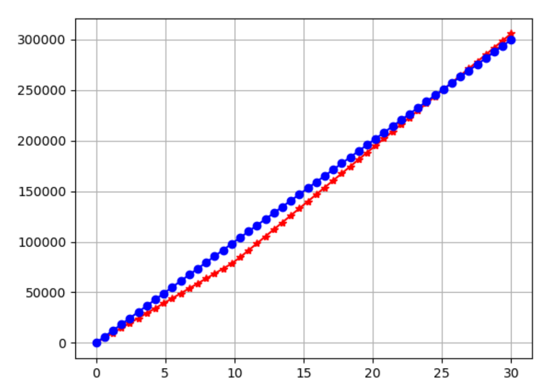
1. Écrire des fonctions `F(n)` et `G(n)` qui retournent respectivement les montants des primes offertes par les clubs F et G en fonction du nombre n de buts marqués.
2. Avec une boucle `while` déterminer le nombre de buts que doit marquer l'attaquant pour que le montant de la prime offerte par le club F soit la plus intéressante.
3. Tracer le graphe des deux fonctions pour vérifier le résultat
4. Demander au module `scipy` de calculer la solution de $F(n) = G(n)$.

Correction

On a les deux fonctions

$$F(b) = \begin{cases} 8000n & \text{si } n \le 10, \\ 11300(n-10) + 8000 \times 10 & \text{sinon;} \end{cases}$$

$$G(b) = 10000b.$$




```
F = lambda b : (b*8000) if (b<=10) else (8000*10+11300*(b-10))
G = lambda b : 10000*b

n=1
while F(n)<G(n):
    n+=1
print(n)

import matplotlib.pyplot as plt
import numpy as np
nn = np.linspace(0,30)
ff = [F(b) for b in nn]
gg = [G(b) for b in nn]
plt.figure(num=None, figsize=(20, 10))
plt.plot(nn,ff,'r*-',nn,gg,'bo-')
plt.grid()
plt.show()

from scipy.optimize import fsolve
print(fsolve(lambda n:F(n)-G(n),20))
```

26 [25.38461538]

Exercice 8.9 (Résolution graphique d'une équation)

Soit la fonction

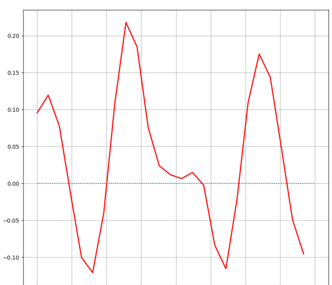
$$f \colon [-10,10] \to \mathbb{R}$$
$$x \mapsto \frac{x^3\cos(x)+x^2-x+1}{x^4-\sqrt{3}x^2+127}$$

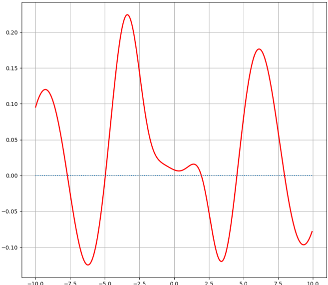
1. Tracer le graphe de la fonction f en utilisant seulement les valeurs de $f(x)$ lorsque la variable x prend successivement les valeurs $-10, -9.2, -8.4, \dots, 8.4, 9.2, 10$ (*i.e.* avec un pas 0.8).
2. Apparemment, l'équation $f(x) = 0$ a une solution α voisine de 2. En utilisant le zoom, proposer une valeur approchée de α.
3. Tracer de nouveau le graphe de f en faisant varier x avec un pas de 0.05. Ce nouveau graphe amène-t-il à corriger la valeur de α proposée ?
4. Demander au module `scipy` d'approcher α.

Correction

On affiche le graphe de $y = f(x)$ et la graphe de $y = 0$. En utilisant un pas de 0.8 il semblerai que $\alpha = 1.89$. En utilisant un pas de 0.05 il semblerai que $\alpha = 1.965$. En utilisant la fonction `fsolve` on trouve $\alpha = 1.96289995$.

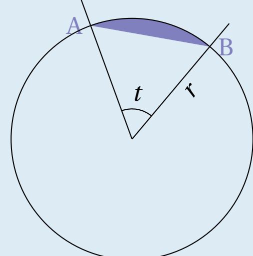
```
import matplotlib.pyplot as plt
import numpy as np

f = lambda x:
↪  (x**3*np.cos(x)+x**2-x+1)/(x**4-np.sqrt(3)*x**2+127)

fig,ax = plt.subplots(2,1,figsize=(10, 20))

xx = np.arange(-10,10,0.8)
yy = f(xx)
ax[0].plot(xx,yy,'r-',lw=2)
ax[0].plot([-10,10],[0,0],':')
ax[0].grid()

xx = np.arange(-10,10,0.05)
yy = f(xx)
ax[1].plot(xx,yy,'r-',lw=2)
ax[1].plot([-10,10],[0,0],':')
ax[1].grid()

plt.show()

from scipy.optimize import fsolve
print(fsolve(f,1.9))
```





[1.96289995]

Exercice 8.10 (Résolution graphique d'une équation)



Considérons un cercle de rayon r. Si nous traçons un angle t (mesuré en radians) à partir du centre du cercle, les deux rayons formant cet angle coupent le cercle en A et B. Nous appelons a l'aire délimitée par la corde et l'arc AB (en bleu sur le dessin). Cette aire est donnée par $a = \frac{r^2}{2}(t - \sin(t))$. Pour un cercle donné (c'est à dire un rayon donné), nous choisissons une aire (partie en bleu) a. Quelle valeur de l'angle t permet d'obtenir l'aire choisie ? Autrement dit, connaissant a et r, nous voulons déterminer t solution de l'équation

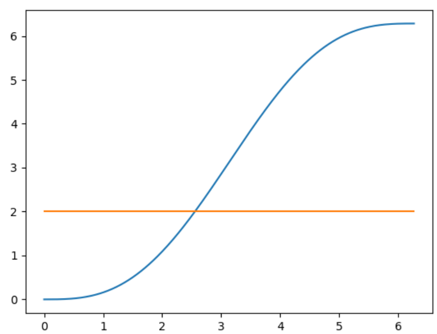
$$\frac{2a}{r^2} = t - \sin(t).$$

1. Résoudre graphiquement l'équation en traçant les courbes correspondant aux membres gauche et droit de l'équation (pour $a = 4$ et $r = 2$). Quelle valeur de t est solution de l'équation ?
2. Comment faire pour obtenir une valeur plus précise du résultat ?

Correction

```
import matplotlib.pyplot as plt
import numpy as np
a, r = 4, 2
tt = np.arange(0,2*np.pi,np.pi/180)
rhs = tt-np.sin(tt)
# rhs = [t-np.sin(t) for t in tt]
lhs = [2*a/r**2 for t in tt]
plt.plot(tt,rhs,tt,lhs)
plt.show()

from scipy.optimize import fsolve
f = lambda t: t-np.sin(t)-2*a/r**2
print(fsolve(f,2.5))
```

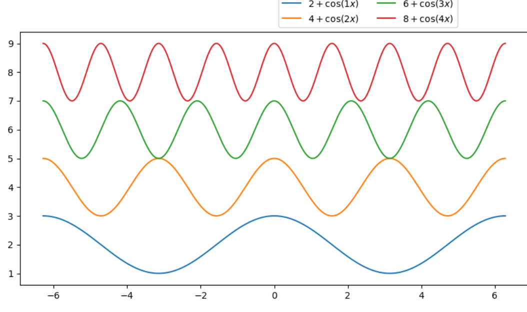
[2.55419595]



Le graphe montre que la solution est entre 2 et 3. On peut alors calculer une solution approchée avec `fsolve`.

Exercice 8.11 (Tracer plusieurs courbes)

Tracer dans le même repère le graphe des fonctions $f_n(x) = 2n + \cos(nx)$ pour $n = 1,2,3,4$.




```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(num=None, figsize=(10, 5))
f = lambda x,n : 2*n+np.cos(n*x)
xx = np.linspace(-2*np.pi,2*np.pi,1001)
for n in range(1,5):
    yy = [f(x,n) for x in xx]
    plt.plot(xx,yy,
        label=fr'${2*n}+\cos({n}x)$')

plt.legend(bbox_to_anchor=(0.5, 1),ncol=2);
plt.show()
```

Attention

Jusqu'ici nous avons représenté des courbes engendrées par des équations cartésiennes, c'est-à-dire des fonctions de la forme $y = f(x)$. Pour cela, nous générons d'abord un ensemble de valeurs $\{x_i\}_{i=0...N}$ puis l'ensemble de valeurs $\{y_i\}_{i=0...N}$ avec $y_i = f(x_i)$. Il existe d'autres types de courbes comme par exemple les courbes paramétrées (engendrées par des équations paramétriques). Les équations paramétriques de courbes planes sont de la forme

$$\begin{cases} x = u(t), \\ y = v(t), \end{cases}$$

où u et v sont deux fonctions cartésiennes et le couple $(x; y)$ représente les coordonnées d'un point de la courbe paramétrée. La courbe engendrée par l'équation cartésienne $y = f(x)$ est une courbe paramétrée car il suffit de poser $u(t) = t$ et $v(t) = f(t)$. Un type particulier de courbe paramétrique est constitué par les équations polaires de courbes planes qui sont de la forme [a]

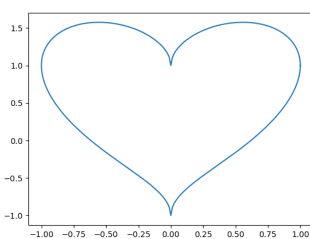
$$\begin{cases} x = r(t)\cos(t), \\ y = r(t)\sin(t). \end{cases}$$

a. r représente la distance de l'origine O du repère $(O; \mathbf{i}, \mathbf{j})$ au point $(x; y)$ de la courbe, et t l'angle avec l'axe des abscisses.

Exercice 8.12 (Courbe paramétrée)

Tracer la courbe mystère suivante pour $t \in [0; 2\pi]$:

$$\begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t) + \sqrt{|\cos(t)|} \end{cases}$$



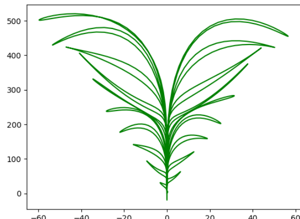
```
import matplotlib.pyplot as plt
import numpy as np
x = lambda t : np.cos(t)
y = lambda t : np.sin(t)+np.sqrt(abs(np.cos(t)))
tt = np.linspace(0,2*np.pi,501)
xx = [x(t) for t in tt]
yy = [y(t) for t in tt]
plt.plot(xx,yy)
# plt.fill(xx,yy,color='pink')
plt.show()
```

Notons qu'on peut écrire directement `xx = x(tt)` et `yy = y(tt)` car les fonctions $t \mapsto x(t)$ et $t \mapsto y(t)$ sont des fonctions définies par composition de fonctions `numpy` qui sont donc vectorisées.

Exercice 8.13 (Courbe paramétrée)

Tracer la courbe mystère suivante pour $t \in [0; 39\pi/2]$:

$$\begin{cases} x(t) = t\cos^3(t) \\ y(t) = 9t\sqrt{|\cos(t)|} + t\sin(t/5)\cos(4t) \end{cases}$$

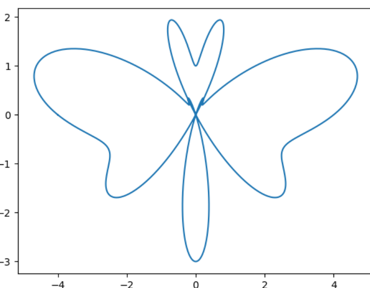


```
from numpy import linspace, cos, sin, sqrt, pi
import matplotlib.pyplot as plt
tt = linspace(0,39*pi/2,1000)
xx = tt*cos(tt)**3
yy = 9*tt*sqrt(abs(cos(tt)))+tt*sin(0.2*tt)*cos(4*tt)
plt.plot(xx,yy,c="green")
plt.show();
```

Exercice 8.14 (Courbe polaire)

Tracer la courbe papillon ($t \in [0; 2\pi]$) :

$$\begin{cases} x(t) = r(t)\cos(t) \\ y(t) = r(t)\sin(t) \end{cases} \quad \text{avec } r(t) = \sin(7t) - 1 - 3\cos(2t).$$



```
import matplotlib.pyplot as plt
import numpy as np
r = lambda t : np.sin(7*t)-1-3*np.cos(2*t)
x = lambda t : r(t)*np.cos(t)
y = lambda t : r(t)*np.sin(t)
tt = np.linspace(0,2*np.pi,1001)
xx = [x(t) for t in tt]
yy = [y(t) for t in tt]
plt.plot(xx,yy)
plt.show()
```

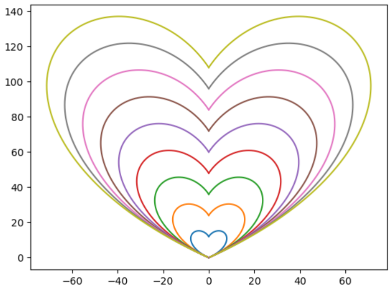
La fonction $t \mapsto r(t)$ est une fonction définie par composition de fonctions `numpy` qui sont vectorisées, on peut alors écrire directement

```
import matplotlib.pyplot as plt
import numpy as np
r = lambda t : np.sin(7*t)-1-3*np.cos(2*t)
tt = np.linspace(0,2*np.pi,1001)
xx = r(tt)*np.cos(tt)
yy = r(tt)*np.sin(tt)
plt.plot(xx,yy)
plt.show()
```

Exercice 8.15 (Courbe polaire)

Pour $h = \frac{1}{10}$ et $k = h, 2h, \dots, 1$, tracer la courbe suivante ($t \in [0; 60]$) :

$$\begin{cases} x(t) = \pm r(t)\sin(\pi t/180) \\ y(t) = r(t)\cos(\pi t/180) \end{cases} \quad \text{avec } r(t) = kh(-t^2 + 40t + 1200).$$



```
from numpy import cos, sin, sqrt, pi, arange
import matplotlib.pyplot as plt
h = 0.1
tt = arange(0,60,h)
idx = 0
for k in arange(h,1,h):
    rr = k*h*(-tt**2+40*tt+1200)
    xx_R =  rr*sin(pi*tt/180)
    yy_R =  rr*cos(pi*tt/180)
    xx_L = -rr*sin(pi*tt/180)
    yy_L = yy_R
    plt.plot(xx_L,yy_L,color='C'+str(idx))
    plt.plot(xx_R,yy_R,color='C'+str(idx))
    idx += 1
plt.show()
```

Exercice 8.16 (Proies et prédateurs)

En 1926, le mathématicien Volterra a proposé un modèle pour étudier l'évolution de deux populations, en l'occurrence les sardines et les requins de l'Adriatique. Ce modèle décrit les fluctuations périodiques des effectifs des deux espèces, avec la même période mais en étant décalées dans le temps, en fonction de l'incidence de l'une sur l'autre. Les équations utilisées pour modéliser ces fluctuations sont des équations différentielles, mais nous nous concentrerons ici sur le problème discrétisé à l'aide de suites.

Le modèle suppose que le nombre de proies et de prédateurs varie dans le temps, et on note e le nombre de proies et c le nombre de prédateurs à l'instant t. Dans le modèle de Volterra, le taux de reproduction des proies entre les instants t_n et t_{n+1} est supposé constant et noté A, tandis que le taux de mortalité des prédateurs entre les instants t_n et t_{n+1} est supposé constant et noté C. Le taux de mortalité des proies dû aux prédateurs est supposé proportionnel au nombre de prédateurs, avec un coefficient de proportionnalité noté B, et le taux de reproduction des prédateurs en fonction des proies mangées est également supposé proportionnel au nombre de proies, avec un coefficient de proportionnalité noté D.

