Licence Sciences et Techniques

L1 MATHS - PI-M-11

UNIVERSITÉ DE TOULON

Initiation à la programmation informatique avec python

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 2020 – 2021

Dernière mise-à-jour
Mardi 30 juin 2020

Programme Indicatif

Semaine	CM (lundi)	TD (lundi)	TP (vendredi)
42	CM-1 : Ch. 1-2	–	–
43	–	TD-1 : préparation TP 1-2	–
45	CM-2 : Ch. 3-4	–	TP-1 : 1.1 - 1.2 - 1.3 - 1.4 - 1.5 - 1.6 - 1.7 - 1.8 - 1.9 - 1.10 - 1.11 - 1.12 - 1.15 - 1.16 - 1.17 - 1.18 - 1.19 - 1.20 1.21 - 1.22 - 1.23
46	–	TD-2 : préparation TP 3-4	TP-2 : 2.8 - 2.3 - 2.4 - 2.5 - 2.6 - 2.7 3.1 - 3.2 - 3.4 - 3.5 - 3.6 - 3.7 - 3.8
47	CM-3 : Ch. 5.1-6	–	TP-3 : 4.7 - 4.8 - 4.1 - 4.11 - 4.12 - 4.3 - 4.5 - 4.6 - 4.10 - 4.13 - 7.17 - 4.14 - 4.15 - 4.24 - 4.25 - 4.26 - 4.27 - 4.28 - 4.31 4.32 - 4.34 - 4.35 - 4.36 - 4.37 - 4.40
48	–	TD-3 : préparation TP 5-6	TP-4 : 5.2 - 5.3 - 5.8 - 5.9 - 5.1 - 5.5 - 5.11 5.12 - 5.13 - 5.14 - 5.15 - 5.16 - 5.17 - 5.18 - 5.19 - 5.21 - 5.32 - 5.33
49	CM-4 : Ch. 7-8	–	TP-5 : 6.1 - 6.2 - 6.3 - 6.4 - 6.10 - 6.11 - 6.12 - 6.13 - 6.15 - 6.16 - 6.17 - 6.21 6.26 - 6.27 - 6.28 - 6.29 - 6.30 - 6.32
50	–	TD-4 : préparation TP 7	TP-6 : 7.1 - 7.2 - 7.5 - 7.7 - 7.8 - 7.9 - 7.10 - 7.11 - 7.12 - 7.13 - 7.14 - 6.18 - 7.15 - 7.18 - 7.19 - 7.20 - 6.20 - 7.21 - 7.22 - 7.23 7.24 - 7.25 - 7.26- 7.27
51	–	–	TP-7 : 8.1 - 8.2 - 8.3 - 8.5 - 8.10 - 8.11 8.12 - 8.13 - 8.14 - A.1 - A.2 - A.3 - A.4 - A.5 - A.7 - A.8
1			TP-8 : CC (en salle de TP)
2			CT (en salle de TP)

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

Les besoins d'un mathématicien (appliqué)

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

- ▷ 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 : vous apprendre à résoudre des problèmes (souvent issus des mathématiques) en langage algorithmique et être capable d'écrire des petits programmes en Python qui implémentent ces algorithmes.

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, SymPy)
M43 au semetre 4 (L2) : *TP mathématiques* (notebook IPython, modules NumPy, SymPy)
M62 au semetre 6 (L3) : *analyse numérique* (notebook IPython)

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

Déroulement du cours et évaluation

CM 6h : 4 séances de 1h30
TD 6h : 4 séances de 1h30
TP 24h : 8 séances de 3h
CC 3h : 1 séance de 3h en salle de TP le vendredi 20 décembre 2019
CT 3h : 1 séance de 3h en salle de TP en janvier 2020
Note finale $\max\{ CT; 0.3CC + 0.7CT \}$

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 polycopié est là pour éviter un travail de copie qui empêche parfois de se concentrer sur les explications données oralement mais il est loin d'être exhaustif ! De plus, ne vous étonnez pas si vous découvrez des erreurs (merci de me les communiquer).

On a inclus dans ce texte nombreux exercices (plus de 200) de difficulté variée et dont la correction est disponible sur ma page web. Cependant, veuillez noter que le seul moyen d'apprendre est de comprendre et pour cela il faut **essayer par soi-même !** Ces exercices (et leurs corrections) sont présents non pour être lus mais pour être résolus. En effet, la lecture ne sert rigoureusement à rien si elle n'a pas été accompagnée de recherche. Pour que la méthode d'étude soit vraiment efficace, il faut faire l'effort de se creuser la tête plutôt que de copier-coller la correction. En particulier, il faut avoir un papier brouillon à coté de soi et un crayon. La première étape consiste

alors à traduire l'énoncé, en particulier s'il est constitué de beaucoup de jargon mathématique. Ensuite il faut essayer d'écrire un algorithme (une recette, une suite d'instructions). C'est ici que l'intuition joue un grand rôle et il ne faut pas hésiter à remplir des pages pour s'apercevoir que l'idée qu'on a eu n'est pas la bonne. Elle pourra toujours resservir dans une autre situation. Quand finalement on pense tenir le bon bout, il faut rédiger soigneusement en s'interrogeant à chaque pas sur la validité (logique et mathématique) de ce qu'on a écrit et si l'on a pris en compte tous les cas possibles.

Bien-sûr, au cours des exercices proposés, il est possible de bloquer. Dès lors, pour progresser, il ne faut surtout pas hésiter à s'aider du cours ou à demander de l'aide à ces camarades et à l'enseignant bien sûr.

Enfin, pour les corrections, il est tout à fait possible que votre code ne soit pas semblable au mien. Pas d'inquiétude : le principal est qu'il soit fonctionnel. Cependant, 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 préparé 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 réponse 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 !")
```

```
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, dans des fenêtres appelées `Trinkets` 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://trinket.io/python
- ▷ From blocks to code
  https://hourofpython.trinket.io/from-blocks-to-code-with-trinket

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

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.

Installer Anaconda

Étant donné qu'au deuxième semestre nous allons travailler dans des `notebook IPython`, je vous conseille d'installer Anaconda qui installera Python avec des modules utiles en mathématiques (`Matplotlib`, `NumPy`, `SciPy`, `SymPy` etc.), ainsi que `IPython` et `Spyder`. Les procédures d'installations détaillées selon chaque système d'exploitation sont décrites à l'adresse : https://docs.anaconda.com/anaconda/install/. 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.

Quel éditeur ?

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.

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. Il y a une énorme diversité d'IDE et on peut constater que certains IDE sont conçus spécialement pour un langage

Je déconseille l'utilisation de Pycharm pour les néophytes.

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 3.7 a été publiée en juin 2018. La version 2 de Python est désormais obsolète et cessera d'être maintenue après le 1er janvier 2020. Dans la mesure du possible évitez de l'utiliser.

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.

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.
- 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 : [1]

- ▷ dans un terminal écrire `python3` 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» :

```
>>> 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=5
>>> b=10
>>> c=a*b
>>> print(a,b,c)
5 10 50
```

- ▷ 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 comme `Gedit` ou `Geany`. On voit qu'il n'y a rien dans cette nouvelle fenêtre (pas d'en-tête 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.
- ▷ 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 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.

1. 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.

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

Le symbole dièse (#) indique le début d'un commentaire : tous les caractères entre # et la fin de la ligne sont ignorés par l'interpréteur.

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.).

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
```

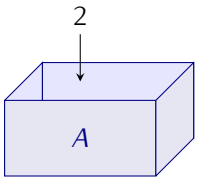
1.4. Variables et affectation

Une variable peut être vue comme une boîte représentant un emplacement en mémoire qui permet de stocker une valeur, et à qui on a donné un nom afin de facilement l'identifier (boîte ← valeur).

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 $A \leftarrow 2$).

Il est très important de donner un nom clair et précis aux variables. Par exemple, 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

Python distingue les majuscules des minuscules. Donc `mavariable`, `Mavariable` et `MAVARIABLE` sont des variables différentes.

Les noms de variables peuvent être non seulement des lettres, mais aussi des mots ; ils peuvent contenir des chiffres, ainsi que certains caractères spéciaux comme le tiret bas «_» (appelé *underscore* en anglais).

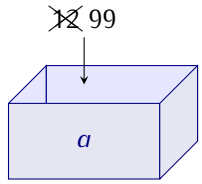
Par ailleurs, un nom de variable ne peut pas utiliser d'espace, ne doit pas débuter par un chiffre et il n'est pas recommandé de le faire débuter par le caractère _ (sauf cas très particuliers).

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
↪ ...
```

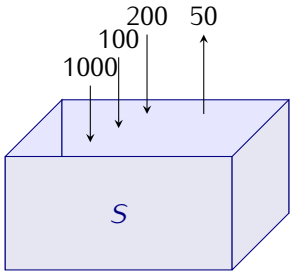
Une fois une variable initialisée, on peut modifier sa valeur en utilisant de nouveau l'opérateur d'affectation (=). La valeur actuelle de la variable est remplacée par la nouvelle valeur qu'on lui affecte. Dans l'exemple suivant, on initialise une variable à la valeur 12 et on remplace ensuite sa valeur 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) :

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



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».

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
>>> 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.

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

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

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

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
>>> a=c
>>> c=b
>>> b=a
>>> print(a); print(b)
4
4
```

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

1.5. Nombres

Il y a trois types numériques en Python :

- ▷ Le type entier `int` permet de représenter n'importe quel nombre entier, peu importe sa taille.
- ▷ Le type flottant `float` 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^{-324} et 10^{308}. La valeur spéciale `math.inf` représente l'infini (voir chapitre sur les modules).
- ▷ Le type complexe `complex` permet de représenter des nombres complexes, où le nombre imaginaire se note `j` (par exemple `3.1+5.2j`)

1.6. Opérations arithmétiques

Dans Python on a les opérations arithmétiques usuelles :

+	Addition
-	Soustraction
*	Multiplication
/	Division
**	Exponentiation
//	Quotient de la division euclidienne
%	Reste de la division euclidienne

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
```

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.

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.