Les équations des proies et des prédateurs s'écrivent ainsi :

$$\begin{cases} \dfrac{e_{n+1}-e_n}{e_n} = \mathrm{A} - \mathrm{B}c_n, & \text{équation des proies,} \\ \dfrac{c_{n+1}-c_n}{c_n} = -\mathrm{C} + \mathrm{D}e_n, & \text{équation des prédateurs.} \end{cases}$$

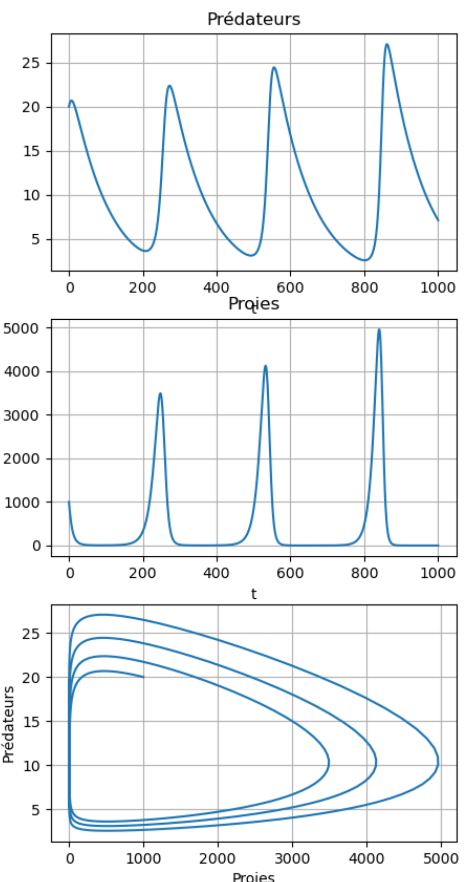
La première équation décrit la variation de la population de proies, dont la croissance est affectée par le taux de prédation des prédateurs. La deuxième équation décrit la croissance de la population de prédateurs, qui est affectée à la fois par leur mortalité naturelle et par le taux de reproduction lié à la consommation de proies. Le système d'équations du modèle de Volterra est alors donné par :

$$\begin{cases} e_{n+1} = e_n(1+\mathrm{A}-\mathrm{B}c_n), & \text{équation des proies,} \\ c_{n+1} = c_n(1-\mathrm{C}+\mathrm{D}e_n), & \text{équation des prédateurs.} \end{cases}$$

Afficher l'évolution des deux populations entre $n = 0$ et $n = 1000$ si $e_0 = 1000$, $c_0 = 20$, $\mathrm{A} = 0.1$, $\mathrm{B} = \mathrm{C} = 0.01$ et $\mathrm{D} = 0.00002$.

Correction

```
import matplotlib.pyplot as plt
import numpy as np

e,c = 1000,20
Er = [e]
Ca = [c]
A,B,C,D = 0.1,0.01,0.01,0.00002
for n in range(1000):
    e,c =  e*(1+A-B*c), c*(1-C+D*e)
    Ca.append(c)
    Er.append(e)

fig,ax = plt.subplots(3,1, figsize=(5,10))

ax[0].plot(range(len(Ca)),Ca)
ax[0].set_xlabel('t')
ax[0].grid()
ax[0].set_title('Prédateurs')

ax[1].plot(range(len(Er)),Er)
ax[1].set_title('Proies')
ax[1].set_xlabel('t')
ax[1].grid()

ax[2].plot(Er,Ca)
ax[2].set_xlabel("Proies")
ax[2].set_ylabel("Prédateurs")
ax[2].grid()

plt.show()
```



Exercice 8.17 (Coïncidences et anniversaires)

Combien faut-il réunir de personne pour avoir une chance sur deux que deux d'entre elles aient le même anniversaire ?

Au lieu de nous intéresser à la probabilité que cet événement se produise, on va plutôt s'intéresser à l'événement inverse : quelle est la probabilité pour que n personnes n'aient pas d'anniversaire en commun (on va oublier les années bissextiles et le fait que plus d'enfants naissent neuf mois après le premier de l'an que neuf mois après la Toussaint.)

- si $n = 1$ la probabilité est 1 (100%) : puisqu'il n'y a qu'une personne dans la salle, il y a 1 chance sur 1 pour qu'elle n'ait pas son anniversaire en commun avec quelqu'un d'autre dans la salle (puisque, fatalement, elle est toute seule dans la salle) ;
- si $n = 2$ la probabilité est $\frac{364}{365}$ (= 99,73%) : la deuxième personne qui entre dans la salle a 364 chances sur 365 pour qu'elle n'ait pas son anniversaire en commun avec la seule autre personne dans la salle ;
- si $n = 3$ la probabilité est $\frac{364}{365} \times \frac{363}{365}$ (= 99,18%) : la troisième personne qui entre dans la salle a 363 chances sur 365 pour qu'elle n'ait pas son anniversaire en commun avec les deux autres personnes dans la salle mais cela sachant que les deux premiers n'ont pas le même anniversaire non plus, puisque la probabilité pour que les deux premiers n'aient pas d'anniversaire en commun est de 364/365, celle pour que les 3 n'aient pas d'anniversaire commun est donc 364/365 × 363/365 et ainsi de suite ;
- si $n = k$ la probabilité est $\dfrac{364}{365} \times \dfrac{363}{365} \times \dfrac{362}{365} \times \cdots \times \dfrac{365-k+1}{365}$.

On obtient la formule de récurrence

$$\begin{cases} P_1 = 1, \\ P_{k+1} = \dfrac{365-k+1}{365} P_k. \end{cases}$$

1. Tracer un graphe qui affiche la probabilité que deux personnes ont la même date de naissance en fonction du nombre de personnes.
2. Calculer pour quel k on passe sous la barre des 50%.

Source : blog http ://eljjdx.canalblog.com/archives/2007/01/14/3691670.html et https://calmcode.io/birthday-problem/birthdays.html

Correction
La probabilité que deux personnes dans un groupe de k personne n'ont pas la même date de naissance est

$$P_{\text{same}}(k) = 1 - P_{\text{no overlap}}(k)$$

et

$$P_{\text{no overlap}}(k) = \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365} \times \cdots \times \frac{365-k+1}{365}.$$

```
from math import prod
k = 3
nP = prod([(365-i+1)/365 for i in range(2,k+1)])
P = 1-nP
print(f"{k = }, {nP = }, {P = }")
```

```
k = 3, nP = 0.9917958341152187, P = 0.008204165884781345
```

Dans un groupe de 23 personnes (`=indice`), il y a plus d'une chance sur deux (`seuil=0.5`) pour que deux personnes de ce groupe aient leur anniversaire le même jour (`tot=365`). Ou, dit autrement, il est plus surprenant de ne pas avoir deux personnes qui ont leur anniversaire le même jour que d'avoir deux personnes qui ont leur anniversaire le même jour (et avec 57, on dépasse les 99% de chances!)

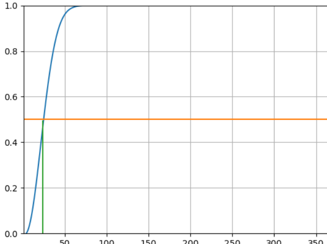
```
from math import prod

def calculate(k):
    nP = prod([(365-i+1)/365 for i in range(2,k+1)])
    P = 1-nP
    return nP, P

k = 23
nP, P = calculate(k)
print(f"{k = }, {nP = }, {P = }")
```

```
k = 23, nP = 0.4927027656760144, P = 0.5072972343239857
```

Cas générale (pour un groupe de $k \in [1;365]$ personne) avec détection du nombre de personnes pour dépasser un seuil imposé :

```
tot, seuil = 365, 0.5
nn = range(1,tot)
P = [1]
for k in range(tot-2):
    P.append( (tot-k+1)*P[k]/tot )
nP = [1-p for p in P]
indice = ([np<seuil for np in nP]).count(True)
print(f"Dans un groupe de {indice-1} personnes on a {seuil*100}\% pour que deux personnes")
print("de ce groupe aient leur anniversaire le même jour")
plt.plot(nn,nP, [min(nn),max(nn)],[seuil,seuil],[indice,indice],[0,P[indice]])
plt.axis([1,tot,0,1])
plt.grid()
```

Dans un groupe de 23 personnes on a 50.0% pour que deux personnes de ce groupe aient leur anniversaire le même jour



On peut s'amuser à adapter les calculs à d'autres problèmes, par exemple on a `seuil=61%` de chances que parmi `indice=5` personnes prises au hasard, deux ont le même signe astrologique (`tot=12`).

Exercice 8.18 (Conjecture de Syracuse)

Considérons la suite récurrente

$$\begin{cases} u_1 \in \mathbb{N}^* \text{ donné,} \\ u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases} \end{cases}$$

En faisant des tests numérique on remarque que la suite obtenue tombe toujours sur 1 peut importe l'entier choisit [a] au départ. La conjecture de Syracuse affirme que, peu importe le nombre de départ choisi, la suite ainsi construite atteint le chiffre 1 (et donc boucle sur le cycle 4, 2, 1). Cet énoncé porte le nom de «Conjecture» et non de théorème, car ce résultat n'a pas (encore) été démontré pour tous les nombres entiers. En 2004, la conjecture a été "juste" vérifiée pour tous les nombres inférieurs à 2^{64}.

1. Écrire un script qui, pour une valeur de $u_1 \in]1;10^6]$ donnée, calcule les valeurs de la suite jusqu'à l'apparition du premier 1.
2. Tracer les valeurs de la suite en fonction de leur position (on appelle cela la trajectoire ou le vol), *i.e.* les points $\{(n,u_n)\}_{n=1}^{n=N}$
3. Calculer ensuite le *durée de vol*, *i.e.* le nombre de terme avant l'apparition du premier 1 ; l'*altitude maximale*, *i.e.* le plus grand terme de la suite et le *facteur d'expansion*, c'est-à-dire l'altitude maximale divisée par le premier terme.

On peut s'amuser à chercher les valeurs de u_1 donnant la plus grande durée de vol ou la plus grandes altitude maximale. On notera que, même en partant de nombre peu élevés, il est possible d'obtenir des altitudes très hautes. Vérifiez que, en partant de 27, elle atteint une altitude maximale de 9232 et une durée de vol de 111. Au contraire, on peut prendre des nombres très grands et voir leur altitude chuter de manière vertigineuse sans jamais voler plus haut que le point de départ. Faire le calcul en partant de 10^6.

Ce problème est couramment appelé Conjecture de Syracuse (mais aussi problème de Syracuse, algorithme de Hasse, problème de Ulam, problème de Kakutani, conjecture de Collatz, conjecture du $3n+1$). Vous pouvez lire l'article de vulgarisation https://automaths.blog/2017/06/20/la-conjecture-de-syracuse/amp/

a. Dès que $u_i = 1$ pour un certain i, la suite devient périodique de valeurs 4, 2, 1

Correction

On écrit d'abord une fonction qui prend en entrée la valeur de u_1 et renvoie la suite de Syracuse. Vue que la division par 2 se fait uniquement sur des nombres pairs, la suite reste une suite d'entiers. Pour garder cette propriété, il faut utiliser la division entière :

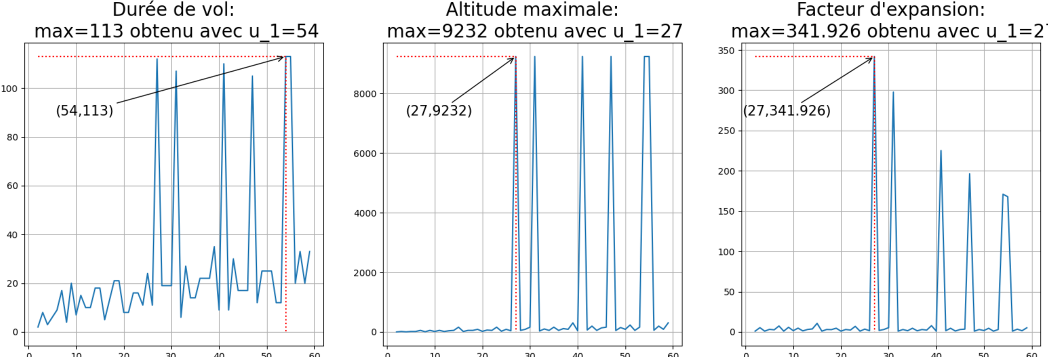
```
def suite(n):
	u = [n]
	while u[-1] != 1 :
		u.append( u[-1]//2 if u[-1]%2==0 else 3*u[-1]+1 )
	return u
```

On teste cette fonction :

```
N=60
Uinit=list(range(2,N))
```

```
L,M,F=[],[],[]
for n in Uinit:
    U=suite(n)
    L.append(len(U))
    M.append(max(U))
    F.append(M[-1]/n)
    #print(f"Avec u\_1={n}\tDurée de vol={L[-1]:3d}\t altitude maximale={M[-1]}\t facteur
     ↪  d'expansion={F[-1]}\n")


import matplotlib.pyplot as plt
plt.figure(figsize=(20,6))

ax1=plt.subplot(1,3,1)
ax1.plot(Uinit,L)
maxL=max(L)
indicemaxL=L.index(maxL)+2
ax1.plot([Uinit[0],indicemaxL,indicemaxL],[maxL,maxL,0],'r:')
ax1.set_title(f"Durée de vol:\n max={maxL} obtenu avec u_1={indicemaxL}",size=20)
ax1.annotate(f'({indicemaxL},{maxL})',
        xy=(indicemaxL, maxL), xycoords='data' ,
        xytext=(0.3, 0.75), textcoords='axes fraction', size=15,
        arrowprops=dict(arrowstyle="->"),
        horizontalalignment='right',
        verticalalignment='bottom'
        )
ax1.grid()

ax2=plt.subplot(1,3,2)
ax2.plot(Uinit,M)
maxM=max(M)
indicemaxM=M.index(maxM)+2
ax2.plot([Uinit[0],indicemaxM,indicemaxM],[maxM,maxM,0],'r:')
ax2.set_title(f"Altitude maximale:\n max={maxM} obtenu avec u_1={indicemaxM}",size=20)
ax2.annotate(f'({indicemaxM},{maxM})',
        xy=(indicemaxM, maxM), xycoords='data' ,
        xytext=(0.3, 0.75), textcoords='axes fraction', size=15,
        arrowprops=dict(arrowstyle="->"),
        horizontalalignment='right',
        verticalalignment='bottom'
        )
ax2.grid()

ax3=plt.subplot(1,3,3)
ax3.plot(Uinit,F)
maxF=max(F)
indicemaxF=F.index(maxF)+2
ax3.plot([Uinit[0],indicemaxF,indicemaxF],[maxF,maxF,0],'r:')
ax3.set_title(f"Facteur d'expansion:\n max={maxF:g} obtenu avec u_1={indicemaxF}",size=20)
ax3.annotate(f'({indicemaxF},{maxF:g})',
        xy=(indicemaxF, maxF), xycoords='data' ,
        xytext=(0.3, 0.75), textcoords='axes fraction', size=15,
        arrowprops=dict(arrowstyle="->"),
        horizontalalignment='right',
        verticalalignment='bottom'
        )
ax3.grid()
```

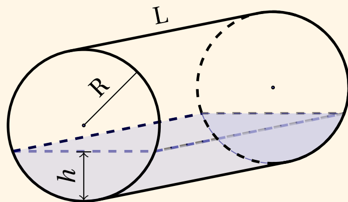
```
plt.show()
```



☆ Exercice Bonus 8.19 (Cuve de fioul enterrée)

J'ai acheté une ancienne maison qui utilise du mazout pour le chauffage. Le constructeur avait enterré une cuve cylindrique dont je ne connais pas la capacité, je peux juste mesurer la hauteur du mazout dans la cuve et le rayon qui est de 0.80 m. Sachant qu'au départ la hauteur du mazout dans la cuve est de $h_1 = 0.36$ m et qu'après avoir ajouté 3000 L la hauteur est de $h_2 = 1.35$ m,

1. calculer le volume total de la cuve
2. tracer la fonction qui renvoie les litres que je peux ajouter en fonction de la hauteur de mazout présent dans la cuve.



Correction

1. **Calcul de la capacité de la cuve.**

   Notons L la longueur de la cuve, R son rayon et h la hauteur du mazout. On a $0 \le h \le 2R$. Considérons une coupe verticale de la cuve comme à la Figure 8.1a.

   L'introduction de 3000 L, c'est-à-dire 3 m^3, de mazout fait passer h de h_1 à h_2. Ce volume correspond à $L \times \mathscr{A}$ où $\mathscr{A}$ est la surface coloriée de la Figure 8.1b.

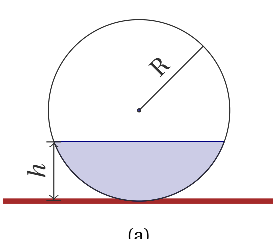
   Pour calculer l'aire coloriée, on "tourne" le cercle et on le plonge dans un repère orthonormé comme à la Figure 8.1c. On peut alors calculer l'aire grâce au calcul d'une intégrale :

   $$\mathscr{A} = 2\int_{h_1}^{h_2} \sqrt{x(2R-x)}\,dx.$$

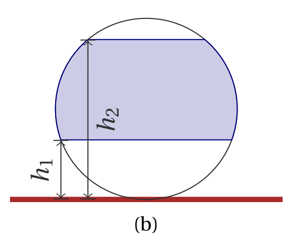
   Comme $L \times \mathscr{A} = 3$ m^3, on trouve ensuite la longueur L de la cuve (en mètres) et le volume totale de la cuve est $\pi R^2 L$ (en mètres cubes), c'est-à-dire $10^3\pi R^2 L$ (en litres).

   Le calcul analytique est possible (voir à la fin de cette correction) mais nous pouvons nous appuyer sur un calcul approché :

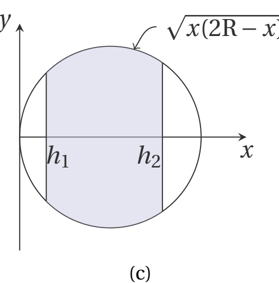
```
from numpy import sqrt, pi
from scipy import integrate

R  = 0.8  # en mètres
h1 = 0.36 # en mètres
h2 = 1.35 # en mètres
```



(a)



(b)



(c)

FIGURE 8.1. – Exercice 8.19

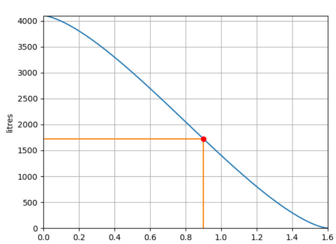
```
g = lambda x : sqrt(x*(2*R-x))

A_m2 = 2*integrate.quad(g,h1,h2)[0]   # en mètres carrés
L_m  = 3/A_m2                         # en mètres
V_m3 = pi*R**2*L_m                    # en mètres cubes
V_l  = V_m3*1.e3                      # en litres

print(f'A = {A_m2:g} m^2')
print(f'L = {L_m:g} m')
print(f'V = {V_m3:g} m^3 = {V_l:g} litres')
```

```
A = 1.47136 m^2
L = 2.03893 m
V = 4.09952 m^3 = 4099.52 litres
```

2. **Affichage de la fonction : litres à ajouter en fonction de la hauteur de mazout déjà présent dans la cuve.** Soit h la hauteur de mazout dans la cuve (en mètres) et ℓ les litres qu'on peut ajouter pour remplir la cuve, alors :

$$f\colon [0;2R] \to [0;4000]$$
$$h \mapsto 10^3 \times \left(V_{m^3} - 2\times L_m \times \int_0^h \sqrt{x(2R-x)}\,dx\right)$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate

f = lambda h : 1.e3*(V_m3-2*L_m*integrate.quad(g,0,h)[0])

hh = np.linspace(0,2*R,101)
yy = [f(h) for h in hh]
plt.plot(hh,yy)

htest = 0.9
ftest = f(htest)
plt.plot([htest,htest,0],[0,ftest,ftest])
plt.plot([htest],[ftest],lw=0.5,marker='o',color='red')
plt.xlabel("mètres")
plt.ylabel("litres")
plt.grid()
plt.axis([0,2*R,0,V_l])
plt.show()
```



A = 1.47136 m^2 L = 2.03893 m V = 4.09952 m^3 = 4099.52 litres

3. **Calcul analytique** Calculons tous d'abord les primitives de $\sqrt{x(2\mathrm{R}-x)}$:

$$\begin{aligned}\int\sqrt{x(2\mathrm{R}-x)}\,\mathrm{d}x &\overset{\substack{x=\mathrm{R}-y\\ \mathrm{d}x=-\mathrm{d}y}}{=} -\int\sqrt{\mathrm{R}^2-y^2}\,\mathrm{d}y \overset{\substack{y=\mathrm{R}\sin(t)\\ \mathrm{d}y=\mathrm{R}\cos(t)\,\mathrm{d}t}}{=} -\mathrm{R}^2\int\cos^2(t)\,\mathrm{d}t\\
&= -\frac{\mathrm{R}^2}{2}\left(\sin(t)\cos(t)+t\right)+c = -\frac{\mathrm{R}^2}{2}\left(\sin(t)\sqrt{1-\sin^2(t)}+t\right)+c\\
&\overset{t=\arcsin\left(\frac{y}{\mathrm{R}}\right)}{=} -\frac{\mathrm{R}^2}{2}\left(\frac{y}{\mathrm{R}}\sqrt{1-\frac{y^2}{\mathrm{R}^2}}+\arcsin\left(\frac{y}{\mathrm{R}}\right)\right)+c\\
&\overset{y=\mathrm{R}-x}{=} -\frac{\mathrm{R}-x}{2}\sqrt{x(2\mathrm{R}-x)}-\frac{\mathrm{R}^2}{2}\arcsin\left(1-\frac{x}{\mathrm{R}}\right)+c.\end{aligned}$$

Par conséquent

$$\mathscr{A} = 2\int_{h_1}^{h_2}\sqrt{x(2\mathrm{R}-x)}\,\mathrm{d}x \approx 1{,}48\,\mathrm{m}^2.$$

Comme $\mathrm{L}\times\mathscr{A} = 3\,\mathrm{m}^3$, on trouve ensuite $\mathrm{L}\approx 2\,\mathrm{m}$ et le volume totale de la cuve est $\pi\mathrm{R}^2\mathrm{L}\approx 4\,\mathrm{m}^3$, *i.e.* environs 4 000 L. Enfin

$$\begin{aligned}f(h) &= 10^3\times\left(\mathrm{V}_{m^3}-2\times\mathrm{L}_m\times\int_0^h\sqrt{x(2\mathrm{R}-x)}\,\mathrm{d}x\right)\\
&= 10^3\times\left(\mathrm{V}_{m^3}-2\times\mathrm{L}_m\times\left(-\frac{\mathrm{R}-h}{2}\sqrt{h(2\mathrm{R}-h)}-\frac{\mathrm{R}^2}{2}\arcsin\left(1-\frac{h}{\mathrm{R}}\right)+\frac{\mathrm{R}^2}{2}\arcsin(1)\right)\right)\\
&= 10^3\times\left(\mathrm{V}_{m^3}+\mathrm{L}_m\left((\mathrm{R}-h)\sqrt{h(2\mathrm{R}-h)}+\mathrm{R}^2\arcsin\left(1-\frac{h}{\mathrm{R}}\right)-\frac{\pi\mathrm{R}^2}{2}\right)\right)\end{aligned}$$

- $f(0)=\mathrm{V}_\ell$, $f(2\mathrm{R})=0$;
- $f'(h)=-2\times10^3\times\mathrm{L}_m\sqrt{h(2\mathrm{R}-h)}$;
- $f'(h)=0$ ssi $h=0$ ou $h=2\mathrm{R}$;
- f est décroissante pour tout $h\in[0;2\mathrm{R}]$;
- $f''(h)=-2\times10^3\times\mathrm{L}_m\frac{h-\mathrm{R}}{\sqrt{h(2\mathrm{R}-h)}}$;
- $h=\mathrm{R}$ est un point d'inflexion ;
- f convexe pour $h\in[\mathrm{R};2\mathrm{R}]$, concave pour $h\in[0;\mathrm{R}]$.

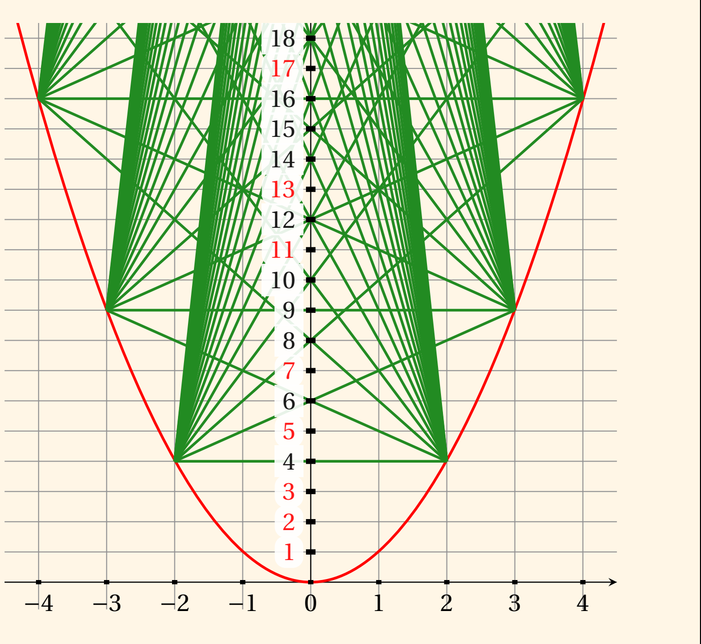
☆ Exercice Bonus 8.20 (Le crible de MATIYASEVITCH)

Traçons la parabole d'équation $y = x^2$. Sur ce graphe, on va considérer deux familles de points :

- pour tout $i \geq 2$, i entier, on note A_i le point de coordonnées $(-i, i^2)$,
- pour tout $j \geq 2$, j entier, on note B_j le point de coordonnées (j, j^2).

1. Relions tous les points A_i à tous les points B_j et tracer la figure ainsi obtenue avec `matplotlib`.
2. On constate que tous les segments $[A_iB_j]$ croisent l'axe des ordonnées en un point de coordonnées $(0, n)$ avec $n \in \mathbb{N}^*$. Démontrez-le mathématiquement.
3. Montrer qu'un nombre situé sur l'axe des ordonnées n'est pas premier si, et seulement si, un des segments $[A_iB_j]$ traverse l'axe des ordonnées en ce point.
4. Que représente le nombre de segments qui passent par les points de coordonnées $(0, n)$ avec $n \in \mathbb{N}^*$ et n non premier?

Rappel : un nombre $n \in \mathbb{N}^*$ est premier s'il n'est divisible que par 1 et lui même.



Correction

1.
```python
import matplotlib.pyplot as plt
import numpy as np
# la parabole
xx = np.linspace(-5,5,101)
yy = xx**2
plt.plot(xx,yy)
# les segments
A=[[-i,i**2] for i in range(1,10)]
B=[[ j,j**2] for j in range(1,10)]
for a in A:
	for b in B:
		plot([a[0],b[0]],[a[1],b[1]],'r-')
# fixons les axes pour un meilleur affichage
plt.axis([-5,5,0,19])
plt.xticks(range(-5,5,1), size='small')
plt.yticks(range(0,19,1), size='small')
plt.grid()
plt.show()
```

2. Le segment $[A_iB_j]$ appartient à la droite d'équation

$$y = \frac{j^2 - i^2}{j + i}(x - j) + j^2 = (j - i)x + ij$$

et il croise l'axe des ordonnées en le point de coordonnées $(0, ij)$. Comme $i, j \in \mathbb{N}^* \setminus \{1\}$, le produit ij appartient à $\mathbb{N}$.

3. Un nombre $n \in \mathbb{N}^*$ situé sur l'axe des ordonnées n'est pas premier si, et seulement si, il existe un couple $(i, j) \in \mathbb{N}^* \setminus \{1\}$ tel que $n = ij$ donc si, et seulement si, il existe un couple $(i, j) \in \mathbb{N}^* \setminus \{1\}$ tel que le segment $[A_iB_j]$ traverse l'axe des ordonnées en ce point.

4. Le nombre de segments qui passent par les points de coordonnées $(0, n)$ avec $n \in \mathbb{N}^*$ et n non premier représente le double du nombre de diviseurs propres de n si n n'est pas un carré parfait, sinon c'est le nombre de diviseurs propres de n.

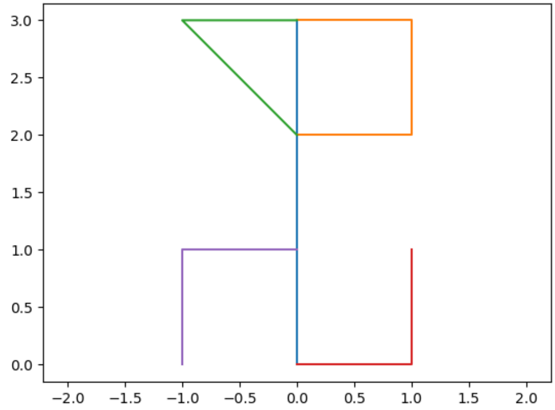
Exercice 8.21 (Notation cistercienne)
Le système de comptage numérique cistercien utilisé par l'ordre monastique cistercien à la fin de la période médiévale. Les chiffres de 1 à 9 sont représentés par des symboles disposés autour d'une portée verticale (le 0). En plaçant ces symboles réfléchis verticalement et/ou horizontalement à chacun des quatre emplacements, les chiffres décimaux dans les positions unités, dizaines, centaines et milliers pouvaient être représentés, permettant ainsi de définir les nombres de 0 à 9999.

cf. https://en.wikipedia.org/wiki/Cistercian_numerals

Écrire une fonction qui, pour un entier compris entre 0 et 9999, trace sa représentation cistercienne.

Correction
A chaque chiffre on associe un liste de deux tuples : le premier tuple contient les abscisses, le deuxième les ordonnées des points à relier.

```
import matplotlib.pyplot as plt

def arabic2cistercian(n):
    base = [(0,0),(0,3)]

    units = { "1": [(0,1),(3,3)] ,
              "2": [(0,1),(2,2)] ,
              "3": [(0,1),(3,2)] ,
              "4": [(0,1),(2,3)] ,
              "5": [(0,1,0),(3,3,2)] ,
              "6": [(1,1),(3,2)] ,
              "7": [(0,1,1),(3,3,2)] ,
              "8": [(0,1,1),(2,2,3)] ,
              "9": [(0,1,1,0),(2,2,3,3)]
              }

    tens = { cle: [ tuple( -x for x in val[0]), val[1] ] for cle, val in units.items() }
    hundreds = { cle: [ val[0], tuple( 3-y for y in val[1]) ] for cle, val in units.items() }
    thousands = { cle: [ tuple( -x for x in val[0]), tuple( 3-y for y in val[1]) ] for cle,
     ↪  val in units.items() }

    plt.axis('equal')
    s = str(n)[::-1]
    l = len(s)
    plt.plot( *base )
    if int(s[0])>0:
        plt.plot( *units[s[0]] )
    if l>1 and int(s[1])>0:
        plt.plot( *tens[s[1]] )
    if l>2 and int(s[2])>0:
        plt.plot( *hundreds[s[2]] )
    if l>3 and int(s[3])>0:
        plt.plot( *thousands[s[3]] )
    plt.show()