Il existe aussi les 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

1.7. Opérateurs de comparaison et connecteurs logiques

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

`<` signifie $<$
`>` signifie $>$
`<=` signifie $\leq$
`>=` signifie $\geq$
`==` signifie $=$
`!=` signifie $\neq$
`in` signifie $\in$

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	Ça signifie
and	et
or	ou
not	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
```

Exemple (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 $\neg$, $\wedge$, $\vee$, $\Longrightarrow$, $\Longleftrightarrow$, les fonctions de vérités du calcul propositionnel classique sont données par la table de vérité suivante :

P	Q	$\neg(P)$	$\neg(Q)$	$P \wedge Q$	$P \vee Q$	$P \Longrightarrow Q$	$Q \Longrightarrow P$	$P \Longleftrightarrow Q$
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

Voici comment peut-on vérifier la table qu'on a écrit :

```
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 {
```

```
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
```

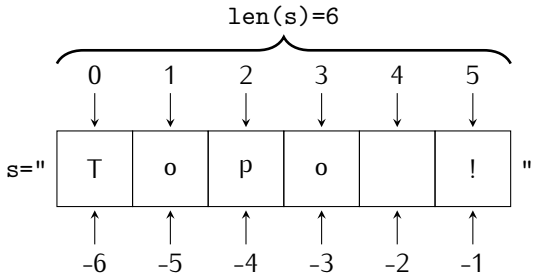
1.8. Chaîne de caractères (Strings)

Une *chaîne de caractères* est une séquence de caractères entre guillemets (simples ou doubles).

En Python, les éléments d'une chaîne sont *indexés à partir de* 0 et non de 1.

```
>>> s = 'Topolino'
>>> print(s[2])
p
```

On peut extraire une sous-chaîne en déclarant l'indice de **début (inclus)** et l'indice de **fin (exclu)**, séparés par deux-points : `s[i:j]`, ou encore une sous-chaîne en déclarant l'indice de début (inclus), l'indice de fin (exclu) et le pas, séparés par des deux-points : `s[i:j:k]`. Cette opération est connue sous le nom de *slicing* (en anglais). Un petit dessin et quelques exemples permettront de bien comprendre cette opération fort utile :




```
>>> s='Topo !'
>>> s[2:4]
'po'
>>> s[2:]
'po !'
>>> s[:2]
'To'
>>> s[:]
'Topo !'
>>> s[2:5]
'po '
```

```
>>> s[2:9]
'po !'
>>> s[1:6:2]
'oo!'
>>> s[-4:-2]
'po'
>>> s[-1]
'!'
>>> s[-1:-7:-1]
'! opoT'
```

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
```

À noter que lorsqu'on utilise des tranches, les dépassements d'indices sont licites.

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
```

Voici quelques opérations et méthodes très courantes associées aux chaîne de caractères :

Opérations :	`s1 + s2`	concatène la chaîne `s1` à la chaîne `s2`
	`"x" in s`	renvoi `True` si la chaîne `s` contient l'élément `"x"`, `False` sinon
	`"x" not in a`	renvoi `True` si la chaîne `s` ne contient pas l'élément `"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`

Exemples :

```
>>> n=123.45
>>> s=str(n)
>>> print(s[0:3])
123

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

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

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

>>> print(len(s))
34
>>> print("g" in s)
True
>>> print("x" in s)
True
```

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")`	remplace chaque sous-chaîne `"x"` par la sous-chaîne `"y"` dans la chaîne `s`

Exemples :

```
>>> string1 = 'Minnie'
>>> string2 = 'Topolino'

>>> s3 = string1 + ' et ' + string2
>>> print(s3)
Minnie et Topolino

>>> s4 = s3.replace('et','&')
>>> print(s4)
Minnie & Topolino

>>> texte= "Une fonctttion 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(s.index("n"))
```

```
11
>>> print(s.count("n"))
1

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

Attention

Attention, l'opérateur de concaténation (+) ne fonctionne qu'entre deux données de type chaîne de caractères :

```
>>> s = 'Hello '
>>> t = 'to you'
>>> print(3*s) # Repetition
Hello Hello Hello
>>> print(s+t) # Concatenation
Hello to you
```

Si vous tentez de l'utiliser avec un autre type, l'interpréteur Python produira une erreur d'exécution.

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

L'interpréteur Python va générer une erreur d'exécution qui signale qu'il ne parvient pas à convertir implicitement la donnée de type `int` en une donnée de type `str`. On doit alors écrire

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

1.9. La fonction `print`

Pour afficher à l'écran des objets on utilise la fonction `print(object1, object2,...)` qui convertit `object1`, `object2` en chaînes de caractères et les affiche sur la même ligne séparés par des espace.

```
>>> a = 12345,6789
>>> b = [2, 4, 6, 8]
>>> print(a,b)
(12345, 6789) [2, 4, 6, 8]
```

Plus précisément, la fonction `print()` affiche l'argument qu'on lui passe entre parenthèses **et un retour à ligne**. Ce retour à ligne supplémentaire est ajouté par défaut. Si toutefois, on ne veut pas afficher ce retour à la ligne, on peut utiliser l'argument par «mot-clé» `end` :

```
>>> a = 12345,6789
>>> b = [2, 4, 6, 8]
>>> print(a,b)
(12345, 6789) [2, 4, 6, 8]
>>> print(a,b,end="")
(12345, 6789) [2, 4, 6, 8]
```

Le retour à la ligne peut être forcé par le caractère `\n`, la tabulation par le caractère `\t` :

```
>>> a = 12345,6789
>>> b = [2, 4, 6, 8]
>>> print("a =", a, "\nb =", b)
a = (12345, 6789)
b = [2, 4, 6, 8]
```

```
>>> print("a =", a, "\tb =", b)
a = (12345, 6789) ⟶b = [2, 4, 6, 8]
```

1.9.1. Écriture formatée

Pour mettre en colonne des nombres on pourra utiliser

- ▷ soit l'opérateur % : la commande `print('%format1, %format2,...' %(n1,n2,...))` affiche les nombres `n1,n2,...` selon les règles `%format1, %format2,...`. Typiquement on utilise
  - ▷ *w.d* pour un entier,
  - ▷ *w.d*f pour un nombre en notation *floating point*,
  - ▷ *w.d*e pour un nombre en notation scientifique.

  où *w* est la largeur du champ total et *d* le nombre de chiffres après la virgule. Voici quelques exemples :

```
>>> a = -1234.56789
>>> n = 9876
>>> print('%7.2f' %a)
-1234.57
>>> print('n = %6d' %n)
n =   9876
>>> print('n = %06d' %n)
n = 009876
>>> print('%12.3f %6d' %(a,n))
   -1234.568   9876
>>> print('%12.4e %6d' %(a,n))
 -1.2346e+03   9876
```

- ▷ soit la méthode `.format()` : la commande `print('{},{},...'.format(n1,n2,...))` affiche les nombres `n1,n2,...`. Une version abrégée de la même commande est `print(f'{n1},{n2},...')` mais elle n'est disponible qu'à partir de la version 3.6 de python.

  Voici quelques exemples pour le choix dès règles de formatage :

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

>>> print('a={}'.format(a))    # idem que print(f'a={a}')
a=-1234.56789

>>> print('a={}, n={}'.format(a,n)) # idem que print(f'a={a}, n={n}')
a=-1234.56789, n=9876

>>> print('a={:g}'.format(a)) # choisit le format le plus approprie - idem que
 ↪  print(f'a={a:g}')
a=-1234.57

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

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

>>> print('{:.2f}'.format(a)) # fixe le nombre de decimales - idem que print(f'{a:.2f}')
-1234.57

>>> print('{:12.2f}'.format(a)) # precise la longueur totale de la chaine - idem que
 ↪  print(f'{a:12.2f}')
    -1234.57

>>> print('{num:{fill}{width}}'.format(num=123, fill='0', width=6)) # idem que
 ↪  print(f'{123:{0}{6}}')
000123
```

```
>>> print('{:>12.2f}'.format(a)) # justifie a droite - idem que print(f'{a:>12.2f}')
    -1234.57

>>> print('{:<12.2f}'.format(a)) # justifie a gauche -  idem que print(f'{a:<12.2f}')
-1234.57

>>> print('{:^12.2f}'.format(a)) # centre - idem que print(f'{a:^12.2f}')
  -1234.57

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

>>> print('n={1:+.2f} a={0:+.2f}'.format(a,n)) # ordre d'apparition != ordre dans format
n=+9876.00 a=-1234.57
>>> # idem que print(f'n={n:+.2f} a={a:+.2f}')
```

Attention
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")`.

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.

```
x=input("Comment tu t'appelles ? ")
print(f"Tu t'appelle {x}")
```

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

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

1.11. Exercices

Exercice 1.1 (Initiation Linux)

1. **Utiliser l'interface graphique**.
   - ▷ Lancer un programme dont l'icône est sur le bureau (s'il y en a).
   - ▷ Lancer un programme à partir du menu (Gedit, Geany, Firefox, Libre Office Calc).
   - ▷ Gestion des fichiers et arborescence.
2. **Utiliser un terminal/console**.
   - ▷ Ouvrir un terminal. Taper la commande `gedit` et appuyez sur la touche Enter. Que se passe-t-il ? Si on ferme la fenêtre du terminal, que se passe-t-il ?
   - ▷ Lancer d'autre programmes par ligne de commande (*e.g.* `geany`, `firefox`)
   - ▷ Tester les commande ci-dessous (appuyez sur la touche Enter à chaque fin de ligne) :

```
ls
cd Bureau
mkdir PIM11-TP1
ls
cd PIM11-TP1
touch first.py
ls
cd
cd Bureau/PIM11-TP1/
ls
cd ..
ls
whoami
pwd
```

   Utiliser l'**auto-complétion** : pour éviter d'avoir à saisir des noms de fichier ou de dossier en entier, il est possible d'utiliser la **touche tabulation** ⇆. Un appui sur ⇆ va provoquer la recherche de noms de fichiers (ou de répertoires) dont le début correspond à ce qui a déjà été saisi. Si une seule correspondance est trouvée, elle sera affichée. Si plusieurs correspondances sont trouvées, rien ne sera affiché et il faudra un deuxième appui sur ⇆ pour les lister toutes.
   Historique des instructions saisies : pour naviguer dans l'historique des instructions saisies on peut utiliser les raccourcis ↑ et →.
   - ▷ Commandes de base :
     - ▷ Qui suis-je (quel est l'utilisateur courant) ? `whoami`
     - ▷ Où suis-je (quel est le répertoire courant) ? `pwd` : **p**rint **w**orking **d**irectory
     - ▷ Qu'y a-t-il à l'endroit où je suis (quel est le contenu du répertoire courant) ? `ls` : **l**i**s**t contents of current directory
     - ▷ Changer de répertoire courant (a.k.a «se déplacer» dans l'arborescence) : `cd` : **c**hange **d**irectory
     - ▷ Copie un fichier : `cp` : **c**o**p**y
     - ▷ Déplacer un fichier : `mv` : **m**o**v**e
     - ▷ Créer un dossier : `mkdir` : **m**a**k**e **dir**ectory)
     - ▷ Effacer un fichier : `rm` : **r**e**m**ove efface le(s) fichier(s) dont le(s) chemin(s) est (sont) donné(s) en argument. NB ce qui est effacé avec `rm` ne peut pas être récupéré (jamais, never, mai più)
     - ▷ Gestion des droits : `chmod`
     - ▷ Documentation : `man` : **man**ual. Chaque commande (qui se respecte) est documentée ; pour obtenir la documentation complète d'une commande, on peut utiliser la commande man en lui spécifiant en argument le nom de la commande dont on veut obtenir la documentation

Correction
Essayer les différentes commandes

```
man ls
```

Par défaut, `ls` n'affiche pas les fichiers dont les noms commencent par un point. Il faut ajouter l'option `-a` pour les inclure dans l'affichage

```
ls
ls -a
```

L'utilisation du caractère * dans l'argument de `ls` restreint l'affichage aux fichiers/répertoires dont le nom correspond au motif formé par l'argument

```
ls *.py
```

Exercice 1.2 (Mode interactive)

On peut utiliser l'interpréteur Python en mode interactif comme une calculatrice. En effet, si vous y tapez une expression mathématique, cette dernière sera 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).

Dans un Terminal lancer l'interpréteur Python avec la commande `python3` et appuyer sur la touche Enter

On voit apparaître la version de python (3.5.2 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 :

```
42
2*12
2**10
((19.99 * 1.21) - 5) / 4
```

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

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

Correction

```
>>> 42
42
>>> 2*12
24
>>> 2**10
1024
>>> ((19.99 * 1.21) - 5) / 4
4.796975
```

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

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

Exercice 1.3 (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 ; plus besoin de tout réécrire à chaque lancement. Le programme s'écrit dans un fichier texte que l'on sauvegarde avec l'extension `.py`. Le lancement du programme peut se faire à partir d'un terminal par la commande `python3 fichier.py`.

1. Écrire et exécuter un script par ligne de commande :
   1.1. Ouvrir un éditeur de texte pur (par exemple Geany ou Gedit, pas de Word ni Libre Office Writer)
   1.2. Écrire les lignes suivantes

```
a=5
b=6
c=a+b
print("c =",c)
```

   1.3. Sauvegarder le fichier comme `first.py`.
   1.4. Ouvrir un terminal et y écrire `python3 first.py` puis appuyer sur la touche Enter .
2. Utiliser un IDE :
   2.1. Lancer le logiciel Idle3, ouvrir le fichier `first.py` puis appuyer sur la touche F5 (on peut enlever les deux première lignes)

Correction

Dans le terminal on obtient c = 11

☆ Exercice Bonus 1.4 (Rendre un script executable)

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).

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`.

Exercice 1.5 (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}")
```

```
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}")
```

Correction

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

Le troisième donne a=100 b=1 et le quatrième a=100 b=1 .

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

Exercice 1.6 (Devine le résultat – logique)

Quel résultat donnent les codes suivants ?

Cas 1
```
a=15
print(a>5 and a<10)
```

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

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

Cas 4
```
a=15
print(a<5 or a>10)
```

Cas 5
```
a, b, c = 1, 10, 100
print(a<b<c)
```

Cas 6
```
a, b, c = 1, 10, 100
print(a>b>c)
```

Correction

Cas 1 : False , Cas 2 : True , Cas 3 : True , Cas 4 : True , Cas 5 : True , Cas 6 : False

Exercice 1.7 (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.

```
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 <- 14
y=y-1    # y <- 13 (on peut réécrire l'instruction y-=1)
x=x+3*y  # x <- 46 (on peut réécrire l'instruction x+=3*y)
print(x) # on affiche 46
```

affichera

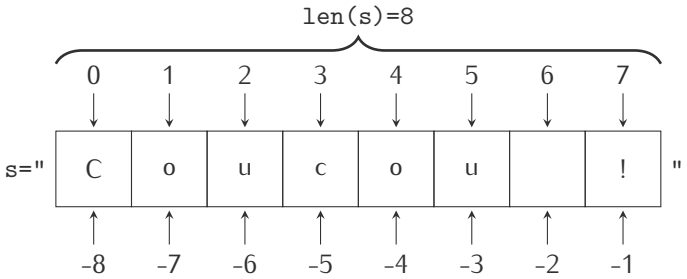
46

Exercice 1.8 (Devine le résultat – `string`)

Quel résultat donne le code suivant ?

```
s='Coucou !'
print(s)
print(s[0])
print(s[0:])
print(s[3:])
print(s[3::1])
print(s[3::2])
print(s[3:0:-1])
print(s[::-1])
print(s*2)
print(s+" TEST")
```

Correction




```
Coucou !
C
Coucou !
cou !
cou !
cu!
cuo
! uocuoC
Coucou !Coucou !
Coucou ! TEST
```

Exercice 1.9 (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[-3:-1]`, `s[:-4]`, `s[-5:]`, `s[0:7:2]`, `s[2:8:3]`.

Correction

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

Exercice 1.10 (Devine le résultat – opérations et conversions de types)

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

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

Correction

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

Exercice 1.11 (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.

Correction

```
h = 'Happy birthday'
t = 'to you'
p = 'Prenom'
print(h+' '+t+'\n'+h+' '+t+'\n'+h+' '+p+'\n'+h+' '+t)
```

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

Exercice 1.12 (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

Lorsqu'on écrit

```
s="Bonjour le monde !"
print(s.count(" "))
```

on obtient comme sortie

3

Exercice 1.13 (String + et *)

Que vaut l'expression suivante en Python

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

Correction

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

En effet :

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

Exercice 1.14 (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 += d # a -> 'sixty'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>> e *= 2 # equivalente a e = 2*e, e -> '11'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'e' is not defined
>>> e += c # e -> '11 > '
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'e' is not defined
>>> a + c + e + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Exercice 1.15 (Calculer l'age)

Affecter la variable `year` avec l'année courante et `birthyear` avec l'année de votre naissance. Afficher la phrase "Je suis né en xxxx donc cette année j'aurai xx ans" où xx sera calculé automatiquement.

Correction

Voici différentes façons d'utiliser la fonction `print` :

```
>>> year = 2019
>>> birthyear = 2000
>>> age = year - birthyear
>>> print('Né en', birthyear, ", j'ai", age, 'ans.')
Né en 2000 , j'ai 19 ans.
>>> print('Né en ' + str(birthyear) + " j'ai " + str(age) + ' ans.')
Né en 2000 j'ai 19 ans.
>>> print("Né en {} j'ai {} ans".format(birthyear, age))
Né en 2000 j'ai 19 ans
>>> print(f"Né en {birthyear} j'ai {age} ans")
Né en 2000 j'ai 19 ans
```

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

Exercice 1.16 (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 !) :

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

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

ainsi on aurai pu écrire

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

Exercice 1.17 (Nombre de chiffre)

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

Correction

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
```

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

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

où $\ell = E(\log_{10}(n))$ est la partie entière de $\log_{10}(n)$ et le nombre de chiffres de n est $\ell + 1$:

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

Pour utiliser les fonctions $\log_{10}$ et E il faut importer le module `math` (voir à la page 135).

Exercice 1.18 (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).

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

Correction

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

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

Remarque : pour enlever les espaces blanc on pourra écrire

```
>>> init="a reveler mon nom mon nom relevera"
>>> s=[x for x in init if x!=" "]
>>> print(s==s[::-1])
True
```

Il s'agit d'une liste définie par compréhension (*cf.* chapitre 5.1).

Exercice 1.19 (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.

Correction

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

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

Exercice 1.20 (Conversion h/m/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
minutesTOT, sec  = divmod(secTOT,60)
hour, minutes = divmod(minutesTOT, 60)
print(secTOT,"secondes correspondent à " , hour, "h ", minutes, "m ", sec, "s")
```

ou

```
secTOT = 12546
hour, r = divmod(secTOT, 60*60)
minutes, sec  = divmod(r,60)
print(secTOT,"secondes correspondent à " , hour, "h ", minutes, "m ", sec, "s")
```

☆ Exercice Bonus 1.21 (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")
print(f"{aM} an(s) et {round(jM-(aM*668.599))} jour(s) martiens")
```

☆ Exercice Bonus 1.22 (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(50)
'        Un EXEMPLE de Chaîne de Caractères        '
```

☆ Exercice Bonus 1.23 (Comptage, recherche et remplacement)

Avec `s = "Monsieur Jack, vous dactylographiez bien mieux que votre ami Wolf"`, que donneront les commandes suivantes ?

```
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'
```

Chapitre 2.

Listes et Tuples

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.

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

>>> liste = ["a","d","m"]
>>> print(liste)
['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.

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

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]`.

Attention

En Python, les éléments d'une liste sont *indexés à partir de* 0 et non de 1.

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

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

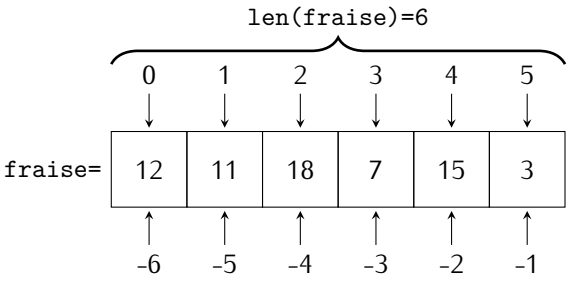
On peut modifier les éléments d'une liste :

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

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]`. Cette opération est connue sous le nom de *slicing* (en anglais). Un petit dessin et quelques exemples permettrons de bien comprendre cette opération fort utile :




```
>>> fraise[2:4]
[18, 7]
>>> fraise[2:]
[18, 7, 15, 3]
>>> fraise[:2]
[12, 11]
>>> fraise[:]
[12, 11, 18, 7, 15, 3]
>>> fraise[2:5]
[18, 7, 15]
>>> fraise[2:6]
[18, 7, 15, 3]
>>> fraise[2:7]
[18, 7, 15, 3]
>>> fraise[2:6:2]
[18, 15]
>>> fraise[-2:-4]
[]
>>> fraise[-4:-2]
[18, 7]
>>> fraise[-1]
3
```

À noter que lorsqu'on utilise des tranches, 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])`. 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
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`
	`max(a)`	renvoi le plus grand élément de la liste `a`
	`min(a)`	renvoi le plus petit élément de la liste `a`
	`sorted(a)`	renvoi une liste triée (avec les mêmes éléments que la liste `a`)
	`reversed(a)`	renvoi une liste avec les mêmes éléments que la liste `a` en ordre inverse

```
>>> a = [2, 37, 20, 83, -79, 21]
>>> print(a)
[2, 37, 20, 83, -79, 21]
>>> del a[1]
>>> print(a)
[2, 20, 83, -79, 21]
>>> print(len(a))
5
>>> a[0] = 21
>>> print(a)
[21, 20, 83, -79, 21]
>>> a[2:4] = [-2,-5,-1978]
>>> print(a)
[21, 20, -2, -5, -1978, 21]
>>> L=sorted(a)
>>> print(L)
[-1978, -5, -2, 20, 21, 21]
>>> B=reversed(a)
>>> print(B)
<list_reverseiterator object at 0x7fafd301e748>
```

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`
	`a.remove(x)`	supprime la première occurrence de l'élément `x` dans la liste `a`
	`a.pop(i)`	supprime l'élément d'indice `i` dans la liste `a` et le renvoie
	`a.index(x)`	renvoie l'indice de la première occurrence de l'élément `x` dans la liste `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 booléen `reverse` permet d'inverser le tri
	`a.reverse()`	modifie la liste `a` en inversant les éléments

Attention : **ces méthodes modifient la liste !**

```
>>> 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)
>>> print(a)
[2, 37, 20, 83, -79, 100, 17, 34, 21]
>>> a.count(21)
1
>>> a.pop(4)
-79
>>> print(a)
[2, 37, 20, 83, 100, 17, 34, 21]
>>> a.index(100)
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]
```

Les opérations * et + sont aussi définies pour les listes comme dans l'exemple suivant :

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

Cependant, l'utilisation de la méthode `append` est plus efficace que l'instruction `a+=[x]`.