arabic2cistercian(8759)
```



☆ Exercice Bonus 8.22 (Taux Marginal, Taux Effectif : comprendre les impôts)

Dans l'imaginaire populaire, changer de tranche de revenu est vécu comme un drame. Combien de fois a-t-on entendu parents ou proches s'inquiéter de savoir si telle ou telle augmentation de revenu n'allait pas les faire changer de tranche et donc payer soudain plus d'impôts? Le plus simple, pour comprendre ce qu'il se passe, est de tracer la courbe des impôts en fonction du revenu ou plutôt du "quotient familial" (QF), parce qu'un même revenu n'est pas imposé de la même façon selon le nombre de personnes (parts) qu'il est censé faire vivre. On appelle QF le quotient du revenu par le nombre de parts et le nombre de parts est de 1 par adulte et de 0.5 par enfant, sauf cas particuliers.

Le principe de l'impôt sur les revenus est le suivant : le revenu imposable [a] pour l'année 2022 est partagé en tranches et l'impôt à payer en 2023 est calculé en prenant un certain pourcentage de chaque tranche. Pour les revenus de 2022 (impôts 2023), les tranches de revenus et les taux d'imposition correspondants, selon les données officielles prises sur le site du ministère des finances, sont les suivants : [b]

Tranche Du Revenu 2022	Taux d'imposition 2023
• jusqu'à 10777 € :	0%
• de 10778 € à 27478 € :	11%
• de 27479 € à 78570 € :	30%
• de 78571 € à 161994 € :	41%
• plus de 161995 € :	45%

Dans ce tableau, ce qu'on nomme le "taux d'imposition" est en fait le taux marginal : c'est un pourcentage qui **ne s'applique qu'à une partie des revenus**, celle de la tranche concernée. On voit que le taux marginal est plus important pour les forts revenus : c'est ce qu'on appelle la progressivité de l'impôt. On voit aussi que dans chaque tranche, le montant d'impôt à payer est proportionnel au QF : on dit que la fonction impôts est linéaire par morceaux.

Ces tranches d'imposition signifient la chose suivante :

- Si le revenu est inférieur à 10777 €, on ne paye pas d'impôt.
- Si le revenu est compris entre 10778 € et 27478 €, on ne paye pas d'impôt sur les premiers 10777 € de son revenu, et on paye 11% de la partie qui excède 10777 €. Par exemple, si le revenu est 10778 €, on payera 11% de 1 €, c'est-à-dire 11 centimes.
- Si le revenu est compris entre 27479 € et 78570 €, on ne paye rien sur la première tranche de 10777 €, puis 11% sur la deuxième tranche, allant de 10778 € à 27478 € (soit 11% de $(27478-10778)=(27478-10778)\times 0.11=1837$ €), et enfin 30% sur la partie du revenu qui excède 27478 €. Par exemple : si le revenu est de 72000 €, l'impôt sera de $0+1837.0+30\%\times(72000-27478)=13356.60$ €.

Prenons l'exemple concret d'un contribuable, célibataire, qui en 2023 déclare 100000 € de revenu pour l'année 2022. On veut calculer explicitement les impôts à payer. Ce contribuable est dans une tranche d'imposition marginale de 41% donc au total a un montant d'impôts de

$$0\times(10777-0)+11\%\times(27478-10777)+30\%\times(78570-27478)+41\%\times(100000-78570)=25951\text{ €}$$

et est ainsi redevable de 25.951% de ses revenus (et non 41%).

 © G. Faccanoni

Écrire et tracer le graphe des fonctions suivantes en fonction du revenu imposable (ou du QF) :

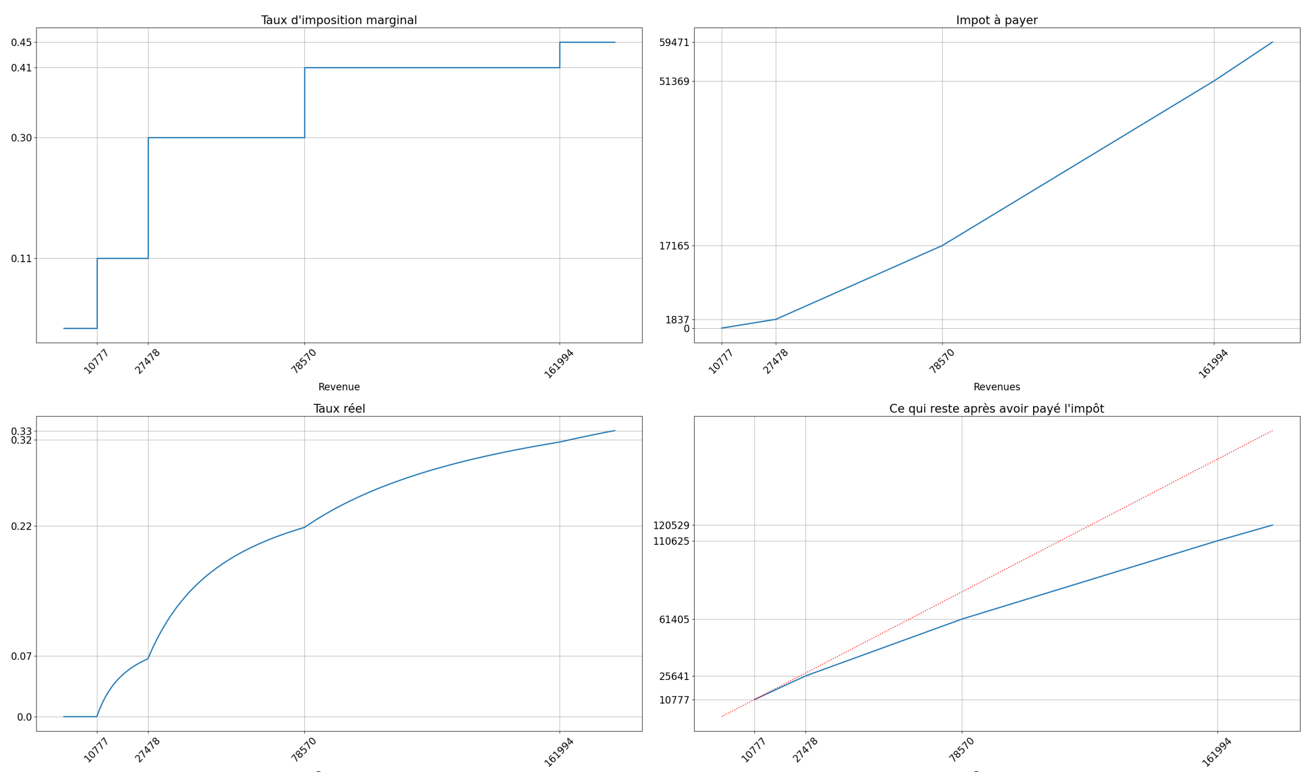
1. Taux d'imposition marginal,
2. Montant de l'impôt,
3. Taux réel d'imposition,
4. Ce qui reste après avoir payé l'impôt.

Vérifier notamment l'exemple donné ici https://www.service-public.fr/particuliers/vosdroits/F1419 : si un célibataire déclare 30000 €, le montant de l'impôt est 2583.71 €.

En traçant la courbe des impôts payés en fonction du QF on verra tout de suite qu'il n'y a aucun «saut» dans la courbe lorsqu'on change de tranche. Mathématiquement parlant, l'impôt est une *fonction continue* du QF pour qu'il n'y ait pas d'injustices : une petite modification du revenu n'entraîne qu'une petite modification de l'impôt (sauter d'une tranche n'est pas un drame). Avec des taux marginaux par tranches, on obtient une fonction *croissante* (plus le revenu est élevé et plus l'impôt est élevé) et affine par morceaux; et puisque le taux marginal augmente en fonction du revenu il s'agit d'une fonction *convexe*. Cela signifie que la courbe «monte toujours plus vite» : plus le revenu est élevé et plus le taux marginal augmente. Non seulement l'impôt augmente en fonction du revenu mais il augmente de plus en plus vite.

a. Il s'agit du revenu du contribuable auquel on retire certaines sommes (par exemple au titre des frais professionnels).
b. https://www.service-public.fr/particuliers/vosdroits/F1419

Correction
Les graphes demandés sont les suivants :



On commence par comprendre mathématiquement comment les obtenir, puis on écrira le code.

1. Commençons par tracer le graphique responsable de la crainte du saut d'une tranche : en horizontale le revenu imposable, en verticale le taux de la tranche correspondante (appelé *taux marginal)*. Ce graphique présente effectivement des sauts importants. Mais *ces sauts ne concernent pas l'impôt mais la dérivée de la fonction Impôt* I *par rapport au revenu* R. La fonction Impôt $\mathrm{I}\colon \mathbb{R}_+ \to \mathbb{R}_+$ est une fonction continue mais pas sa dérivée $\mathrm{I}'\colon \mathbb{R}_+ \to \mathbb{R}_+$

qui est constante par morceaux et définie par

$$I'(R) = \begin{cases} 0 & \text{si } R \le 10777\,€ \\ 0.11 & \text{si } 10777\,€ < R \le 27478\,€ \\ 0.30 & \text{si } 27478\,€ < R \le 78570\,€ \\ 0.41 & \text{si } 78570\,€ < R \le 161994\,€ \\ 0.45 & \text{si } R > 161994\,€ \end{cases}$$

Ce graphique indique le pourcentage d'imposition qui serait appliqué *sur chaque euro supplémentaire* qui viendrait s'ajouter au revenu.

2. La fonction Impôt $I\colon \mathbb{R}_+ \to \mathbb{R}_+$ est bien continue et définie par

$$\begin{aligned} I(R) &= \int_0^R I'(r)\,dr \\ &= \begin{cases} \int_0^R 0\,dr & \text{si } R \le 10777\,€ \\ \int_0^{10777} 0\,dr + \int_{10777}^R 0.11\,dr & \text{si } 10777\,€ < R \le 27478\,€ \\ \int_0^{10777} 0\,dr + \int_{10777}^{27478} 0.11\,dr + \int_{27478}^R 0.30\,dr & \text{si } 27478\,€ < R \le 78570\,€ \\ \int_0^{10777} 0\,dr + \int_{10777}^{27478} 0.11\,dr + \int_{27478}^{78570} 0.30\,dr + \int_{78570}^R 0.41\,dr & \text{si } 78570\,€ < R \le 161994\,€ \\ \int_0^{10777} 0\,dr + \int_{10777}^{27478} 0.11\,dr + \int_{27478}^{78570} 0.30\,dr + \int_{78570}^{161994} 0.41\,dr + \int_{161994}^R 0.45\,dr & \text{si } R > 161994\,€ \end{cases} \\ &= \begin{cases} 0 & \text{si } R \le 10777\,€ \\ 0 + 0.11(R-10777) & \text{si } 10777\,€ < R \le 27478\,€ \\ 0 + 0.11(27478-10777) + 0.30(R-27478) & \text{si } 27478\,€ < R \le 78570\,€ \\ 0 + 0.11(27478-10777) + 0.30(78570-27478) + 0.41(R-78570) & \text{si } 78570\,€ < R \le 161994\,€ \\ 0 + 0.11(27478-10777) + 0.30(78570-27478) + 0.41(161994-78570) + 0.45(R-161994) & \text{si } R > 161994\,€ \end{cases} \end{aligned}$$

Elle est bien une fonction continue du revenu, croissante, affine par morceaux et convexe.

3. Le taux d'imposition que l'on retient en général est celui de la tranche de revenu dans laquelle on est : le célibataire qui en 2023 déclare 100000 € de revenu pour l'année 2022 se trouve dans la tranche à 41%. Pourtant, il s'agit là d'un taux marginal et non pas de l'impôt réellement payé. Ce qui compte, c'est plutôt ce qu'on paye vraiment à la fin, au total, sur l'ensemble de son revenu ou QF. Le taux réel d'imposition (ou taux effectif) désigne le pourcentage d'impôts réellement payé. Pour calculer le Taux réel d'imposition, c'est-à-dire le pourcentage qu'il faut appliquer au revenu pour avoir l'impôt, il s'agit simplement de diviser I par R. Par exemple, si le revenu est de 50000 €, l'impôt est de 8593.71 €, si bien que le taux global est de $8593.71/50000 = 0.172$, c'est-à-dire $\simeq 17.2\%$.
   Il s'agit de la fonction $G\colon \mathbb{R}_+ \to \mathbb{R}_+$ définie par $G(R) = \frac{I(R)}{R}$. Chaque changement de tranche se repère ici par un point anguleux dans la courbure de la fonction.

4. Ce dernier graphique montre une autre propriété qui mérite d'être signalée car elle n'est pas évidente pour tout le monde. Horizontalement, toujours le revenu. Verticalement, on indique «ce qui reste quand on a payé les impôts», autrement dit la différence entre le revenu et l'impôt. Eh bien, cette fonction est encore croissante. Qu'est-ce que cela signifie? Tout simplement que plus on gagne et plus on est riche, même après avoir déduit les impôts.
   Il s'agit de la fonction $A\colon \mathbb{R}_+ \to \mathbb{R}_+$ définie par $A(R) = R - I(R)$.

```python
import matplotlib.pyplot as plt
import numpy as np

# tranche = [10084,25710,73516,158122] #2020
# tranche = [10225,26070,74545,160336] #2021
tranche = [10777, 27478, 78570, 161994]  # 2022
taux = [0.11, 0.30, 0.41, 0.45]


def Impot(R):
    if R <= tranche[0]:
        return 0
    elif R <= tranche[1]:
        return 0 + taux[0] * (R - tranche[0])
    elif R <= tranche[2]:
        return 0 + taux[0] * (tranche[1] - tranche[0]) + taux[1] * (R - tranche[1])
```

```
    elif R <= tranche[3]:
        return (
            0
            + taux[0] * (tranche[1] - tranche[0])
            + taux[1] * (tranche[2] - tranche[1])
            + taux[2] * (R - tranche[2])
        )
    else:
        return (
            0
            + taux[0] * (tranche[1] - tranche[0])
            + taux[1] * (tranche[2] - tranche[1])
            + taux[2] * (tranche[3] - tranche[2])
            + taux[3] * (R - tranche[3])
        )


Taux_Reel = lambda R: Impot(R) / R if R > 0 else 0

Reste = lambda R: R - Impot(R)


# ==================================================================
# TEST
# ==================================================================
R = 30000
print(f"""Pour un célibataire dont le revenu net imposable est de {R} €, sans aucune réduction
↪ ni déduction,
son impôt brut est de {Impot(R):.2f},
le taux réel d'imposition est {Taux_Reel(R)*100:.2f}%
Il lui reste {Reste(R):.2f} €. """)

# ==================================================================
# COURBES
# ==================================================================
fig = plt.figure(figsize=(30, 18))
plt.rcParams["font.size"] = "16"

rr = tranche + [180000]

yy_impot = [Impot(r) for r in rr]
yy_reste = [Reste(r) for r in rr]
yy_taux_reel = [round(Taux_Reel(r),2) for r in rr]


xx_ticks = [
    0,
    tranche[0],
    tranche[0],
    tranche[1],
    tranche[1],
    tranche[2],
    tranche[2],
    tranche[3],
    tranche[3],
    180000,
]

# ==================================================================
```

```
plt.subplot(2, 2, 1)
plt.plot(
    xx_ticks,
    [0, 0, taux[0], taux[0], taux[1], taux[1], taux[2], taux[2], taux[3], taux[3]],
    lw=2,
)
plt.xticks(tranche, rotation=45)
plt.yticks(taux)
plt.xlabel("Revenue")
plt.title("Taux d'imposition marginal")
plt.grid(True)

# ====================================================================
plt.subplot(2, 2, 2)
plt.plot(rr, yy_impot, lw=2)
plt.xticks(tranche, rotation=45)
plt.yticks(yy_impot)
plt.xlabel("Revenues")
plt.title("Impot à payer")
plt.grid(True)

# ====================================================================
plt.subplot(2, 2, 3)

rr_1 = np.linspace(0, 180000, 1000)
yy_1 = [Taux_Reel(r) for r in rr_1]
plt.plot(rr_1,yy_1,lw=2)
plt.xticks(tranche, rotation=45)
plt.yticks(yy_taux_reel,yy_taux_reel)
plt.xlabel("Revenues")
plt.title("Taux réel")
plt.grid(True)

# ====================================================================
plt.subplot(2, 2, 4)
plt.plot(rr, yy_reste, lw=2)
plt.plot([0, 180000], [0, 180000], "r:")
plt.xticks(tranche, rotation=45)
plt.yticks(yy_reste)
plt.xlabel("Revenues")
plt.title("Ce qui reste après avoir payé l'impôt")
plt.grid(True)

# ====================================================================
plt.tight_layout()
fig.set_dpi(50)
plt.savefig("Images/global.png")
plt.show();
```

```
Pour un célibataire dont le revenu net imposable est de 30000 €, sans aucune réduction ni
↪ déduction,
son impôt brut est de 2593.71,
le taux réel d'imposition est 8.65%
Il lui reste 27406.29 €.
```

ANNEXE A

Les «mauvaises» propriétés des nombres flottants et la notion de précision

- Calcul numérique : calcul en utilisant des nombres, en général en virgule flottante.
- Calcul numérique $\neq$ calcul symbolique.
- Nombre flottant : signe + mantisse («chiffres significatifs») + exposant.
- Valeur d'un flottant = $(-1)^{\text{signe}} + 1.\text{mantisse} * 2^{\text{exposant}}$
- Précision avec les flottants Python (double précision = 64 bits) :
  - 1 bit de signe
  - 52 bits de mantisse ($\Rightarrow$ environ 15 à 16 décimales significatives)
  - 11 bits d'exposant ($\Rightarrow$ représentation des nombres de 10^{-308} à 10^{308})

Exemple (Un calcul de π)
On veut utiliser la propriété mathématique

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

pour calculer la valeur de π. Comme ce calcul est réalisé avec des nombres flottants, il y a plusieurs problèmes de précision :

1. le meilleur calcul de π possible ne pourra donner qu'un arrondi à $\approx 10^{-15}$ près tandis que la vraie valeur de π n'est pas un flottant représentable (il n'est même pas rationnel) ;
2. en utilisant des nombres flottants, chaque opération ($/$, $+$, $\ldots$) peut faire une erreur d'arrondi (erreur relative de 10^{-15}). Les erreurs peuvent se cumuler.
3. La formule mathématique est infinie. Le calcul informatique sera forcé de s'arrêter après un nombre fini d'itérations.