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 !

Pour créer une copie `c` de la liste `a` qui soit vraiment indépendante on utilisera la commande `deepcopy` du module `copy` comme dans l'exemple 3 suivant :

Exemple 1

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a
>>> a[0] = 5.0
>>> print(a)
[5.0, 2.0, 3.0]
>>> print(b) # Pb
[5.0, 2.0, 3.0]
```

Exemple 2

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a
>>> b[0] = 5.0
>>> print(a) # Pb!!!
[5.0, 2.0, 3.0]
>>> print(b)
[5.0, 2.0, 3.0]
```

Exemple 3

```
>>> import copy
>>> a = [1.0, 2.0, 3.0]
>>> c = copy.deepcopy(a)
>>> c[0] = 5.0
>>> print(a)
[1.0, 2.0, 3.0]
>>> print(c)
[5.0, 2.0, 3.0]
```

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.

2.2. Les tuples

Pour simplifier, les tuples sont des listes qui ne peuvent pas être modifiées. 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.

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 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 $0, 1, 2, \ldots, n-1$;
- `range(n,m)` renvoi un itérateur parcourant $n, n+1, n+2, \ldots, m-1$;
- `range(n,m,p)` renvoi un itérateur parcourant $n, n+p, n+2p, \ldots, m-1$.

```
>>> 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["nom"] = "Faccanoni"
>>> Mon1Dico["prenom"] = "Gloria"
>>> print(Mon1Dico)
{'nom': 'Faccanoni', 'prenom': 'Gloria'}

>>> 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) ] )
>>> print(Mon4Dico)
{'a': 1, 'b': 2, 'toto': 3}
```

On peut utiliser des tuples comme clé comme lors de l'utilisation de coordonnées :

```
>>> b = {}
>>> b[(3,2)]=12
>>> b[(4,5)]=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.

```
>>> fiche = {"nom":"Engel","prenom":"Olivier"}
>>> print( [cle for cle in fiche.keys()] )
['nom', 'prenom']
>>> print( [valeur for valeur in fiche.values()] )
['Engel', 'Olivier']
>>> 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', 'M12', 'M13', 'I11', 'M14', 'M11'}
{'P111', 'M12', 'I12', 'I11', 'M11'}
{'P111', 'I11', 'C111', 'M11'}
{'P111', 'I11', 'M11', 'M12'}
```

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

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

Nota bene : on ne peut ajouter qu'une valeur immuable (comme une chaîne ou un tuple) à un ensemble. 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)
```

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

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)
```

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

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`. 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)
```

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

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']
```

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

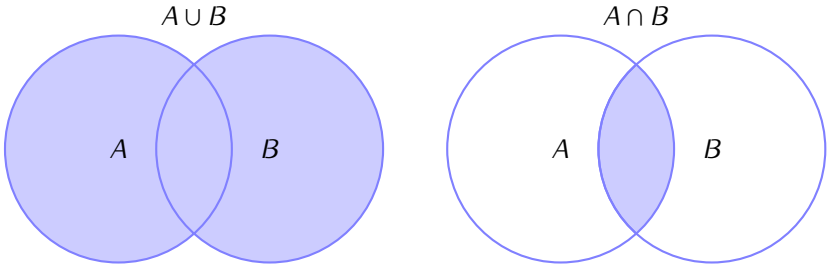
```
L=[1,2,3,4,3,2,3,1,2]
print(L)
L=list(set(L))
print(L)
```

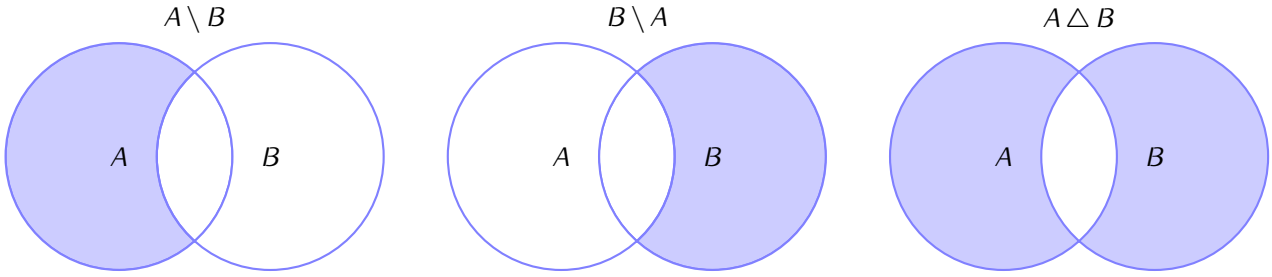
```
[1, 2, 3, 4, 3, 2, 3, 1, 2]
[1, 2, 3, 4]
```

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 $\mathcal{P}(E)$ l'ensemble des parties de E. Soient A et B deux éléments de $\mathcal{P}(E)$. Les quatre éléments $A \cup B$, $A \cap B$, $A \setminus B$, $A \triangle B$ de $\mathcal{P}(E)$ 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)}')
```

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

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

2.6. Exercices

Exercice 2.1 (Devine le résultat)

```
L=[1,2,3,"a","b","toto","zoo","-3.14",-3.14,[8,9,5]]
print(L[2])
print(L[5:7])
print(L[5:])
print(L[9][2])
print(L[2]*2)
print(L[2]+2)
print(L[8]*3)
print(L[7]*3)
print(float(L[7])*3)
print(L[8]+2)
print(L[7]+2)
print(L[7]+"2")
```

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)
5
>>> print(L[8]*3) # L[8] etant un nombre, on multiplie tout simplement
-9.42
>>> print(L[7]*3) # L[7] est un string donc on concatene 3 fois
-3.14-3.14-3.14
>>> print(float(L[7])*3) # float(L[7]) est cette fois-ci un nombre donc on multiplie tout
 ↪  simplement
-9.42
>>> print(L[8]+2) # L[8] etant un nombre, on additionne tout simplement
-1.1400000000000001
>>> print(L[7]+2) # L[7] est un string, on ne peut pas lui concatener un int
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> print(L[7]+"2") # L[7] est un string et "2" aussi, on peut les concatener
-3.142
```

Exercice 2.2 (Listes et sous-listes)

On considère la liste

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

À quelle expression correspond la chaîne `'roi'` ?

Correction

```
>>> L = [ (0 ,1), "3.1415", 2, 3, [0.1, 0.2, "trois"] ]
>>> L[4][2][1:4]
'roi'
```

Exercice 2.3 (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.

Correction

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

Exercice 2.4 (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 (effectif de la valeur / effectif total).

Correction

```
>>> L=[0,10,20,10,20,30,20,30,40,20]
>>> n=20
>>> eff=L.count(n)
>>> print(f"Effectif={eff}, Fréquence={eff/len(L)}")
Effectif=4, Fréquence=0.4
```

Exercice 2.5 (Range)

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

- ▷ $A = [2, 3, 4, 5, 6, 7, 8, 9]$
- ▷ $B = [9, 18, 27, 36, 45, 54, 63, 72, 81, 90]$
- ▷ $C = [2, 4, 6, 8, \ldots, 48, 50]$
- ▷ $D = [1, 3, 5, 7, \ldots, 47, 49]$

Correction

```
>>> A=list(range(2,10))
>>> print(f"A={A}")
A=[2, 3, 4, 5, 6, 7, 8, 9]
>>> B=list(range(9,91,9))
>>> print(f"B={B}")
B=[9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
>>> C=list(range(2,51,2))
>>> print(f"C={C}")
C=[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48,
↪ 50]
>>> D=list(range(1,50,2))
>>> print(f"D={D}")
D=[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47,
↪ 49]
```

Exercice 2.6 (LE piège avec la copie de listes)

Devine le résultat.

1. Modification de la première liste

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

2. Modification de la deuxième liste

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

3. *Shallow copy* (copie superficielle)

```
a=list(range(5))
b=a[:]
print(f"a={a}\nb={b}")
b[0]=99
print(f"a={a}\nb={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
140224971387912
>>> print(id(m))           # on voit qu'ils correspondent bien au même objet en mémoire
140224971387912

>>> l[0] = 'a'    # puisqu'on modifie l, m aussi est modifiée !
>>> 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))
140224971387912
>>> print(id(n)) # n a un id différent de l : il s'agit de 2 objets distincts
140224971387208

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

Exercice 2.7 (Copie de listes de listes)

Devine le résultat.

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

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

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

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

Correction

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

2. *Shallow copy*
```
>>> a=[ [1,2] , [3,4] ]
>>> b=a[:]              # copie superficielle
>>> print(f"a={a}\nb={b}")
a=[[1, 2], [3, 4]]
b=[[1, 2], [3, 4]]
>>> b[0]=99             # affecte juste b
>>> b[1][0]=88          # affecte b et a
>>> print(f"a={a}\nb={b}")
a=[[1, 2], [88, 4]]
b=[99, [88, 4]]
```

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

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

Exercice 2.8 (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))
>>> y = x[:]
>>> z = y
>>> print(x,y,z)
[1, 0, 0] [1, 0, 0] [1, 0, 0]
>>> z[0] = [y[k] for k in x]
>>> print(x,y,z)
[1, 0, 0] [[0, 1, 1], 0, 0] [[0, 1, 1], 0, 0]
>>> x[1:3] = y[0][1:3]
>>> print(x,y,z)
[1, 1, 1] [[0, 1, 1], 0, 0] [[0, 1, 1], 0, 0]
>>> z[len(y[0])-1] = 0
>>> print(x,y,z)
[1, 1, 1] [[0, 1, 1], 0, 0] [[0, 1, 1], 0, 0]
```

☆ Exercice Bonus 2.9 (Chaînes de caractères : `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 2.10 (Défis Turing n°72 – dictionnaires)

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 les carrée des nombres de 10 à 10^5 et vérifier qu'il a au moins 3 chiffres, au plus 9 chiffres, que les derniers trois chiffres sont identiques mais pas celui qui les précède. Il n'est pas nécessaire d'utiliser un dictionnaire mais c'est pratique pour stocker à la fois le nombre et son carré pour pouvoir vérifier en phase de débogage :

```
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
```

☆ Exercice Bonus 2.11 (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 Bonus 2.12 (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] ]

dp = {}
for l in L:
⟶k = len(l)
⟶if k not in dp: dp[k] = []
⟶dp[k].append(l)

print(dp)
```

```
{1: [[1]], 2: [[4, 5]], 3: [[-1, 8, 9], [11, 12, 13]]}
```

☆ Exercice Bonus 2.13 (Dictionnaires : construction d'un histogramme)

Supposons par exemple que nous voulions établir l'histogramme qui représente la fréquence d'utilisation de chacune des lettres de l'alphabet dans un texte donné. L'algorithme permettant de réaliser ce travail est extraordinairement simple si on le construit sur base d'un dictionnaire.

Commençons par créer un dictionnaire vide appelé `lettres`. Ensuite, on remplit ce dictionnaire en utilisant les caractères de l'alphabet en guise de clés. Les valeurs que nous mémoriserons pour chacune de ces clés seront les fréquences des caractères correspondants dans le texte. Afin de calculer celles-ci, nous effectuons un parcours de la chaîne de caractères `texte`. Pour chacun de ces caractères, nous interrogeons le dictionnaire à l'aide de la méthode `get()`, en utilisant le caractère en guise de clé, afin d'y lire la fréquence déjà mémorisée pour ce caractère. Si cette valeur n'existe pas encore, la méthode `get()` doit renvoyer une valeur nulle. Dans tous les cas, nous incrémentons la valeur trouvée, et nous la mémorisons dans le dictionnaire, à l'emplacement qui correspond à la clé (c'est-à-dire au caractère en cours de traitement).

Pour fignoler notre travail, nous pouvons encore souhaiter afficher l'histogramme dans l'ordre alphabétique. Pour ce faire, nous pensons immédiatement à la méthode sort(), mais celle-ci ne peut s'appliquer qu'aux listes. On convertit alors le dictionnaire en une liste de tuples.

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)]
```

Chapitre 3.

Structure conditionnelle

Supposons vouloir calculer la valeur $y = f(x)$ d'un nombre x selon la règle suivante :

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

On a besoin d'une instruction qui opère une disjonction de cas. En Python il s'agit de l'instruction de choix introduite par le mot-clé `if`.

La syntaxe complète 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.

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. 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
```

3. 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
```

4. Dans cette exemple on calcule y selon la règle donnée au debout du chapitre :

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

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

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

```
if a < 0.0:
    print('a is negative')
elif a > 0.0:
    print('a is positive')
else:
    print('a is zero')
```

et on teste le code pour différentes valeurs de a :

```
a=2
if a < 0.0:
    print('a is negative')
elif a > 0.0:
    print('a is positive')
else:
    print('a is zero')
```

```
a is positive
```

```
a=0
if a < 0.0:
    print('a is negative')
elif a > 0.0:
    print('a is positive')
else:
    print('a is zero')
```

```
a is zero
```

```
a=-2
if a < 0.0:
    print('a is negative')
elif a > 0.0:
    print('a is positive')
else:
    print('a is zero')
```

```
a is negative
```

3.1. Exercices

Exercice 3.1 (Devine le résultat)

Quel résultat donnent les codes suivants ?

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=10
if b>10:
    b=a*b
else:
    a=b
c = a+b
print(a,b,c)
```

Cas 6 :

```
a=2
b=13
if b>10:
    b=a*b
else:
    a=b
c = a+b
print(a,b,c)
```

Cas 7 :

```
a=2
b=4
if b>10:
    b=a*b
a=b
c = a+b
print(a,b,c)
```

Cas 8 :

```
a=2
b=10
if b>10:
    b=a*b
a=b
c = a+b
print(a,b,c)
```

Après avoir écrit votre réponse, vérifiez-là avec l'ordinateur.

Correction

Cas 1 : 2 4 6

Cas 2 : 2 10 12

Cas 3 : 26 26 52

Cas 4 : 4 4 8

Cas 5 : 10 10 20

Cas 6 : 2 26 28

Cas 7 : 4 4 8

Cas 8 : 10 10 20

Exercice 3.2 (Blanche Neige)

Soit le code

```
if a<b<c:
    if 2*a<b:
        print("Prof")
    else:
        print("Timide")
    if 2*c<b:
        print("Atchoum")
else:
    if a<b:
        print("Joyeux")
    if a<c:
        print("Simplet")
    elif b<c:
        print("Dormeur")
    else:
        print("Grincheux")
```

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 `Proof` 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. 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
2. Il n'est pas possible d'afficher `Proof` et `Timide`en même temps.
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 ~~$a < b < c$~~ et $a < c$. Qu'est-ce que ça signifie que $a < b < c$ n'est pas satisfaite ? $a < b < c$ équivaut à $a < b$ ET $b < c$ donc si elle n'est pas satisfaite cela signifie que
   ▷ soit $a \geq b$ quel qu'il soit c
   ▷ soit $b \geq c$ quel qu'il soit a
   Nous cherchons donc un triplet a, b, c tel que soit ($a \geq b$ ET $a < c$) soit ($b \geq c$ ET $a < c$) :
   ▷ $a \geq b$ et $a < c$ signifie $b \leq a < c$
   ▷ $b \geq c$ et $a < c$ signifie $a < c \leq b$
   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

```
T=-2
if T<0:
⟶print("SOLIDE")
elif T<=100:
⟶print("LIQUIDE")
else:
    print("GAZEUX")
```

```
SOLIDE
```

Version abrégée :

```
T=-2
print( "SOLIDE"*(T<0) + "LIQUIDE"*(0<=T<=100) + "GAZEUX"*(T>100) )
```

```
SOLIDE
```

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:
⟶print(x)
else:
⟶print(-x)
```

Idée 2 : En réalité il suffit de changer x avec $-x$ si $x < 0$:

```
if x<0:
⟶x=-x
```

Idée 3 : Ou encore, sans utiliser de `if` explicite :

```
x=(-x)*(x<0)+(x)*(x>=0)
```

Voici des exemples :

```
for x in [-3,0,10]:
⟶if x<0:
⟶⟶x=-x
⟶print(x)
```

```
3
0
10
```

Exercice 3.5 (Indice IMC)

Écrire un script qui, connaissant la taille (en mètres) et la masse (en kg) d'un individu, lui renvoie sa masse IMC 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

$$0.3\mathrm{TP} + \max\{\, 0.7\mathrm{CT}; 0.5\mathrm{CT} + 0.2\mathrm{CC} \,\}$$

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

```
if CT>=CC:
⟶print( 0.3*TP+0.7*CT )
else:
⟶print( 0.3*TP+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

Version abrégée :

```
print( (0.3*TP+0.7*CT)*(CT>=CC) + (0.3*TP+0.5*CT+0.2*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. $a = 1$, $b = 2$, $c = 5^{1/2}$, (quelconque, rectangle)
6. $a = 1$, $b = 1$, $c = 2^{1/2}$, (isocèle rectangle)

Nota bene : au lieu d'écrire `x==y` on utilisera `abs(x-y)<1.e-10`

Correction

Attention à l'ordre dans lequel on écrit la condition : d'abord la condition la plus restrictive (être équilatéral) puis celle moins restrictive (être isocèle). Si on inverse l'ordre, on n'obtiendrait jamais de triangle équilatéral car, étant aussi isocèle, la première condition serait satisfaite et on n'entrerai jamais dans la deuxième.

```
if a<=0 or b<=0 or c<=0:
⟶print("erreur de saisie")
⟶continue
elif a==b and b==c : # mieux abs(a-b)<1.e-10 and abs(b-c)<1.e-10
⟶print("équilatéral")
elif a==b or a==c or b==c :  # mieux abs(a-b)<1.e-10 or abs(b-c)<1.e-10 or abs(a-c)<1.e-10
⟶print("isocèle")
```

```
else :
    print("quelconque")
# Pour ne pas tester 3 conditions, on ordonne a,b,c ainsi,
# si le triangle est rectangle, "c" est l'hypothénuse
[a,b,c].sort()
if abs(c**2-a**2-b**2)<1.e-10:
    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,
- ▷ une seule valeur si la racine est unique,
- ▷ un tuple vide si les racines sont complexes conjuguées

Tester le script dans les trois 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. $a = 0$, $b = 1$, $c = 2$

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

```
if a==0:
    racines=-c/b
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)
```

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

☆ 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 = valide[random.randint(0,2)]
```

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 utente not in valide:
    print("input incorrect")
else:
    cpu = valide[random.randint(0,2)]
    print (f"Choix cpu: {cpu}")
    if cpu == user:
        print ("Égalité")
    elif cpu == "pierre":
        print( "Gagné") if user == "feuille" else "Perdu"
    elif cpu == "ciseaux":
        print ("Gagné") if user == "pierre" else "Perdu"
    else:
        print ("Gagné") if user =="ciseaux" else "Perdu"
```

Chapitre 4.

Structures itératives

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`

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 ou une chaîne de caractères). Tant que `target` appartient à `sequence`, on exécute les instructions `instruction_i`.

Quelques exemples.

▷ Avec `range` :

```
for n in range(5):
⟶print(n)
```

produit

```
0
1
2
3
4
```

▷ On boucle sur les éléments d'une liste :

```
L=["fraises", "cerises", "prunes"]
for truc in L:
⟶print(truc)
```

produit

```
fraises
cerises
prunes
```

▷ On boucle sur une chaîne de caractères :

```
L="Minnie"
for c in L:
⟶print(c)
```

produit

```
M
i
n
n
i
e
```

▷ Pour accéder en même temps à la position et au contenu on utilisera `enumerate(liste)`.

```
L=["fraises", "cerises", "prunes"]
for indice,truc in enumerate(L):
    print(truc, indice)
```

produit

```
fraises 0
cerises 1
prunes 2
```

▷ Voici une autre façon de créer la liste $\left[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right]$ avec un itérateur généré par la fonction `range` :

```
nMax = 5
a = [] # Create empty list
for n in range(1,nMax):
    a.append(1.0/n) # Append element to list
print(a)
```

```
[1.0, 0.5, 0.3333333333333333, 0.25]
```

▷ 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 [10,20,30,40,50]:
    for y in [3,7]:
        print(x+y)
```

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
53
57
```

4.2. Boucle `while` : répétition conditionnelle

`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`.