```
print(4*sum([(-1)**n/(2*n+1) for n in range(1)]))
print(4*sum([(-1)**n/(2*n+1) for n in range(10)]))
print(4*sum([(-1)**n/(2*n+1) for n in range(100)]))
print(4*sum([(-1)**n/(2*n+1) for n in range(1000)]))
```

```
4.0
3.0418396189294032
3.1315929035585537
3.140592653839794
```

Valeur prédéfinie par le module math

```
from math import *
print(pi)
```

```
3.141592653589793
```

A.1. Ne jamais faire confiance aveuglément aux résultats d'un calcul obtenu avec un ordinateur...

L'expérimentation numérique dans les sciences est un sujet passionnant et un outil incontournable pour de nombreux scientifiques.

Malgré la puissance de calcul vertigineuse de nos ordinateurs modernes, et encore plus de certains centres de calculs, il ne faut pas oublier complètement la théorie et prendre à la légère le fonctionnement des machines, sous peine d'avoir quelques surprises...

Observons des calculs quelque peu surprenants.

- Dans le calcul suivant, la différence entre 4.9 et 4.845 devrait être 0.055 mais python nous dit que non :
```
>>> 4.9 - 4.845 == 0.055
False
```
Pourquoi cela se produit-il ? Si nous regardons $4.9 - 4.845$, nous pouvons voir que nous obtenons en fait
```
>>> 4.9-4.845
0.055000000000000604
```
- Un autre exemple est le suivant qui montre que $0.1 + 0.2 + 0.3 \neq 0.6$:
```
>>> 0.1 + 0.2 + 0.3
0.6000000000000001
>>> 0.1 + 0.2 + 0.3 == 0.6
False
>>> round(0.1 + 0.2 + 0.3, 5)  == round(0.6, 5)
True
```
-
```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.551115123125783e-17
>>> 1.1 + 2.2
3.3000000000000003
```

Que s'est-il passé ? Tout simplement, les calculs effectués ne sont pas exacts et sont entachés d'erreurs d'arrondis. En effet, tout nombre réel possède un développement décimal soit fini soit illimité. Parmi les nombres réels, on peut alors distinguer les rationnels (dont le développement décimal est soit fini soit illimité et périodique à partir d'un certain rang) des irrationnels (dont le développement décimal est illimité et non périodique). Il est aisé de concevoir qu'il n'est pas possible pour un ordinateur de représenter de manière exacte un développement décimal illimité, mais même la représentation des développements décimaux finis n'est pas toujours possible. En effet, un ordinateur stocke les nombres non pas en base 10 mais en base 2. Or, un nombre rationnel peut tout à fait posséder un développement décimal fini et un développement binaire illimité ! C'est le cas des décimaux 1.1 et 2.2 qui, en base 2, s'écrivent $01.\overline{01}$ et $10.\overline{001}$ respectivement.

Remarque

Un exemple fréquent d'arrondi se produit avec la simple instruction `t=0.1` La valeur mathématique t stockée dans `t` n'est pas exactement 0.1 car l'expression de la fraction décimale $1/10$ en binaire nécessite une série infinie. En effet,

$$\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \cdots$$

La séquence de coefficients 1, 1, 0, 0 se répète à l'infini. En regroupant les termes résultants quatre par quatre, on exprime $1/10$ comme

$$\frac{1}{10} = \frac{1}{2^4}\left(1 + \sum_{i=1}^{\infty} \frac{9}{16^i}\right).$$

Les nombres à virgule flottante de part et d'autre de $1/10$ sont obtenus en terminant la partie fractionnaire de cette série après 52 termes binaires ou 13 termes hexadécimaux et en arrondissant le dernier terme vers le haut ou vers le bas. Ainsi,

$$t_1 < t < t_2$$

où

$$t_1 = \frac{1}{2^4}\left(1 + \sum_{i=1}^{12} \frac{9}{16^i} + \frac{9}{16^{13}}\right),$$
$$t_2 = \frac{1}{2^4}\left(1 + \sum_{i=1}^{12} \frac{9}{16^i} + \frac{10}{16^{13}}\right).$$

 © G. Faccanoni

Il se trouve que 1/10 est plus proche de t_2 que t_1 ainsi `t`= t_2.

En résumé, la valeur stockée dans `t` est très proche de, mais pas exactement égale à, 0.1. La distinction est parfois importante. Par exemple, la quantité 0.3/0.1 n'est pas exactement égale à 3 car le numérateur réel est un peu inférieur à 0.3 et le dénominateur réel est un peu plus grand que 0.1 :

```
>>> 0.3/0.1
2.9999999999999996
```

Source : Cleve's Corner : Cleve Moler on Mathematics and Computing https://blogs.mathworks.com/cleve/2014/07/07/floating-point-numbers/

Voici quelque source d'erreurs :

Représentation décimale inexacte Dans l'exemple ci-dessous 1.2 n'est pas représentable en machine. L'ordinateur utilise «le flottant représentable le plus proche de 1.2»

```
>>> 1.2 - 1 - 0.2
-5.551115123125783e-17
```

Non-commutativité

```
>>> 1 + 1e-16 - 1
0.0
>>> -1 + 1e-16 + 1
1.1102230246251565e-16
```

Erreurs d'arrondis

```
>>> 1/3 - 1/4 - 1/12
-1.3877787807814457e-17
```

Conséquences

- On ne peut pas espérer de résultat exact
- La précision du calcul dépend de beaucoup d'éléments
- En général, pour éviter les pertes de précision, on essayera autant que faire se peut d'éviter :
  - de soustraire deux nombres très proches
  - d'additionner ou de soustraire deux nombres d'ordres de grandeur très différents. Ainsi, pour calculer une somme de termes ayant des ordres de grandeur très différents (par exemple dans le calcul des sommes partielles d'une série), on appliquera le principe dit "de la photo de classe" : les petits devant, les grands derrière.
- **tester** `x == flottant` **est presque toujours une erreur**, on utilisera plutôt `abs(x-flottant)<1.e-10` par exemple.
- "Si on s'y prend bien", on perd $\approx 10^{-16}$ en précision relative à chaque calcul $\Rightarrow$ acceptable par rapport à la précision des données.
- "Si on s'y prend mal", le résultat peut être complètement faux!

Illustrons ce problème d'arrondis en partant de l'identité suivante :

$$xy = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2.$$

Dans le programme suivant on compare les deux membres de cette égalité pour des nombres x et y de plus en plus grands :

```
prod = lambda x,y : x*y
diff = lambda x,y : ((x+y)/2)**2-((x-y)/2)**2

a = 6553.99
b = a+1

from tabulate import tabulate
T = []
T.append(["a","b","prod(a,b)-diff(a,b)"])
```

```
for i in range(6):
    produit = prod(a,b)
    difference = diff(a,b)
    T.append( [a,b,produit-difference] )
    a, b = produit, a+1

print(tabulate(T, headers="firstrow",floatfmt="1.10e"))
```

On constate que la divergence est spectaculaire :

```
               a                 b     prod(a,b)-diff(a,b)
----------------  ----------------  ----------------------
6.5539900000e+03  6.5549900000e+03        0.0000000000e+00
4.2961338910e+07  6.5549900000e+03       -5.8593750000e-02
2.8161114694e+11  4.2961339910e+07        1.6670720000e+06
1.2098392206e+19  2.8161114694e+11       -4.7982566402e+21
3.4070421054e+30  1.2098392206e+19        1.0466488703e+44
4.1219731654e+49  3.4070421054e+30        1.4043736132e+80
```

Le module `fractions` Pour corriger ce problème on peut utiliser le module `fractions` :

```
from fractions import Fraction
T = []
print(f'{"Float":*^20} vs {"Fraction":*^19}')
T.append([0.1 + 0.1 + 0.1 - 0.3, Fraction(1,10) + Fraction(1,10) + Fraction(1,10) -
    Fraction(3,10)])
T.append([1.1 + 2.2, Fraction(11,10) + Fraction(22,10)])
T.append([1.0 / 3 - 1.0 / 4 - 1.0 / 12, Fraction(1,3) + Fraction(1,4) - Fraction(1,12)])
T.append([ 1 + 1e-16 - 1, Fraction(1,1) + Fraction(1,10**16) - Fraction(1,1)])
T.append([-1 + 1e-16 + 1, -Fraction(1,1) + Fraction(1,10**16) + Fraction(1,1)])
T.append([1.2 - 1.0 - 0.2, Fraction(6,5) - Fraction(1,1) - Fraction(1,5)])
for t in T:
    print(f"{t[0]:<+.17f} vs {t[1]}")
```

```
*******Float******** vs *****Fraction******
+0.00000000000000006 vs 0
+3.30000000000000027 vs 33/10
-0.00000000000000001 vs 1/2
+0.00000000000000000 vs 1/10000000000000000
+0.00000000000000011 vs 1/10000000000000000
-0.00000000000000006 vs 0
```

Revenons sur notre exemple :

```
from fractions import Fraction
from tabulate import tabulate
T = []

prod = lambda x,y : x*y
diff = lambda x,y : Fraction(x+y,2)**2-Fraction(x-y,2)**2

a = Fraction(655399,100)
b = a+1

T.append(["a","b","prod(a,b)-diff(a,b)"])
for i in range(6):
    produit = prod(a,b)
    difference = diff(a,b)
    T.append( [a,b,produit-difference] )
    a, b = produit, a+1
```

```
print(tabulate(T, headers="firstrow",floatfmt="1.15e"))
```

```
                    a                      b    prod(a,b)-diff(a,b)
---------------------  ---------------------  ---------------------
6.553990000000000e+03  6.554990000000000e+03  0.000000000000000e+00
4.296133891010000e+07  6.554990000000000e+03  0.000000000000000e+00
2.816111469423164e+11  4.296133991010000e+07  0.000000000000000e+00
1.209839220626197e+19  2.816111469433164e+11  0.000000000000000e+00
3.407042105375514e+30  1.209839220626197e+19  0.000000000000000e+00
4.121973165408151e+49  3.407042105375514e+30  0.000000000000000e+00
```

Remarque

Voici deux exemples de désastres causés par une mauvaise gestion des erreurs d'arrondi :

- Le 25 février 1991, pendant la Guerre du Golfe, une batterie américaine de missiles Patriot située à Dharan en Arabie Saoudite a manqué l'interception d'un missile Scud irakien, qui a frappé un baraquement de l'armée américaine, causant la mort de 28 soldats. Une commission d'enquête a déterminé que l'échec était dû à un calcul incorrect du temps de parcours du missile Scud, causé par un problème d'arrondi lié à la représentation des nombres en virgule fixe sur 24 bits utilisée par l'ordinateur de bord du système Patriot. Le temps était compté en dixièmes de seconde par l'horloge interne du système, mais la représentation binaire de $1/10$ n'a pas d'écriture finie en binaire : $1/10 = 0.1$ (dans le système décimal) $= 0.00011001100110011001100110011\ldots$ (dans le système binaire). L'ordinateur de bord arrondissait $1/10$ à 24 chiffres, ce qui a entraîné une petite erreur dans le décompte du temps pour chaque dixième de seconde. Environ 100 heures s'étaient écoulées depuis l'allumage du système Patriot, ce qui avait entraîné une accumulation des erreurs d'arrondi de 0,34 s. Pendant ce temps, le missile Scud avait parcouru environ 500 m, expliquant ainsi pourquoi le Patriot avait manqué sa cible.
- Le 4 juin 1996, le lancement d'une fusée Ariane 5 a connu un accident et a explosé 40 secondes après son décollage. Le coût total de la fusée et de son chargement était estimé à 500 millions de dollars. Après deux semaines d'enquête, la commission d'enquête a conclu que l'incident était dû à une erreur de programmation dans le système inertiel de référence. Au cours de la conversion d'un nombre codé en virgule flottante sur 64 bits (représentant la vitesse horizontale de la fusée par rapport à la plate-forme de lancement), celui-ci a été converti en un entier sur 16 bits. Malheureusement, le nombre en question était supérieur à 32 768, la valeur maximale pouvant être codée sur 16 bits, ce qui a conduit à une erreur de conversion.

A.2. Exercices

Exercice A.1 (Devine le résultat)

Expliquer les résultats suivants :

```
>>> print(0.2+0.3+0.4)
0.9
>>> print(0.3+0.4+0.2)
0.8999999999999999
>>> print(0.4+0.2+0.3)
0.9000000000000001
>>> print(0.2+0.4)
0.6000000000000001
>>> print(  (-7.35e22 + 7.35e22) + 100.0  )
100.0
>>> print(  -7.35e22 + (7.35e22 + 100.0)  )
0.0
```

Correction

Python effectue les opérations de gauche à droite. Pour la première expression, il calcule déjà `0.2 + 0.3`, puis ajoute au résultat `0.4`. Pour la seconde opération, il calcule `0.3 + 0.4`, puis ajoute au résultat `0.2`. Or ces nombres n'ont pas une écriture finie en base 2 et sont donc approchés. Les erreurs sur le dernier chiffre provoquent ces différences.

Le module `fractions` permet de faire des calculs exacts sur les rationnels :

```
>>> from fractions import Fraction
>>> print(Fraction(2, 10) + Fraction(3, 10) + Fraction(4, 10))
9/10
```

Pour les deux derniers exemples, le problème est liée à la non associativité des calculs en virgule flottante.

Exercice A.2 (Qui est plus grand ?)

Parmi A et B, qui est plus grand si

$$\mathrm{A} = \frac{2^{2021}-1}{4^{2021}-2^{2021}+1}, \qquad \mathrm{B} = \frac{2^{2021}+1}{4^{2021}+2^{2021}+1}?$$

Correction

Naïvement on pourrait essayer ceci :

```
>>> A = (2**2021-1)/(4**2021-2**2021+1)
>>> B = (2**2021+1)/(4**2021+2**2021+1)
>>> print(A-B)
0.0
```

et en déduire que $\mathrm{A} = \mathrm{B}$, mais regardons un peu mieux :

```
>>> A = (2**2021-1)/(4**2021-2**2021+1)
>>> B = (2**2021+1)/(4**2021+2**2021+1)
>>> print(f"{A=}, {B=}")
A=0.0, B=0.0
```

Or, ceci ce n'est pas possible car $2^{2021} \pm 1 \neq 0$.

Posons $x = 2^{2021} > 0$ alors $\mathrm{A} = \dfrac{x-1}{x^2-x+1}$ et $\mathrm{B} = \dfrac{x+1}{x^2+x+1}$ ainsi

$$\mathrm{A}-\mathrm{B} = \frac{(x-1)(x^2+x+1)-(x+1)(x^2-x+1)}{(x^2-x+1)(x^2+x+1)} = \frac{(x^3+x^2+x-x^2-x-1)-(x^3-x^2+x+x^2-x+1)}{(x^2-x+1)(x^2+x+1)} = \frac{-2}{(x^2-x+1)(x^2+x+1)} < 0$$

Une autre stratégie de calcul est la suivante :

$$\frac{1}{A}-\frac{1}{B}=\frac{x(x-1)+1}{x-1}-\frac{x(x+1)+1}{x+1}=x+\frac{1}{x-1}-x-\frac{1}{x+1}=\frac{1}{x-1}-\frac{1}{x+1}>0.$$

Pour calculer la bonne valeur nous allons utiliser le module `fractions` qui évite les erreurs d'arrondis :

```
from fractions import Fraction
A = Fraction(2**2021-1,4**2021-2**2021+1)
B = Fraction(2**2021+1,4**2021+2**2021+1)
print(A-B)
```

On voit qu'il s'agit d'une fraction avec un numérateur négatif et un dénominateur positif, ainsi $A<B$.

Exercice A.3 (Integer VS Floating point number)

Expliquer les résultats suivants :

```
>>> int(123123123123123123123.0)
123123123123123126272
>>> int(123123123123123123123)
123123123123123123123

>>> int(123123123123123123123.0 % 10**9)
123126272
>>> int(123123123123123123123 % 1e9)
123126272
```

Correction
Dans Python, les nombres entiers ont une précision infinie tandis que les nombres floating point ont une précision finie.

Exercice A.4 (Un contre-exemple du dernier théorème de Fermat?)

Le dernier théorème de Fermat (prouvé par Andriew Wiles en 1994) affirme que, pour tout entier $n>2$, trois entiers positifs x, y et z ne peuvent pas satisfaire l'équation $x^n+y^n-z^n=0$. Expliquez le résultat de cette commande qui donne un contre-exemple apparent au théorème :