Voici un exemple pour afficher un compte à rebours : tant que la condition $n \geq 0$ est vraie, on diminue n de 1. La dernière valeur affichée est $n = 0$ car ensuite $n = -1$ et la condition $n \geq 0$ devient fausse donc la boucle s'arrête.

```
n=10
while n>=0:
    print(n)
    n-=1
print("Go!")
```

```
10
9
8
```

```
7
6
5
4
3
2
1
0
Go!
```

⚠ **Attention**
Si la condition ne devient jamais fausse, le bloc d'instructions est répété indéfiniment et le programme ne se termine pas.

Dans l'exemple suivant 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=n//2
print(n)
```

```
21
```

Voici un exemple pour créer la liste $\left[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right]$:

```
nMax = 5
n = 1
a = [] # Create empty list
while n<nMax:
⟶a.append(1/n) # Append element to list
⟶n += 1
print(a)
```

```
[1.0, 0.5, 0.3333333333333333, 0.25]
```

Dans l'exemple suivant on calcul 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('En effet n(n+1)/2=',n*(n+1)/2)
```

```
5050
En effet n(n+1)/2= 5050.0
```

Voyons un autre exemple classique de ce qu'on appelle une recherche de seuil, où cette fois ci on recherche à partir de quel valeur de n la somme $1 + 2 + 3 + \cdots + n$ dépasse un million :

```
n = 0
somme = 0
while somme < 1000000 :
    n += 1
    somme += n
print(n)
```

```
1414
```

4.3. ⭐ Interrompre une boucle

L'instruction `break` sort de la plus petite boucle `for` ou `while` englobante.

```
for i in [1,2,3,4,5]:
    if i==4:
        print("J'ai trouvé")
        break
    print(i)
```

```
1
2
3
J'ai trouvé
```

```
for lettre in 'AbcDefGhj':
    if lettre=='D':
        break
    print(lettre)
```

```
A
b
c
```

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)
```

```
-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)
print("Boucle terminée")
```

```
1
3
5
Boucle terminée
```

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`.

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

Exercice Bonus 4.2 (Pydéfi – 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://callicode.fr/pydefis/Algorithme/txt

Exercice 4.3 (Triangles)

Créer des scripts qui dessinent des triangles comme les suivants :

```
x           xxxxxxxxx          x          xxxxxxxxx
xx          xxxxxxxx           xx          xxxxxxxx
xxx         xxxxxxx            xxx          xxxxxxx
xxxx        xxxxxx             xxxx          xxxxxx
xxxxx       xxxxx              xxxxx          xxxxx
xxxxxx      xxxx               xxxxxx          xxxx
xxxxxxx     xxx                xxxxxxx          xxx
xxxxxxxx    xx                 xxxxxxxx          xx
xxxxxxxxx   x                  xxxxxxxxx          x
```

Correction

```
for n in range(10):
    print('x'*n)
for n in range(9,0,-1):
    print('x'*n)
```

```
for n in range(10):
    print(' '*(9-n)+'x'*n)
for n in range(9,0,-1):
    print(' '*(9-n)+'x'*n)
```

Exercice 4.4 (Nombre de voyelles)

Pour le texte donné, créer un programme qui affiche le nombre de voyelle.

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean at accumsan nisl, ac aliquet tellus. Sed maximus leo lacus, nec pulvinar purus maximus vel. Morbi sagittis suscipit risus, sed luctus metus bibendum vitae. Sed ac odio dignissim, efficitur ipsum eu, imperdiet ante. Sed eu lobortis nulla, placerat fermentum purus. Cras sollicitudin metus et imperdiet condimentum. Nulla a sapien sollicitudin nunc tincidunt interdum. Praesent elementum dolor id lacus lacinia, eu viverra arcu molestie. In nec neque ac ligula posuere gravida. Nulla maximus augue in consequat viverra. Integer euismod nibh a elit ullamcorper venenatis. In egestas, urna quis congue aliquam, risus lacus mollis nibh, sed venenatis dui lorem sed eros. Aliquam erat volutpat. Sed sagittis quam sed vestibulum ultricies. Cras sit amet efficitur nunc. Pellentesque mattis fermentum dui, finibus sagittis lacus sollicitudin quis. Praesent luctus pharetra nunc, vitae commodo ante laoreet sed. Sed eu lectus lectus. Nam luctus nisi quis porta fringilla. Quisque a tempus lectus. Cras at mauris tincidunt, volutpat eros vel, venenatis urna. Integer et tellus ut dui vulputate interdum ut id nunc.

Correction

```
texte =
 ↪  "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean at accumsan nisl, ac aliquet tell
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(v)
```

```
409
```

Exercice 4.5 (Table de multiplication)

Afficher les tables de multiplication entre 1 et 10. Voici un exemple de ligne à afficher : `7 x 9 = 63` On utilisera une commande d'affichage du style `print(a,"x",b,"=",a*b)`.

Correction

```
for a in range(1,11):
    for b in range(1,11):
        print(f'{a:2d}x{b:<2d}={a*b:3d}')
        # print('{:2d}x{:<2d}={:3d}'.format(a,b,a*b))
```

```
 1x1 =  1          2x1 =  2          3x1 =  3          4x1 =  4          5x1 =  5
 1x2 =  2          2x2 =  4          3x2 =  6          4x2 =  8          5x2 = 10
 1x3 =  3          2x3 =  6          3x3 =  9          4x3 = 12          5x3 = 15
 1x4 =  4          2x4 =  8          3x4 = 12          4x4 = 16          5x4 = 20
 1x5 =  5          2x5 = 10          3x5 = 15          4x5 = 20          5x5 = 25
 1x6 =  6          2x6 = 12          3x6 = 18          4x6 = 24          5x6 = 30
 1x7 =  7          2x7 = 14          3x7 = 21          4x7 = 28          5x7 = 35
 1x8 =  8          2x8 = 16          3x8 = 24          4x8 = 32          5x8 = 40
 1x9 =  9          2x9 = 18          3x9 = 27          4x9 = 36          5x9 = 45
 1x10= 10          2x10= 20          3x10= 30          4x10= 40          5x10= 50
```

```
6x1 =  6        7x1 =  7        8x1 =  8        9x1 =  9        10x1 = 10
6x2 = 12        7x2 = 14        8x2 = 16        9x2 = 18        10x2 = 20
6x3 = 18        7x3 = 21        8x3 = 24        9x3 = 27        10x3 = 30
6x4 = 24        7x4 = 28        8x4 = 32        9x4 = 36        10x4 = 40
6x5 = 30        7x5 = 35        8x5 = 40        9x5 = 45        10x5 = 50
6x6 = 36        7x6 = 42        8x6 = 48        9x6 = 54        10x6 = 60
6x7 = 42        7x7 = 49        8x7 = 56        9x7 = 63        10x7 = 70
6x8 = 48        7x8 = 56        8x8 = 64        9x8 = 72        10x8 = 80
6x9 = 54        7x9 = 63        8x9 = 72        9x9 = 81        10x9 = 90
6x10= 60        7x10= 70        8x10= 80        9x10= 90        10x10=100
```

Exercice 4.6 (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         roi pique       valet coeur     9 carreau       7 trefle        roi trefle
8 pique         as pique        reine coeur     10 carreau      8 trefle        as trefle
9 pique         7 coeur         roi coeur       valet carreau   9 trefle
10 pique        8 coeur         as coeur        reine carreau   10 trefle
valet pique     9 coeur         7 carreau       roi carreau     valet trefle
reine pique     10 coeur        8 carreau       as carreau      reine trefle
```

Exercice 4.7 (Devine le résultat)

```
for i in range(10):
⟶print("Je ne dois pas poser une question sans lever la main")
```

et

```
i = 0
while i < 10:
⟶print("Je ne dois pas poser une question sans lever la main")
⟶i = i +1
```

Correction

```
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
```

Exercice 4.8 (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

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 4.9 (Plus petit n)

Écrire un script qui affiche le plus petit entier n tel que $(n+1)*(n+3)$ dépasse 12345.

Écrire un script qui affiche le plus petit entier n tel que $4+5+6+\cdots+n$ dépasse 12345.

Écrire un script qui affiche le plus petit entier n tel que $1^2+2^2+3^2+\cdots+n^2$ dépasse 12345.

Correction

```
n=0
while (n+1)*(n+3)<12345:
    n+=1
print(n)
print(f'En effet, si n={n} alors (n+1)*(n+3)={(n+1)*(n+3)}')
print(f'tandis que, si n={n-1} alors (n+1)*(n+3)={(n)*(n+2)}')
```

```
110
En effet, si n=110 alors (n+1)*(n+3)=12543
tandis que, si n=109 alors (n+1)*(n+3)=12320
```

```
n = 4
somme = n
while somme < 12345 :
    n += 1
    somme += n
print(n)
print(f'En effet, si n={n} alors somme={somme}')
print(f'tandis que, si n={n-1} alors somme={somme-n}')
```

```
157
En effet, si n=157 alors somme=12397
tandis que, si n=156 alors somme=12240
```

```
n = 1
somme = n**2
while somme < 12345 :
    n += 1
    somme += n**2
print(n)
print(f'En effet, si n={n} alors somme={somme}')
print(f'tandis que, si n={n-1} alors somme={somme-n**2}')
```

```
33
En effet, si n=33 alors somme=12529
tandis que, si n=32 alors somme=11440
```

Exercice 4.10 (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 $u_n < 10^{-4}$?

Correction

$u_n = \left(\left(\frac{7}{10}\right)^3\right)^n = \left(\frac{343}{1000}\right)^n$. Il s'agit d'une suite géométrique de raison $0 < q < 1$: elle est donc décroissante. On a

$$u_n < 10^{-4} \iff \left(\frac{343}{1000}\right)^n < 10^{-4} \iff \log_{10}\left(\frac{343}{1000}\right)^n < -4$$
$$\iff n > -\frac{4}{\log_{10}\left(\frac{343}{1000}\right)} = -\frac{4}{\log_{10}(343) - \log_{10}(10^3)} = \frac{4}{3 - \log_{10}(343)} \simeq \frac{4}{0.5} = 8.$$

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

```
n=0
u = (0.7)**(3*n)
while u>=1.e-4:
    n+=1
    u = (0.7)**(3*n)
print(f"u_{n}={u}")
```

```
u_9=6.571236236353417e-05
```

Pour bien voir ce qu'on fait :

```
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
```

Exercice 4.11 (Plus petit diviseur)

Soit $n \in \mathbb{N}$ 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).

```
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 4.12 (Puissance)

Soit $n \in \mathbb{N}$ donné et cherchons la première puissance de 2 plus grande que n en utilisant une boucle `while`.

Correction

Méthode 1 :

```
n = 100
p = 0
while 2**p<=n:
⟶p+=1 # p=p+1
print(2**p)
```

```
128
```

Méthode 2 :

```
n = 100
p = 1
while p<=n:
⟶p*=2 # p=p*2
print(p)
```

```
128
```

Exercice 4.13 (Suites par récurrence)

Soit $(u_n)_{n\in\mathbb{N}}$ une suite définie par récurrence. Écrire une script qui affiche $u_0, \dots u_{10}$ dans les cas suivants en utilisant une boucle `while` :

$$\begin{cases} u_0 = 1, \\ u_{n+1} = 2u_n + 1; \end{cases} \qquad \begin{cases} u_0 = -1, \\ u_{n+1} = -u_n + 5; \end{cases} \qquad \begin{cases} u_0 = 0, \\ u_1 = 1, \\ u_{n+2} = u_{n+1} + u_n. \end{cases}$$

Correction

```
n=0
u=1
print("u_{}={}".format(n,u))
while n<10:
    n+=1
    u=2*u+1
    print("u_{}={}".format(n,u))
```

```
u_0=1
u_1=3
u_2=7
u_3=15
u_4=31
u_5=63
u_6=127
u_7=255
u_8=511
u_9=1023
u_10=2047
```

```
n=0
u=-1
print("u_{}={}".format(n,u))
while n<10:
    n+=1
    u=-u+5
    print("u_{}={}".format(n,u))
```

```
u_0=-1
u_1=6
u_2=-1
u_3=6
u_4=-1
u_5=6
u_6=-1
u_7=6
u_8=-1
u_9=6
u_10=-1
```

```
n=1
u=[0,1]
while n<10:
    n+=1
    u.append(u[-1]+u[-2])
for ui in u:
    print("u_{}={}".format(u.index(ui),ui))
```

```
u_0=0
u_1=1
u_1=1
u_3=2
u_4=3
u_5=5
u_6=8
u_7=13
u_8=21
u_9=34
u_10=55
```

Exercice 4.14 (Suites et affectations parallèles)

Calculer u_5 et v_5 avec $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ deux suites définies par

$$\begin{cases} u_0 = 1, \\ v_0 = -1, \\ u_{n+1} = 3v_n + 1, \\ v_{n+1} = u_n^2. \end{cases}$$

.

Correction

Avec deux affectations parallèles

```
n,u,v=0,1,-1
print("u_{}={}, v_{}={}".format(n,u,n,v))
for n in range(1,5):
    (u,v)=(3*v+1,u*u)
    print("u_{}={}, v_{}={}".format(n,u,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
```

ou, en introduisant la variable `uold` :

```
n,u,v=0,1,-1
print("u_{}={}, v_{}={}".format(n,u,n,v))
for n in range(1,5):
    uold = u
    u=3*v+1
    v=uold*uold
    print("u_{}={}, v_{}={}".format(n,u,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

```
n,u,v=0,1,-1
print("u_{}={}, v_{}={}".format(n,u,n,v))
for n in range(1,5):
    u=3*v+1
    v=u*u
    print("u_{}={}, v_{}={}".format(n,u,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

$$\begin{cases} u_0 = 1, \\ v_0 = -1, \\ u_{n+1} = 3v_n + 1, \\ v_{n+1} = u_{\mathrm{n+1}}^2. \end{cases}$$

Exercice 4.15 (Suites en parallèles)

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

$$\begin{cases} u_0 = v_0 = 1, \\ u_{n+1} = u_n + v_n, \\ v_{n+1} = 2u_n - v_n. \end{cases}$$

Correction

Calcule explicite du 100-ème terme :

```
u,v = 1,1
for n in range(100):
    (u,v)=(u+v,2*u-v)
print("u_100 =",u,"\nv_100 =",v)
```

```
u_100 = 717897987691852588770249
v_100 = 717897987691852588770249
```

Exercice Bonus 4.16 (Pydéfi – 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.

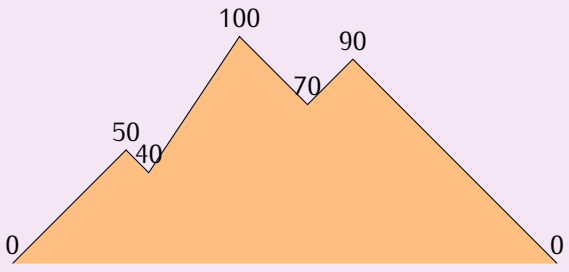
Si l'Hydre a 6 têtes, Hercule doit donc donner 7 coups d'épée pour la vaincre.

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://callicode.fr/pydefis/Herculito02Hydre/txt

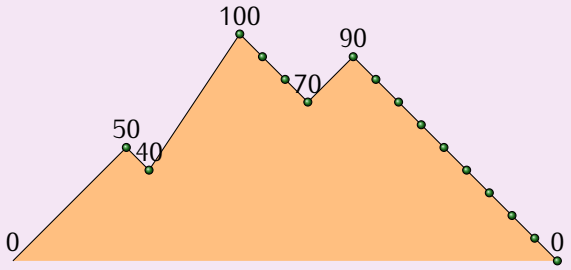
Exercice Bonus 4.17 (Pydéfi – Le sanglier d'Érymanthe)

Histoire : 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://callicode.fr/pydefis/Herculito04Sanglier/txt

Exercice Bonus 4.18 (Pydéfi – 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 :

- ▷ commencez par relever le numéro de série (il est indiqué en entrée du problème sur cette page)
- ▷ coupez le numéro de série en 2. Les trois premiers chiffres forment le nombre U et les 3 derniers le nombre N
- ▷ répétez N fois les opérations 4 et 5 suivantes en partant du nombre U
- ▷ multipliez ce nombre par 13
- ▷ ne conservez que les 3 derniers chiffres.

Une fois cet algorithme terminé, le nombre obtenu est le numéro du fil à couper.

Par exemple, si le numéro de série avait été 317010, il aurait fallu couper le fil 133.

Indiquez à Black Widow le numéro du fil à couper pour valider le défi si le numéro de série est 449149.

Source : https://callicode.fr/pydefis/Desamorcage01/txt

Exercice Bonus 4.19 (Pydéfi – 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://callicode.fr/pydefis/Melange/txt

Exercice Bonus 4.20 (Pydéfi – 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.

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.

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.

L'entrée du problème est constituée de 2 valeurs : u_1 (premier terme de la suite) et n. Que vaut u_n ?

Source : https://callicode.fr/pydefis/SuiteTordue/txt

Exercice Bonus 4.21 (Pydéfi – 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://callicode.fr/pydefis/Desamorcage03/txt

Exercice Bonus 4.22 (Pydéfi – SW VII : Les noms des Ewoks I)

Les Ewoks ont des noms amusants : Hexprakmo, Ashkeethur, Lograyshu... Il est courant que leur nom contienne au moins un "a", comme dans les 3 exemples qui précèdent.

Vous avez en entrée une liste de noms Ewoks (https://callicode.fr/pydefis/EwoksSansA/input). Combien ne contiennent pas de "a" **peu importe la casse** ?

Source : https://callicode.fr/pydefis/EwoksSansA/txt

Exercice Bonus 4.23 (Pydéfi – SW VII : Les noms des Ewoks II)

Les Ewoks ont toujours des noms amusants : Ashbimo, Hexjunprak, Hexphambi... Certains contiennent exactement deux fois plus de consonnes que de voyelles, comme c'est le cas pour Hexphambi (3 voyelles et 6 consonnes).

Vous avez en entrée une liste de noms Ewoks (https://callicode.fr/pydefis/EwoksVoyelle/input) Combien contiennent exactement deux fois plus de consonnes que de voyelles (on comptera le y comme une voyelle) ?

Source : https://callicode.fr/pydefis/EwoksVoyelle/txt

Exercice 4.24 (Devine le résultat)

Quel résultat donne le code suivant ? Après avoir écrit la réponse, vérifiez-là 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 4.25 (Happy Birthday)

Soit les chaînes de caractères suivantes :

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

Imprimer la chanson *Happy birthday*.

Correction

```
h = 'Happy birthday'
t = 'to you'
p = 'Prenom'
for i in range(4):
⟶if i==2:
⟶⟶print(h+' '+p)
⟶else:
⟶⟶print(h+' '+t)
```

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

Exercice 4.26 (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
#print(f'Le jour {jour} mon cadeau est de {cadeau_du_jour}€ ainsi la somme totale est
 ↪ {cadeau_total:.2f}€.')
while (True):
    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}€.')
    if cadeau_total >= 1.e6:
        break
print(f'Le jour {jour} mon cadeau est de {cadeau_du_jour}€ ainsi la somme totale est {cadeau_total:.2
```

```
Le jour 27 mon cadeau est de 671088.64€ ainsi la somme totale est 1342177.27€.
```

Exercice 4.27 (Affichage)

Pour $n \in \mathbb{N}$ donné, afficher $1 + 2 + \cdots + n = N$ 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 4.28 (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

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. Elle doit son nom à Leonardo Fibonacci, un mathématicien italien du XIII^{e} siècle qui, dans un problème récréatif posé dans un de ses ouvrages, le *Liber Abaci*, décrit la croissance d'une population de lapins :

> «Un homme met un couple de lapins dans un lieu isolé de tous les côtés par un mur. Combien de couples obtient-on en un an si chaque couple engendre tous les mois un nouveau couple à compter du troisième mois de son existence ?»

Le problème de Fibonacci est à l'origine de la suite dont le n-ième terme correspond au nombre de couples de lapins au n-ème mois. Dans cette population (idéale), on suppose que :

- ▷ au (début du) premier mois, il y a juste une paire de lapereaux ;
- ▷ les lapereaux ne procréent qu'à partir du (début du) troisième mois ;
- ▷ chaque (début de) mois, toute paire susceptible de procréer engendre effectivement une nouvelle paire de lapereaux ;
- ▷ les lapins ne meurent jamais (donc la suite de Fibonacci est strictement croissante).