```
>>> 844487.**5 + 1288439.**5 - 1318202.**5
0.0
```

Correction
Source : https://scipython.com/book/chapter-9-general-scientific-programming/questions/

Le problème, bien sûr, vient de l'utilisation des nombres à virgule flottante double précision et que la différence entre la somme des deux premiers termes et celle du troisième est inférieure à la précision de cette représentation.

Si on utilise des entiers on a bien :

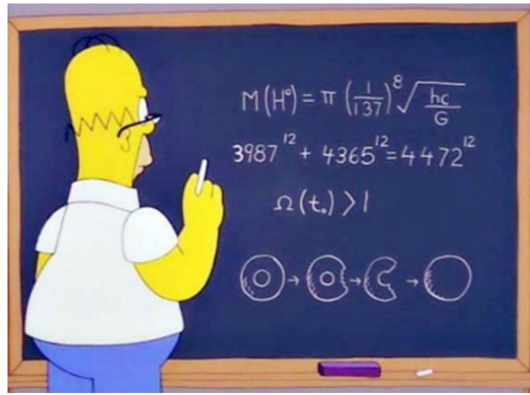
```
>>> 844487**5 + 1288439**5 - 1318202**5
-235305158626
```

En effet, la précision finie de la représentation en virgule flottante utilisée tronque les décimales avant que cette différence soit apparente :

```
>>> print("Exact =", 844487**5 + 1288439**5, "Arrondis =", 844487.**5 + 1288439.**5)
Exact = 3980245235185639013055619497406 Arrondis = 3.980245235185639e+30
>>> print("Exact =", 1318202**5, "Arrondis =", 1318202.**5)
Exact = 3980245235185639013290924656032 Arrondis = 3.980245235185639e+30
```

Ceci est un exemple d'annulation catastrophique.

Dans l'épisode 2 de la saison 10 des Simpson (1998), intitulé “La Dernière Invention d'Homer”, on peut voir le tableau ci-dessous :



La ligne $3987^{12}+4365^{12}=4472^{12}$ est un faux contre-exemple du théorème de Fermat. En effet,

$$\begin{aligned}3987^{12}+4365^{12} &= 63976656349698612616236230953154487896987106\\ 4472^{12} &= 63976656348486725806862358322168575784124416\end{aligned}$$

soit une différence de

$$12118868093738726309859121112862690.$$

Cependant, l'erreur relative

$$\frac{3987^{12}+4365^{12}-4472^{12}}{4472^{12}} = 1.894264062148878e-11$$

est suffisamment faible pour qu'une calculatrice standard considère ces deux termes comme égaux.

Exercice A.5 (Suites)

Calculer analytiquement et numériquement les premiers 100 termes des suites suivantes :

$$\begin{cases} u_0=\frac{1}{4},\\ u_{n+1}=5u_n-1,\end{cases}\qquad \begin{cases} v_0=\frac{1}{5},\\ v_{n+1}=6v_n-1,\end{cases}\qquad \begin{cases} w_0=\frac{1}{3},\\ w_{n+1}=4v_w-1.\end{cases}$$

Correction

Clairement $u_i=\frac{1}{4}$, $v_i=\frac{1}{5}$ et $w_i=\frac{1}{3}$ pour tout $i\in\mathbb{N}$. Cependant, lorsqu'on calcul les premiers 100 termes de ces deux suites avec Python (ou avec un autre langage de programmation) on a quelques surprises.

Si on écrit

```
n=0
u = 1/4
print(f"u_{n}={u}")
for n in range(1,30):
    u = 5*u-1
    print(f"u_{n}={u}")
```

on trouve bien $u_i=0.25$ pour tout $i=0,\dots$:

```
u_0=0.25     u_6=0.25     u_12=0.25     u_18=0.25     u_24=0.25
u_1=0.25     u_7=0.25     u_13=0.25     u_19=0.25     u_25=0.25
u_2=0.25     u_8=0.25     u_14=0.25     u_20=0.25     u_26=0.25
u_3=0.25     u_9=0.25     u_15=0.25     u_21=0.25     u_27=0.25
u_4=0.25     u_10=0.25    u_16=0.25     u_22=0.25     u_28=0.25
u_5=0.25     u_11=0.25    u_17=0.25     u_23=0.25     u_29=0.25
```

Mais si on écrit

```
n=0
v = 1/5
print(f"v_{n}={v}")
for n in range(1,30):
    v = 6*v-1
    print(f"v_{n}={v}")
```

on obtient $v_i \simeq 0.2$ pour $i = 0,\dots,5$, ensuite les erreurs d'arrondis commencent à se voir :

```
v_0=0.2
v_1=0.20000000000000018
v_2=0.20000000000000107
v_3=0.2000000000000064
v_4=0.20000000000003837
v_5=0.20000000000023022
v_6=0.2000000000013813
v_7=0.20000000000828777
v_8=0.20000000004972662
v_9=0.20000000029835974
v_10=0.20000000179015842
v_11=0.20000001074095053
v_12=0.20000006444570317
v_13=0.20000038667421904
v_14=0.20000232004531426
v_15=0.20001392027188558
v_16=0.2000835216313135
v_17=0.20050112978788093
v_18=0.20300677872728556
v_19=0.21804067236371338
v_20=0.3082440341822803
v_21=0.8494642050936818
v_22=4.096785230562091
v_23=23.580711383372545
v_24=140.48426830023527
v_25=841.9056098014116
v_26=5050.43365880847
v_27=30301.60195285082
v_28=181808.6117171049
v_29=1090850.6703026295
```

De même

```
n=0
w = 1/3
print(f"w_{n}={w}")
for n in range(1,41):
    w = 4*w-1
    print(f"w_{n}={w}")
```

À la vingtième répétition, le résultat est $w_{20} = 0.33331298828125$ ce qui est déjà assez éloigné de $1/3$. À la quarantième répétition de la ligne, le résultat est $w_{40} = -22369621.0$ ce qui n'a plus rien à voir. En fait, l'erreur sur l'arrondi se cumule et le résultat devient complètement absurde.

```
w_0=0.3333333333333333
w_1=0.33333333333333326
w_2=0.33333333333333304
w_3=0.33333333333333215
w_4=0.3333333333333286
w_5=0.3333333333333144
w_6=0.33333333333325754
w_7=0.33333333333303017
w_8=0.3333333333321207
w_9=0.3333333333284827
w_10=0.3333333333139308
w_11=0.3333333332557231
w_12=0.3333333330228925
w_13=0.3333333320915699
w_14=0.3333333283662796
w_15=0.3333333134651184
w_16=0.33333325386047363
w_17=0.33333301544189453
w_18=0.3333320617675781
w_19=0.3333282470703125
w_20=0.33331298828125
w_21=0.333251953125
w_22=0.3330078125
w_23=0.33203125
w_24=0.328125
w_25=0.3125
w_26=0.25
w_27=0.0
w_28=-1.0
w_29=-5.0
w_30=-21.0
w_31=-85.0
w_32=-341.0
w_33=-1365.0
w_34=-5461.0
w_35=-21845.0
w_36=-87381.0
w_37=-349525.0
w_38=-1398101.0
w_39=-5592405.0
w_40=-22369621.0
```

En théorie, on démontre que de telles suites convergent si le coefficient multiplicatif est inférieur à 1 en valeur absolue, sinon elles divergent.

Pour calculer la bonne valeur nous allons utiliser le module `fractions` qui évite les erreurs d'arrondis :

```
from fractions import Fraction
n=0
v = Fraction(1,5)
print(f"v_{n}={v}")
for n in range(1,30):
    v = 6*v-1
    print(f"v_{n}={v}")
```

```
n=0
w = Fraction(1,3)
print(f"w_{n}={w}")
for n in range(1,41):
    w = 4*w-1
    print(f"v_{n}={w}")
```

```
v_0=1/5      v_15=1/5     w_0=1/3      v_15=1/3     v_30=1/3
v_1=1/5      v_16=1/5     v_1=1/3      v_16=1/3     v_31=1/3
v_2=1/5      v_17=1/5     v_2=1/3      v_17=1/3     v_32=1/3
v_3=1/5      v_18=1/5     v_3=1/3      v_18=1/3     v_33=1/3
v_4=1/5      v_19=1/5     v_4=1/3      v_19=1/3     v_34=1/3
v_5=1/5      v_20=1/5     v_5=1/3      v_20=1/3     v_35=1/3
v_6=1/5      v_21=1/5     v_6=1/3      v_21=1/3     v_36=1/3
v_7=1/5      v_22=1/5     v_7=1/3      v_22=1/3     v_37=1/3
v_8=1/5      v_23=1/5     v_8=1/3      v_23=1/3     v_38=1/3
v_9=1/5      v_24=1/5     v_9=1/3      v_24=1/3     v_39=1/3
v_10=1/5     v_25=1/5     v_10=1/3     v_25=1/3     v_40=1/3
v_11=1/5     v_26=1/5     v_11=1/3     v_26=1/3
v_12=1/5     v_27=1/5     v_12=1/3     v_27=1/3
v_13=1/5     v_28=1/5     v_13=1/3     v_28=1/3
v_14=1/5     v_29=1/5     v_14=1/3     v_29=1/3
```

Exercice A.6 (Suite de Muller)

Considérons la suite

$$\begin{cases} x_0 = 4, \\ x_1 = 4.25, \\ x_{n+1} = 108 - \dfrac{815-\frac{1500}{x_{n-1}}}{x_n}. \end{cases}$$

On peut montrer que $\lim_{n\to+\infty} x_n = 5$ (voir par exemple https://scipython.com/blog/mullers-recurrence/) Qu'obtient-on numériquement ?

Correction

```
x=[4, 4.25]
print(f"x_{0}={x[0]}")
print(f"x_{1}={x[1]}")
for i in range(2,30):
    x.append(108-(815-(1500/x[-2]))/x[-1])
    print(f"x_{i}={x[i]}")
```

```
x_0=4                      x_10=4.987909232795786     x_20=100.00001247862016
x_1=4.25                   x_11=4.991362641314552     x_21=100.00000062392161
x_2=4.470588235294116      x_12=4.967455095552268     x_22=100.0000000311958
x_3=4.6447368421052175     x_13=4.42969049830883      x_23=100.00000000155978
x_4=4.770538243625083      x_14=-7.817236578459315    x_24=100.00000000007799
x_5=4.855700712568563      x_15=168.93916767106458    x_25=100.0000000000039
x_6=4.91084749866063       x_16=102.03996315205927    x_26=100.0000000000002
x_7=4.945537395530508      x_17=100.0999475162497     x_27=100.00000000000001
x_8=4.966962408040999      x_18=100.00499204097244    x_28=100.0
x_9=4.980042204293014      x_19=100.0002495792373     x_29=100.0
```

Pour calculer la bonne valeur nous allons utiliser le module 'fractions' qui évite les erreurs d'arrondis :

```
from fractions import Fraction
x=[4, Fraction(17, 4)]
```

```
print(f"x_{0}={x[0]}")
print(f"x_{1}={x[1]}")
for i in range(2,30):
    x.append(108 - Fraction((815 - Fraction(1500, x[-2])), x[-1]))
    print(f"x_{i}={x[i]} ≈ {float(x[i])}")
```

```
x_0=4
x_1=17/4
x_2=76/17 ≈ 4.470588235294118
x_3=353/76 ≈ 4.644736842105263
x_4=1684/353 ≈ 4.770538243626063
x_5=8177/1684 ≈ 4.855700712589074
x_6=40156/8177 ≈ 4.910847499082793
x_7=198593/40156 ≈ 4.945537404123916
x_8=986404/198593 ≈ 4.966962581762701
x_9=4912337/986404 ≈ 4.980045701355631
x_10=24502636/4912337 ≈ 4.987979448478392
x_11=122336033/24502636 ≈ 4.992770288062068
x_12=611148724/122336033 ≈ 4.995655891506634
x_13=3054149297/611148724 ≈ 4.997391268381344
x_14=15265963516/3054149297 ≈ 4.998433943944817
x_15=76315468673/15265963516 ≈ 4.999060071970894
x_16=381534296644/76315468673 ≈ 4.999435937146839
x_17=1907542343057/381534296644 ≈ 4.999661524103767
x_18=9537324294796/1907542343057 ≈ 4.9997969007134175
x_19=47685459212513/9537324294796 ≈ 4.999878135477931
x_20=238423809278164/47685459212513 ≈ 4.9999268795046
x_21=1192108586037617/238423809278164 ≈ 4.999956127061158
x_22=5960511549128476/1192108586037617 ≈ 4.9999736760057125
x_23=29802463602463553/5960511549128476 ≈ 4.999984205520272
x_24=149012035582781284/29802463602463553 ≈ 4.999990523282228
x_25=745059330625296977/149012035582781284 ≈ 4.99999431395856
x_26=3725294111260656556/745059330625296977 ≈ 4.999996588371256
x_27=18626462930705797793/3725294111260656556 ≈ 4.9999979530213565
x_28=93132291776736534004/18626462930705797793 ≈ 4.999998771812312
x_29=465661390253305305137/93132291776736534004 ≈ 4.999999263087206
```

Exercice A.7 (Défi Turing n°86 – Le curieux 2000-ème terme d'une suite)

Considérons la suite

$$\begin{cases} x_0 = \dfrac{3}{2}, \\ x_1 = \dfrac{5}{2}, \\ x_{n+1} = 2003 - \dfrac{6002}{x_n} + \dfrac{4000}{x_n x_{n-1}}. \end{cases}$$

Que vaut u_{2000} ?

Correction

```
x=[3/2, 5/2]
print(f"x_{0}={x[0]}")
print(f"x_{1}={x[1]}")
for i in range(2,2001):
    x.append(2003-6002/x[-1]+4000/(x[-1]*x[-2]))
print(f"x_{i}={x[i]}")
```

```
x_0=1.5                x_1=2.5                x_2000=2000.0
```

Pour calculer la bonne valeur nous allons utiliser le module ‘fractions‘ qui évite les erreurs d’arrondis :

```python
from fractions import Fraction
x=[Fraction(3,2),Fraction(5,3)]
print(f"x_{0}={x[0]}")
print(f"x_{1}={x[1]}")
for i in range(2,2001):
	x.append(2003-Fraction(6002,x[-1])+Fraction(4000,x[-2]*x[-1]))
print(f"x_{i} ≈ {float(x[i])}")
```

```
x_0=3/2                x_1=5/3                x_2000 ≈ 2.0
```

Exercice A.8 (Algorithme de Brent et Salamin)
Soient $(a_n, b_n, s_n)_{n\in\mathbb{N}}$ trois suites définies par

$$\begin{cases} a_0 = 1, \\ b_0 = \frac{1}{\sqrt{2}}, \\ s_0 = \frac{1}{2}, \\ a_{n+1} = \frac{a_n+b_n}{2}, \\ b_{n+1} = \sqrt{a_n b_n}, \\ s_{n+1} = s_n - 2^{n+1}(a_{n+1}^2 - b_{n+1}^2) \end{cases}$$

Vérifier que $p_n \stackrel{\text{déf}}{=} \dfrac{2a_n^2}{b_n} \to \pi$.

Correction
On remarque que, après quelques itérations, les erreurs d’arrondis empêchent la convergence. Pour un calcul exacte on peu utiliser le module sympy :

```python
from math import pi
a, b, s = 1, 1/2**0.5, 1/2
N = 9
for n in range(N):
	a, b = (a+b)/2, (a*b)**0.5
	s = s-2**(n+1)*(a**2-b**2)
	print(f"p_{n}={2*a**2/s:1.15f},
	 ↪ |erreur_{n}|={abs(pi-2*a**2/s):g}")
```

```
p_0=3.187672642712109, |erreur_0|=0.04608
p_1=3.141680293297657, |erreur_1|=8.76397e-05
p_2=3.141592653895460, |erreur_2|=3.05667e-10
p_3=3.141592653589831, |erreur_3|=3.81917e-14
p_4=3.141592653589880, |erreur_4|=8.70415e-14
p_5=3.141592653589978, |erreur_5|=1.84741e-13
p_6=3.141592653590173, |erreur_6|=3.8014e-13
p_7=3.141592653590564, |erreur_7|=7.70939e-13
p_8=3.141592653591346, |erreur_8|=1.55254e-12
```

```python
import sympy
a, b, s = 1, 1/sympy.sqrt(2), sympy.S(1)/2
N = 6
for i in range(N):
	a, b = (a+b)/2, sympy.sqrt(a*b)
	s = s-2**(i+1)*(a**2-b**2)
	print(f"p_{i}={float(2*a**2/s):1.15f},
	 ↪ |erreur_{i}|={float(abs(sympy.pi-2*a**2/s)):g}")
```

```
p_0=3.187672642712108, |erreur_0|=0.04608
p_1=3.141680293297653, |erreur_1|=8.76397e-05
p_2=3.141592653895446, |erreur_2|=3.05653e-10
p_3=3.141592653589793, |erreur_3|=3.71722e-21
p_4=3.141592653589793, |erreur_4|=5.4979e-43
p_5=3.141592653589793, |erreur_5|=1.20269e-86
```

Exercice A.9 (Calcul d’intégrale par récurrence)
On veut approcher numériquement l’intégrale $\mathrm{I}_n = \int_0^1 x^n e^{\alpha x} \mathrm{d}x$ pour $n = 50$. On remarque que, en intégrant par

partie, on a

$$\int x^n e^{\alpha x} \mathrm{d}x = x^n \frac{1}{\alpha} e^{\alpha x} - \frac{n}{\alpha} \int x^{n-1} e^{\alpha x} \mathrm{d}x \tag{A.1}$$

ainsi

$$\begin{aligned} I_n &= \int_0^1 x^n e^{\alpha x} \mathrm{d}x \quad &(A.2)\\ &= \frac{1}{\alpha} e^{\alpha} - \frac{n}{\alpha} I_{n-1} \quad &(A.3)\end{aligned}$$

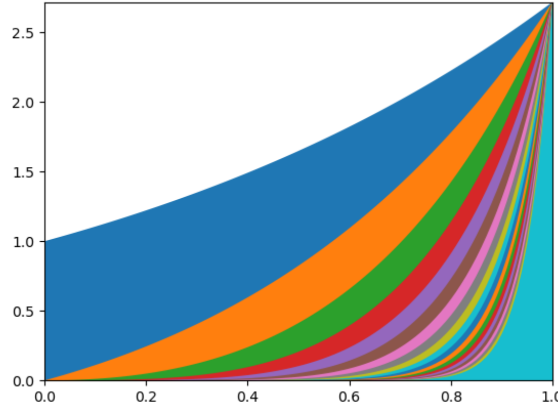
On décide alors de calculer I_{50} par la suite récurrente suivante :

$$\begin{cases} I_0 = \frac{e^{\alpha}-1}{\alpha}, \\ I_{n+1} = \frac{1}{\alpha} e^{\alpha} - \frac{n+1}{\alpha} I_n, \text{pour } n \in \mathbb{N}. \end{cases}$$

Écrire un programme pour calculer cette suite. Comparer le résultat numérique avec la limite exacte $I_n \to 0$ pour $n \to +\infty$.

Correction
On commence par afficher I_n pour différentes valeurs de n : on voit que $I_{n+1} < I_n$.

```
import numpy as np
import matplotlib.pyplot as plt

alpha = 1
x = np.linspace(0,1,101)

plt.axis([0,1,0,max(1,np.exp(alpha))])
line, = plt.plot([],[],lw=2)

for n in range(20):
    plt.fill_between( x, 0, x**n * np.exp(alpha*x) )
plt.show()
```



Si on calcule I_n avec la formule de récurrence avec $\alpha = 1$, on remarque que $0 < I_{n+1} < I_n$ pour $n < 17$, mais $I_{18} < 0$ et la suite est instable. On a le même comportement pour les autres valeurs de α.

```
from math import exp
alpha = 1
I = [(exp(alpha)-1)/alpha]
print(f"I_{0}={I[-1]}")
```

```
for n in range(20):
    I.append(exp(alpha)/alpha -(n+1)*I[-1]/alpha)
    print(f"I_{n} = {I[-1]}")
```

```
I_0=1.718281828459045
I_0 = 1.0
I_1 = 0.7182818284590451
I_2 = 0.5634363430819098
I_3 = 0.4645364561314058
I_4 = 0.395599547802016
I_5 = 0.34468454164694906
I_6 = 0.30549003693040166
I_7 = 0.27436153301583177
I_8 = 0.24902803131655915
I_9 = 0.22800151529345358
I_10 = 0.21026516023105568
I_11 = 0.19509990568637692
I_12 = 0.18198305453614516
I_13 = 0.17051906495301283
I_14 = 0.1604958541638526
I_15 = 0.15034816183740363
I_16 = 0.16236307722318344
I_17 = -0.2042535615582568
I_18 = 6.599099498065924
I_19 = -129.26370813285942
```

La suite obtenue avec le programme ci-dessus ne tend pas vers zéro quand n tend vers l'infini. Pourquoi un tel comportement numérique? Ce comportement est une conséquence directe de la propagation des erreurs d'arrondi : en passant de I_n à I_{n+1}, l'erreur numérique (accumulation des erreurs de représentation et des premiers calculs) est multipliée par n :

$$\begin{aligned}
\varepsilon_{n+1} &= I_{n+1}^{\text{exacte}} - I_{n+1}^{\text{approx}} \\
&= \left(\frac{1}{\alpha}e^{\alpha} - \frac{n+1}{\alpha}I_n^{\text{exacte}}\right) - \left(\frac{1}{\alpha}e^{\alpha} - \frac{n+1}{\alpha}I_n^{\text{approx}}\right) \\
&= -\frac{n+1}{\alpha}(I_n^{\text{exacte}} - I_n^{\text{approx}}) \\
&= -\frac{n+1}{\alpha}\varepsilon_n
\end{aligned}$$

L'erreur numérique $|\varepsilon_n|$ sur l'évaluation de I_n croit donc comme $\dfrac{n!}{\alpha^n}|\varepsilon_0|$.

NB : ici on ne peut pas utiliser le module `fraction` car $e \notin \mathbb{Q}$.

cf. https://scipython.com/book/chapter-9-general-scientific-programming/examples/numerical-stability-of-an-integral-solved-by-recursion/

Exercice A.10 (Évaluer la fonction de Rump)
Évaluer au point $(x, y) = (77617, 33096)$ la fonction de deux variables suivante :

$$f(x, y) = \frac{1335}{4}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11}{2}y^8 + \frac{x}{2y}$$

Correction

```
>>> f = lambda x,y : 1335*y**6/4+x**2*(11*x**2*y**2-y**6-121*y**4-2) + 11*y**8/2+x/(2*y)
>>> print(f(77617,33096))
1.1726039400531787
```

Si on fait le calcul à la main ou on utilise le module `fractions` ou encore le module `sympy` (pour le calcul formel) comme ci-dessous, on trouve une valeur exacte d'environ −0.8273960599 : non seulement l'ordinateur a calculé une valeur très éloignée du résultat mais en plus, le signe n'est même pas le bon.

```
>>> from fractions import Fraction
>>> f = lambda x,y : Fraction(1335,4)*y**6 + x**2*(11*x**2*y**2-y**6-121*y**4-2) \
...                 + Fraction(11,2)*y**8+Fraction(x,2*y)
>>> sol=f(77617,33096)
>>> print(sol,"=",float(sol))
-54767/66192 = -0.8273960599468214
```

```
>>> import sympy
>>> sympy.var('x,y')
(x, y)
>>> g = 1335*y**6/4+x**2*(11*x**2*y**2-y**6-121*y**4-2) + 11*y**8/2+x/(2*y)
>>> sol=g.subs({x:77617, y:33096})
>>> print(sol,"=",sol.evalf())
-54767/66192 = -0.827396059946821
```

Exercice A.11 (Racine du polynôme de Wilkinson)
Le polynôme de Wilkinson

$$p(x) = (x-1)(x-2)\cdots(x-20) = x^{20} - 210x^{19} + 20615x^{18} + \cdots + 2432902008176640000.$$

Ses racines sont $1, 2, \dots, 20$. Cependant, Wilkinson a montré que si on modifie le coefficient de x^{19} de -210 à $-210-2^{-23}$, le nouveau polynôme ainsi obtenu ne s'annule plus en $x = 20$ mais en $x = 20.8$.

Cette sensibilité aux coefficients rend difficile le calcul numérique de ses racines.

Correction

```
>>> import numpy as np
>>> Polynomial = np.polynomial.polynomial.Polynomial
>>> w = Polynomial.fromroots(range(1, 21))
>>> print(w.roots())
[ 1.          2.          3.          4.00000002  4.9999996   6.00000521
  6.99995561  8.00026686  8.99881078 10.00409792 10.98921356 12.02307993
 12.96334362 14.04714444 14.95450431 16.03179803 16.98312518 18.00576725
 18.99876967 20.00011801]
```

On remarque que les petites racines sont assez bien reproduites (elles sont stables par rapport à la perturbation des coefficients) mais que certaines des plus grandes sont très mal reproduites et deviennent complexes en raison de la précision finie de la représentation des coefficients du polynôme.

Source : https://scipython.com/book/chapter-9-general-scientific-programming/examples/wilkinsons-polynomial/

Exercices

1.1. Exercice (Mode interactif) 31
1.2. Exercice (Mode script) 31
1.4. Exercice (Devine le résultat – affectations) 33
1.5. Exercice (Devine le résultat – affectations) 33
1.6. Exercice (Devine le résultat – échanges) 33
1.7. Exercice (Devine le résultat – logique) 34
1.8. Exercice (Séquentialité) 34
1.9. Exercice (Devine le résultat – `string`) 34
1.10. Exercice (Sous-chaînes de caractères) 35
1.11. Exercice (Devine le résultat – ASCII art) 36
1.12. Exercice (Devine le résultat – opérations et conversions de types) 36
1.13. Exercice (Happy Birthday) 36
1.14. Exercice (Compter le nombre de caractères blancs) 37
1.15. Exercice (String + et *) 37
1.16. Exercice (String concatenation) 37
1.17. Exercice (Calculer l'age) 38
1.18. Exercice (Nombre de chiffres de l'écriture d'un entier) 38
1.19. Exercice (Chaîne de caractères palindrome) 39
1.20. Exercice (Nombre palindrome) 39
1.21. Exercice (Conversion h/m/s $\rightsquigarrow$ s) 40
1.22. Exercice (Conversion h/m/s $\rightsquigarrow$ s — *cf.* cours N. MELONI) 40
1.28. Exercice (Devine le résultat – Yoda) 43

2.1. Exercice (Listes et sous-listes) 59
2.2. Exercice (Devine le résultat) 60
2.3. Exercice (Insertions) 61
2.4. Exercice (Moyenne) 61
2.5. Exercice (Effectifs et fréquence) 62
2.6. Exercice (Max-Min) 62
2.7. Exercice (Range) 62
2.8. Exercice (Note ECUE) 63
2.9. Exercice (Devine le résultat - tuples) 63
2.10. Exercice (LE piège avec la copie de listes – I) 64
2.12. Exercice (Copie de listes de listes) 66
2.13. Exercice (Devine le résultat) 67

3.1. Exercice (Devine le résultat) 73
3.2. Exercice (Blanche Neige) 73
3.3. Exercice (Température) 74
3.4. Exercice (Calculer $|x|$) 75
3.5. Exercice (Indice IMC) 75
3.6. Exercice (Note ECUE) 76
3.7. Exercice (Triangles) 76

3.8. Exercice ($ax^2+bx+c=0$) . . . 77

4.1. Exercice (Devine le résultat - `for`) . . . 87
4.2. Exercice (Les animaux sauvages - `for`) . . . 87
4.4. Exercice (Lower to Upper) . . . 89
4.5. Exercice (Vente de voitures) . . . 89
4.6. Exercice (Triangles) . . . 89
4.7. Exercice (Nombre de voyelles) . . . 90
4.8. Exercice (Cartes de jeux) . . . 90
4.9. Exercice (Tables de multiplication) . . . 91
4.10. Exercice (Parcourir deux listes) . . . 91
4.11. Exercice (Parcourir deux chaînes de caractères) . . . 91
4.12. Exercice (Devine le résultat) . . . 92
4.13. Exercice (Devine le résultat - `while`) . . . 93
4.14. Exercice (Suites type ① : $u_n=f(n)$, recherche de seuil (plus petit n tel que...)) . . . 93
4.15. Exercice (Suites type ① : $u_n=f(n)$ (suite géométrique)) . . . 94
4.16. Exercice (Suites type ② : récurrence $u_{n+1}=f(u_n)$) . . . 95
4.17. Exercice (Suites type ③ : récurrence à deux pas $u_{n+1}=f(u_n,u_{n-1})$) . . . 96
4.18. Exercice (Suites type ③ : récurrence à deux pas $u_{n+1}=f(u_n,u_{n-1})$ (Fibonacci)) . . . 97
4.19. Exercice (Suites type ③ : récurrence à deux pas $u_{n+1}=f(u_n,u_{n-1})$ (Fibonacci – bis)) . . . 98
4.21. Exercice (Défi Turing n°2 – Fibonacci) . . . 99
4.22. Exercice (Suites type ③ : récurrence à deux pas $u_{n+1}=f(u_n,u_{n-1})$ (Euclide)) . . . 100
4.23. Exercice (Suites type ④ : récurrence $(u,v)_{n+1}=(f_1(u_n,v_n),f_2(u_n,v_n))$ et affectations //) . . . 100
4.24. Exercice (Suites type ④ : récurrence $(u,v)_{n+1}=(f_1(u_n,v_n),f_2(u_n,v_n))$ et affectations //) . . . 102
4.25. Exercice (Suites type ⑤ : récurrence $(u,v)_{n+1}=(f_1(n,u_n,v_n),f_2(n,u_n,v_n))$ et affectations //) . . . 102
4.26. Exercice ($\sqrt{2\sqrt{2\sqrt{2\sqrt{2\dots}}}}$) . . . 103
4.27. Exercice (4n) . . . 103
4.29. Exercice (Plus petit diviseur) . . . 104
4.30. Exercice (Puissance) . . . 104
4.50. Exercice (Devine le résultat) . . . 115
4.51. Exercice (Cadeaux) . . . 116
4.52. Exercice (Affichage) . . . 116
4.54. Exercice (Maximum d'une liste de nombres sans la fonction `max`) . . . 117
4.55. Exercice (Chaîne la plus longue d'une liste de strings) . . . 117
4.60. Exercice (Entier palindrome dans une base b) . . . 118

5.1. Exercice (Sous-listes) . . . 127
5.2. Exercice (Somme des carrés) . . . 127
5.3. Exercice (Conversion) . . . 127
5.4. Exercice (Chaînes de caractères) . . . 127
5.5. Exercice (Somme des chiffres d'un nombre) . . . 128
5.6. Exercice (Défi Turing n°5 – somme des chiffres d'un nombre) . . . 128
5.7. Exercice (Liste de Moyennes) . . . 128
5.8. Exercice (Morceau) . . . 128
5.9. Exercice (Position du minimum d'une liste de nombres) . . . 128

 © G. Faccanoni

5.10. Exercice (Chaturanga) . . . 129
5.11. Exercice (Filtrer une liste) . . . 130
5.12. Exercice (Liste des diviseurs) . . . 130
5.13. Exercice (Nombres parfaits) . . . 131
5.16. Exercice (Liste de nombres) . . . 132
5.18. Exercice (Soustraire deux listes) . . . 133
5.19. Exercice (Jours du mois) . . . 133
5.20. Exercice (Conversion de températures) . . . 134
5.21. Exercice (Années bissextiles) . . . 134
5.22. Exercice . . . 135
5.23. Exercice (Produit matriciel) . . . 136
5.24. Exercice (Nombres triangulaires) . . . 137
5.26. Exercice (Table de multiplication) . . . 138
5.27. Exercice (Défi Turing n°1 – somme de multiples) . . . 139
5.28. Exercice (Nombres de Armstrong) . . . 140
5.29. Exercice (Nombre de chiffres) . . . 140
5.30. Exercice (Défi Turing n°85 – Nombres composés de chiffres différents) . . . 140

6.1. Exercice (Devine le résultat - variables globales *vs* locales) . . . 159
6.3. Exercice (Devine le résultat - bêtisier) . . . 161
6.4. Exercice (Devine le résultat) . . . 162
6.5. Exercice (J'écris mes premières fonctions) . . . 163
6.6. Exercice (Valeur absolue) . . . 164
6.7. Exercice (Pair Impair) . . . 165
6.8. Exercice (Premières fonctions λ) . . . 165
6.9. Exercice (Doublons) . . . 167
6.10. Exercice (Divisibilité) . . . 167
6.11. Exercice (Multiple de 10 le plus proche) . . . 168
6.12. Exercice (Radars routiers) . . . 168
6.15. Exercice (Masquer tickets de caisse) . . . 170
6.17. Exercice (Liste qui transforme en 0 les termes à partir du premier 0) . . . 171
6.18. Exercice (Password) . . . 171
6.19. Exercice (Distance de Hamming) . . . 172
6.20. Exercice (Nombre de Harshad) . . . 172
6.21. Exercice (Validité d'un code de photocopieuse) . . . 173
6.22. Exercice (Numéro de sécurité sociale) . . . 173
6.23. Exercice (Numéro ISBN-10) . . . 174
6.25. Exercice (Algorithme d'Euclide) . . . 175
6.26. Exercice (Indicatrice d'Euler) . . . 176
6.27. Exercice (Prix d'un billet) . . . 177
6.28. Exercice (Rendu monnaie) . . . 178
6.30. Exercice (Can Balance) . . . 179
6.32. Exercice (Liste impairs) . . . 180
6.33. Exercice (Liste divisibles) . . . 181
6.34. Exercice (Nombre miroir) . . . 181
6.35. Exercice (Point milieu) . . . 181

6.36. Exercice (Normaliser une liste) . . . 182
6.37. Exercice (Couper un intervalle) . . . 182
6.38. Exercice (Représentation et manipulation de polynômes) . . . 183
6.39. Exercice (Triangle de Pascal) . . . 186
6.40. Exercice (Voyelles) . . . 187
6.41. Exercice (Pangramme) . . . 188
6.42. Exercice (Swap) . . . 188
6.43. Exercice (OVNI) . . . 189
6.44. Exercice (Nombres premiers) . . . 189
6.50. Exercice (F: $\mathbb{R} \to \mathbb{R}^2$) . . . 193
6.53. Exercice (Code César) . . . 194
6.60. Exercice (Palindromes) . . . 200
6.65. Exercice (Écriture d'un entier dans une base quelconque et entiers brésiliens) . . . 202

7.1. Exercice (Module `math` – sin, cos, tan, π, e) . . . 225
7.3. Exercice (Module `math` - Angles remarquables) . . . 226
7.4. Exercice (Module `math` - Paper folding) . . . 227
7.5. Exercice (Module `math` – Factorielle) . . . 227
7.6. Exercice (Module `math` – Stirling) . . . 228
7.7. Exercice (Module `math` – Défi Turing n°6 et Projet Euler n°20 – somme de chiffres) . . . 228
7.8. Exercice (Module `math` – Nombres de Brown) . . . 229
7.9. Exercice (Module `math` – Approximation de e) . . . 229
7.10. Exercice (Module `math` – Polignac) . . . 230
7.11. Exercice (Module `math` – Énoncer des chiffres) . . . 230
7.12. Exercice (Module `math` – Overflow) . . . 231
7.13. Exercice (Module `math` – Distance) . . . 231
7.14. Exercice (Module `math` – Distance point droite) . . . 234
7.18. Exercice (Réduire une fraction : module `math` puis module `fractions`) . . . 238
7.20. Exercice (Module `random` - Jeu de dé) . . . 239
7.21. Exercice (Module `random` - Calcul de fréquences) . . . 240
7.22. Exercice (Module `random` - Puce) . . . 240
7.25. Exercice (Module `math` – Cylindre) . . . 242
7.26. Exercice (Module `math` – Formule d'Héron) . . . 242
7.27. Exercice (Module `math` – Formule de Kahan) . . . 242
7.28. Exercice (Module `math` – Cercle circonscrit) . . . 243
7.30. Exercice (Module `random` - Distance moyenne entre deux points aléatoires d'un carré) . . . 245
7.31. Exercice (Module `random` - Kangourou) . . . 245
7.32. Exercice (Module `math` – sin cos) . . . 246
7.33. Exercice (Approximations de ln) . . . 246
7.34. Exercice (Approximations de π) . . . 247
7.35. Exercice (Module `scipy` - Calcul approché d'une intégrale (méthode Monte-Carlo)) . . . 255
7.36. Exercice (Module `numpy` - Statistique) . . . 255
7.37. Exercice (Module `numpy` - Statistique et suites) . . . 256
7.44. Exercice (Approximation valeur ponctuelle dérivées) . . . 262

8.1. Exercice (Bhaskara I) . . . 275

8.2. Exercice (Tracer des droites) . . . 275
8.3. Exercice (Tracer une fonction définie par morceaux) . . . 276
8.6. Exercice (Module `scipy` - Calcul approché d'une intégrale) . . . 279
8.7. Exercice (Dépréciation ordinateur) . . . 280
8.8. Exercice (Mercato) . . . 281
8.9. Exercice (Résolution graphique d'une équation) . . . 282
8.10. Exercice (Résolution graphique d'une équation) . . . 282
8.11. Exercice (Tracer plusieurs courbes) . . . 283
8.12. Exercice (Courbe paramétrée) . . . 284
8.13. Exercice (Courbe paramétrée) . . . 284
8.14. Exercice (Courbe polaire) . . . 285
8.15. Exercice (Courbe polaire) . . . 285
8.16. Exercice (Proies et prédateurs) . . . 286
8.17. Exercice (Coïncidences et anniversaires) . . . 287
8.18. Exercice (Conjecture de Syracuse) . . . 289
8.21. Exercice (Notation cistercienne) . . . 295

A.1. Exercice (Devine le résultat) . . . 307
A.2. Exercice (Qui est plus grand ?) . . . 307
A.3. Exercice (Integer VS Floating point number) . . . 308
A.4. Exercice (Un contre-exemple du dernier théorème de Fermat ?) . . . 308
A.5. Exercice (Suites) . . . 309
A.6. Exercice (Suite de Muller) . . . 311
A.7. Exercice (Défi Turing n°86 – Le curieux 2000-ème terme d'une suite) . . . 312
A.8. Exercice (Algorithme de Brent et Salamin) . . . 313
A.9. Exercice (Calcul d'intégrale par récurrence) . . . 313
A.10. Exercice (Évaluer la fonction de Rump) . . . 315
A.11. Exercice (Racine du polynôme de Wilkinson) . . . 316

Exercices ☆

1.3. ☆ Exercice Bonus (Rendre un script exécutable : la ligne *shebang*) 32
1.23. ☆ Exercice Bonus (Format table) 40
1.24. ☆ Exercice Bonus (Tables de vérité) 41
1.25. ☆ Exercice Bonus (Années martiennes — *cf.* cours N. MELONI) 42
1.26. ☆ Exercice Bonus (Changement de casse & Co.) 42
1.27. ☆ Exercice Bonus (Comptage, recherche et remplacement) 42

2.11. ☆ Exercice Bonus (LE piège avec la copie de listes – II) 65
2.14. ☆ Exercice Bonus (`str`⇝`list` : `split` et `join`) 68
2.15. ☆ Exercice Bonus (Liste de tuples et Tuple de listes) 68

3.9. ☆ Exercice Bonus (Pierre Feuille Ciseaux) 78
3.10. ☆ Exercice Bonus (Pierre, feuille, ciseaux, lézard, Spock) 78

4.28. ☆ Exercice Bonus (Défi Turing n°70 – Permutation circulaire) 104
4.31. ☆ Exercice Bonus (Défis Turing n°72) 105
4.32. ☆ Exercice Bonus (Dictionnaire) 105
4.33. ☆ Exercice Bonus (Dictionnaires : construction d'un histogramme) 105
4.34. ☆ Exercice Bonus (Dictionnaires : The Most Frequent) 106
4.56. ☆ Exercice Bonus (Défi Turing n°9 – triplets pythagoriciens) 117
4.57. ☆ Exercice Bonus (Défi Turing n°11 - nombre miroir) 118
4.58. ☆ Exercice Bonus (Défi Turing n°13 – Carré palindrome) 118
4.59. ☆ Exercice Bonus (Défi Turing n°43 – Carré palindrome) 118
4.61. ☆ Exercice Bonus (Défi Turing 22 – Les anagrammes octuples) 119
4.62. ☆ Exercice Bonus (Défi Turing n°21 – Bonne année 2013 !) 119

5.14. ☆ Exercice Bonus (Défi Turing n°18 – Somme de nombres non abondants) 131
5.15. ☆ Exercice Bonus (Défis Turing n°71 – ensembles) 132
5.25. ☆ Exercice Bonus (Nombres pentagonaux) 137
5.34. ☆ Exercice Bonus (Défi Turing n° 40 – La constante de Champernowne) 141
5.38. ☆ Exercice Bonus (Défi Turing n°29 – puissances distincts) 143
5.39. ☆ Exercice Bonus (Défi Turing n°45 – Nombre triangulaire, pentagonal et hexagonal) 143
5.40. ☆ Exercice Bonus (Défi Turing n°94 – Problème d'Euler n° 92) 144
5.44. ☆ Exercice Bonus (Défi Turing n°60 – Suicide collectif) 146
5.45. ☆ Exercice Bonus (Triplets pytagoriciens) 147
5.46. ☆ Exercice Bonus (Dictionnaire ordonné) 147

6.2. ☆ Exercice Bonus (Devine le résultat - Function scope) 160
6.13. ☆ Exercice Bonus (Soirée parents-enfants) 169
6.14. ☆ Exercice Bonus (Leet speak) 169
6.16. ☆ Exercice Bonus (Fusion de listes) 170
6.24. ☆ Exercice Bonus (CheckSum) 174
6.31. ☆ Exercice Bonus (Matching Brackets) 179
6.45. ☆ Exercice Bonus (Mot parfait) 190
6.48. ☆ Exercice Bonus (Défi Turing n° 19 – Rencontre du quatrième type) 191
6.51. ☆ Exercice Bonus (Défi Turing n° 52 – multiples constitués des mêmes chiffres) 193
6.52. ☆ Exercice Bonus (Défi Turing n° 61 – Non à l'isolement !) 194
6.54. ☆ Exercice Bonus (Défi Turing n° 17 – Nombres amicaux) 195
6.55. ☆ Exercice Bonus (`map`) 196

6.56. Exercice Bonus (Tickets t+ RATP) . . . 196
6.57. Exercice Bonus (Tickets réseau Mistral) . . . 197
6.58. Exercice Bonus (Problème d'Euler n°9) . . . 199
6.61. Exercice Bonus (Défi Turing n°73 – Palindrome et carré palindrome) . . . 201
6.62. Exercice Bonus (Défi Turing n°4 – nombre palindrome) . . . 201
6.66. Exercice Bonus (Suites de Kaprekar)) . . . 203
6.69. Exercice Bonus (Défi Turing n°141 – Combien de 6 ?) . . . 205
6.70. Exercice Bonus (Défi Turing n°33 – Pâques en avril) . . . 206
6.71. Exercice Bonus (Sun angle) . . . 207
6.72. Exercice Bonus (Angle entre les aiguilles d'une horloge) . . . 208
6.73. Exercice Bonus (Most Wanted Letter) . . . 208
6.74. Exercice Bonus (Bigger Price) . . . 209
6.75. Exercice Bonus (Sum by Types) . . . 210
6.76. Exercice Bonus (Common Words) . . . 210
6.77. Exercice Bonus (Flatten list) . . . 211
6.78. Exercice Bonus (Plan cyclique) . . . 212

7.2. Exercice Bonus ($\sqrt{\tan(\pi)}$) . . . 226
7.15. Exercice Bonus (Module `math` – Distance point segment) . . . 235
7.16. Exercice Bonus (Module `math` – Longueur d'une courbe) . . . 236
7.19. Exercice Bonus (Module `Fraction` – Triangles semblables) . . . 238
7.23. Exercice Bonus (Module `random` - Le nombre mystère) . . . 240
7.24. Exercice Bonus (Module `random` - Yahtzee) . . . 241
7.29. Exercice Bonus (Centre et rayon d'un cercle pour un arc donné) . . . 244
7.38. Exercice Bonus (Module `numpy` - Représentation et manipulation de polynômes) . . . 257
7.39. Exercice Bonus (Module `sympy` - Représentation et manipulation de polynômes) . . . 259
7.40. Exercice Bonus (Permutations avec itertools) . . . 259
7.41. Exercice Bonus (Permutations avec itertools) . . . 260
7.42. Exercice Bonus (Module `sympy` – devine le résultat) . . . 261
7.43. Exercice Bonus (Module `sympy` – calcul formel d'une dérivée) . . . 262
7.45. Exercice Bonus (Module `sympy` – composition de fonctions) . . . 262
7.46. Exercice Bonus (Module `sympy` – calcul des paramètres) . . . 262
7.47. Exercice Bonus (Module `sympy` – calcul des paramètres) . . . 263
7.48. Exercice Bonus (Module `sympy` – calcul de paramètres) . . . 263

8.4. Exercice Bonus (Approximations de deux fonctions polynomiales) . . . 277
8.5. Exercice Bonus (Approximations de deux fonctions trigonométriques) . . . 278
8.19. Exercice Bonus (Cuve de fioul enterrée) . . . 291
8.20. Exercice Bonus (Le crible de MATIYASEVITCH) . . . 293
8.22. Exercice Bonus (Taux Marginal, Taux Effectif : comprendre les impôts) . . . 296

Pydéfis

4.3. Exercice Bonus (Pydéfis – L'algorithme du professeur Guique) . . . 88
4.20. Exercice Bonus (Pydéfis – Fibonacci) . . . 99
4.35. Exercice Bonus (Pydéfis – Vous parlez Fourchelangue ?) . . . 106
4.36. Exercice Bonus (Pydéfis - Code Konami) . . . 107
4.37. Exercice Bonus (Pydéfis - Difficile de comprendre un lapin crétin) . . . 107
4.38. Exercice Bonus (Pydéfis – Le retourneur de temps) . . . 109
4.39. Exercice Bonus (Pydéfis – Entrée au Ministère) . . . 109
4.40. Exercice Bonus (Pydéfis – Créatures nocturnes pas si sympathiques que cela...) . . . 110
4.41. Exercice Bonus (Pydéfis – SW I : À l'assaut de Gunray. Découpage de la porte blindée) . . . 110
4.42. Exercice Bonus (Pydéfis – L'hydre de Lerne) . . . 111
4.43. Exercice Bonus (Pydéfis – Le sanglier d'Érymanthe) . . . 111
4.44. Exercice Bonus (Pydéfis – Les dragées surprises) . . . 112
4.45. Exercice Bonus (Pydéfis – Les tubes à essai d'Octopus. Méli-mélo de poison) . . . 113
4.46. Exercice Bonus (Pydéfis – Désamorçage d'un explosif (I)) . . . 114
4.47. Exercice Bonus (Pydéfis – Méli Mélo de nombres) . . . 114
4.48. Exercice Bonus (Pydéfis – Suite Tordue) . . . 115
4.49. Exercice Bonus (Pydéfis – Bombe à désamorcer) . . . 115
4.53. Exercice Bonus (Pydéfis – Insaisissable matrice) . . . 116
4.63. Exercice Bonus (Pydéfis – La suite Q de Hofstadter) . . . 119
4.64. Exercice Bonus (Pydéfis – L'escargot courageux) . . . 120
4.65. Exercice Bonus (Pydéfis – Mon beau miroir...) . . . 120
4.66. Exercice Bonus (Pydéfis – Persistance) . . . 121
4.67. Exercice Bonus (Pydéfis – Toc Boum) . . . 121
4.68. Exercice Bonus (Pydéfis – Les juments de Diomède) . . . 121
4.69. Exercice Bonus (Pydéfis – Produit et somme palindromiques) . . . 122

5.17. Exercice Bonus (Pydéfi – SW III : L'ordre 66 ne vaut pas 66...) . . . 133
5.31. Exercice Bonus (Pydéfi – Piège numérique à Pokémons) . . . 140
5.32. Exercice Bonus (Pydéfi – Le jardin des Hespérides) . . . 141
5.33. Exercice Bonus (Pydéfi – Constante de Champernowne) . . . 141
5.35. Exercice Bonus (Pydéfi – Nos deux chiffres préférés) . . . 142
5.36. Exercice Bonus (Pydéfi – Série décimée...) . . . 142
5.37. Exercice Bonus (Pydéfi – Désamorçage de bombe à distance (II)) . . . 142
5.41. Exercice Bonus (Pydéfi – Le pistolet de Nick Fury) . . . 144
5.42. Exercice Bonus (Pydéfi – Les nombres heureux) . . . 145
5.43. Exercice Bonus (Pydéfi – Le problème des boîtes à sucres) . . . 146

6.29. Exercice Bonus (Pydéfis – Monnaie) . . . 178
6.46. Exercice Bonus (Pydéfis – Premier particulier (1)) . . . 191
6.47. Exercice Bonus (Pydéfi – Einstein) . . . 191
6.49. Exercice Bonus (Pydéfi – Vif d'or) . . . 192
6.59. Exercice Bonus (Pydéfis – Cerbère) . . . 200
6.63. Exercice Bonus (Pydéfi – Les bœufs de Géryon) . . . 202

6.64. Exercice Bonus (Pydéfi – Le lion de Némée) 202
6.67. Exercice Bonus (Pydéfi – Numération Gibi : le Neutubykw) 205
6.68. Exercice Bonus (Pydéfi – Pokédex en vrac) 205

7.17. Exercice Bonus (Pydéfis – La biche de Cyrénée) 237

 © G. Faccanoni