Notons F_n le nombre de couples de lapins au début du mois n. Jusqu'à la fin du deuxième mois, la population se limite à un couple (ce qu'on note $F_1 = F_2 = 1$). Dès le début du troisième mois, le couple de lapins a deux mois et il engendre un autre couple de lapins ; on note alors $F_3 = 2$. Plaçons-nous maintenant au mois n et cherchons à exprimer ce qu'il en sera deux mois plus tard, soit au mois $n+2$: F_{n+2} désigne la somme des couples de lapins au mois $n+1$ et des couples nouvellement engendrés. Or, n'engendrent au mois $(n+2)$ que les couples pubères, c'est-à-dire ceux qui existent deux mois auparavant. On a donc, pour tout entier n strictement positif, $F_{n+2} = F_{n+1} + F_n$. On obtient ainsi la forme récurrente de la suite de Fibonacci : 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 impaire 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

La suite peut aussi se réécrire comme le produit matriciel

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}$$

En exploitant cette relation matricielle on obtient par récurrence

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix} = \cdots = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

Il est même possible de calculer F_n en exploitant une expression fonctionnelle de la suite, c'est-à-dire une expression telle que le calcul de F_n pour une valeur de n donnée ne présuppose la connaissance d'aucune autre valeur de n. En effet, comme la suite de Fibonacci est linéaire d'ordre deux, on peut écrire son équation caractéristique. On obtient une équation du second degré $x^2 - x - 1 = 0$ qui a pour solutions $x_1 = \varphi = \frac{1+\sqrt{5}}{2}$ (le nombre d'or) et $x_2 = 1 - \varphi = \frac{1-\sqrt{5}}{2}$. Il en résulte que $F_n = \alpha\varphi^n + \beta(1-\varphi)^n$ où α et β sont deux constantes à déterminer à partir de F_0 et F_1. On a $\alpha + \beta = 0$ et $(\alpha - \beta)\varphi + \beta = 1$ ce qui donne $\alpha = -\beta = 1/\sqrt{5}$. On trouve alors l'expression générale de la suite de Fibonacci (appelée formule de Binet) :

$$F_n = \frac{1}{\sqrt{5}}\left(\varphi^n - (1-\varphi)^n\right).$$

Si on calcule la limite du rapport de deux nombres consécutifs de la suite de Fibonacci on trouve le nombre d'or :

$$\frac{F_{n+1}}{F_n} = \frac{\varphi^{n+1} - (1-\varphi)^{n+1}}{\varphi^n - (1-\varphi)^n} = \varphi \frac{1 - \left(\frac{1-\varphi}{\varphi}\right)^{n+1}}{1 - \left(\frac{1-\varphi}{\varphi}\right)^{n}} \xrightarrow[n\to\infty]{} \varphi$$

car $1 < \varphi < 2$ et donc $-1 < \frac{1-\varphi}{\varphi} < 1$.

Exercice Bonus 4.29 (Pydéfi – Fibonacci)

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent

$$\begin{cases} F_0 = 0, \\ F_1 = 1, \\ F_{n+2} = F_{n+1} + F_n \quad \text{pour } n = 1, 2, \ldots \end{cases}$$

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://callicode.fr/pydefis/FiboChiffres/txt

Exercice Bonus 4.30 (Pydéfi – 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://callicode.fr/pydefis/AlgoMat/txt

Exercice 4.31 (Maximum d'une liste de nombres)

Soit une liste de nombres. Trouver le plus grand élément de cette liste **sans utiliser la fonction prédéfinie** `max`.

Correction

```
L=[10,5,15,-2,17,-22]
mymax=L[0]
for ell in L:
⟶if ell>mymax:
⟶⟶mymax=ell
print(f"L={L}, max(L)={max(L)}, mymax(L)={mymax}")
```

```
L=[10, 5, 15, -2, 17, -22], max(L)=17, mymax(L)=17
```

☆ Exercice Bonus 4.32 (Le nombre mystère)

Écrire un script où l'ordinateur choisit un nombre mystère entre 0 et 99 (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 $[0; 99]$ 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}")
```

☆ Exercice Bonus 4.33 (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 :( ")
```

☆ Exercice Bonus 4.34 (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 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 $p = 3600$).

```
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 4.35 (Défi Turing n°11 - nombre miroir)

On appellera "miroir d'un nombre n" le nombre n écrit de droite à gauche. Par exemple, `miroir(7423) = 3247`. Quel est le plus grand nombre inférieur à 10 millions ayant la propriété : `miroir(n) = 4n` ?

Correction

```
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 4.36 (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 1.18). 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

```
for n in range (10**7):
⟶sc=str(n*n)
⟶if len(sc)%2==0 and sc[::-1]==sc:
⟶⟶print(n,n*n)
```

```
836 698896
798644 637832238736
```

☆ Exercice Bonus 4.37 (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

```
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 Bonus 4.38 (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 4.39 (Défi Turing n°70 – Permutation circulaire)

Prenons le nombre 102564. En faisant passer le dernier chiffre complètement à gauche, on obtient un multiple (différent du nombre de départ). En effet, $410256 = 102564 \times 4$.

Additionner tous les nombres de 6 chiffres ayant cette propriété.

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(L)
print(sum(L))
```

```
[102564, 128205, 142857, 153846, 179487, 205128, 230769]
1142856
```

☆ Exercice Bonus 4.40 (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)
print("Q1 :", Q1)
ecarts=[l[i]-l[i-1] for i in range(1,len(l))]
Q2=max(ecarts)
print("Q2 :", Q2)
print("Produit :", Q1*Q2)
```

```
Q1 : 1243
Q2 : 105
Produit : 130515
```

Exercice Bonus 4.41 (Pydéfi – La suite Q de Hofstadter)

La Q-Suite est définie ainsi :

$$\begin{cases} Q_1 = 1, \\ Q_2 = 1, \\ Q_n = Q_{n-Q_{n-1}} + Q_{n-Q_{n-2}} \quad \text{pour } n > 2. \end{cases}$$

Que vaut $\sum_{i=2313}^{i=2375} Q_i$?

Source : https://callicode.fr/pydefis/QSuite/txt

Exercice Bonus 4.42 (Pydéfi – 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://callicode.fr/pydefis/Escargot/txt

Exercice Bonus 4.43 (Pydéfi – 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://callicode.fr/pydefis/MiroirAjout/txt

Exercice Bonus 4.44 (Pydéfi – 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

$$371 \to 21 \to 2$$
$$372 \to 42 \to 8$$
$$373 \to 63 \to 18 \to 8$$
$$374 \to 84 \to 32 \to 6$$
$$375 \to 105 \to 0$$
$$376 \to 126 \to 12 \to 2$$
$$377 \to 147 \to 28 \to 16 \to 6$$
$$378 \to 168 \to 48 \to 32 \to 6$$
$$379 \to 189 \to 72 \to 14 \to 4$$

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://callicode.fr/pydefis/Persistance/txt

Exercice Bonus 4.45 (Pydéfi – 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://callicode.fr/pydefis/TocBoum/txt

Exercice Bonus 4.46 (Pydéfi – 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://callicode.fr/pydefis/Herculito08Juments/txt

Exercice Bonus 4.47 (Pydéfi – 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://callicode.fr/pydefis/Palindromes/txt

☆ Exercice Bonus 4.48 (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

```
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(x):
    m=x[0]
    j=x[1]
    if m==3:
        print(f"Le dimanche de Pâques est le {j+1} mars")
    elif m==4:
        print(f"Le dimanche de Pâques est le {j+1} avril")
    else:
        print("Error")

# TEST
#afficher(paques(2020))

# DEFIS
dico={3:0,4:0}
for a in range(2001,9999+1):
    m,_=paques(a)
    dico[m]+=1

print(dico)
```

3 : 1880, 4 : 6119

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.

Les expressions en compréhension sont un substitut complet aux boucles, à la fonction lambda ainsi qu'aux fonctions `map()`, `filter()` et `reduce()`.

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 :

$$\{\ f(x)\ \mid\ x\ \in\ E\ \}$$
$$[\ \texttt{f(x)}\ \texttt{for}\ \texttt{x}\ \texttt{in}\ \texttt{E}\ ]$$

Syntaxe :

```
[function(item) for item in list if condition(item)]
```

Le code suivant avec une boucle

```
L=[]
for i in range(5):
    L.append(i**2)
print(L)
```

affiche [0, 1, 4, 9, 16]

Le code suivant avec une liste en compréhension

```
L=[i**2 for i in range(5)]
print(L)
```

affiche la même chose [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, enfin on génère une autre liste qui n'a pas la même longueur que `E` et on multiplie les éléments terme à terme :

```
>>> 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]
```

```
>>> liste2 = list(range(3))
>>> [x*y for x in E for y in liste2]
[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 :

```
>>> E = [1, 2, 3, 4, 5, 6, 7] # E=list(range(1,8))

>>> [x**2 for x in E] # { x^2 | x ∈ E }
[1, 4, 9, 16, 25, 36, 49]

>>> [x for x in E if x%2 == 0] # pairs
[2, 4, 6]

>>> [x**2 for x in E if x**2%2 == 0] # carres pairs
[4, 16, 36]

>>> # idem que
>>> L=[]
>>> for x in E:
...     if x**2%2 == 0:
...         L.append(x**2)
...
>>> print(L)
[4, 16, 36]

>>> # idem que
>>> [x for x in [a**2 for a in E] if x%2 == 0]
[4, 16, 36]

>>> # idem que
>>> L=[]
>>> for a in E:
...     x=a**2
...     if x%2 == 0:
...         L.append(x)
...
>>> print(L)
[4, 16, 36]
```

▷ Une autre façon de créer la liste $\left[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right]$ 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],[7,8,9]]
>>> M_transpose = [ [row[i] for row in M] for i in range(len(M)) ]
>>> print(M)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> print(M_transpose)
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

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']}
{'PYTHON', 'SQL'}
```

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'}
```

5.4. Exercices

Exercice 5.1 (Somme des carrés)

Soit L une liste de nombres. Calculer la somme des carrés des éléments de L. Par exemple, si L=[0,1,2], on doit obtenir 5.

Correction

```
>>> n=3
>>> s=sum([x**2 for x in range(n)])
>>> s
5
```

Exercice 5.2 (Conversion)

Soit la liste de chaîne de caractères ["5", "10", "15"]. Obtenir la liste d'entier [5, 10, 15].

Correction

```
>>> S=["5", "10", "15"]
>>> S
['5', '10', '15']
>>> L=[int(x) for x in S]
>>> L
[5, 10, 15]
```

Exercice 5.3 (Chaînes de caractères)

Soit la chaîne de caractères "Ciao". Obtenir la liste ["C", "i", "a", "o"].

Correction

```
>>> s="Ciao"
>>> L=[c for c in s]
>>> L
['C', 'i', 'a', 'o']
```

ou, plus simplement

```
>>> s="Ciao"
>>> L=list(s)
>>> L
['C', 'i', 'a', 'o']
```

Exercice 5.4 (Liste de Moyennes)

Soit P une liste de listes de nombres. Définir 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 5.5 (Somme des chiffres d'un nombre)

Soit $n \in \mathbb{N}$. Calculer la somme de ses chiffres. Par exemple, si n=30071966, on doit obtenir 32.

Correction
Pour résoudre ce problème on converti ce nombre en chaîne de caractères, on lit les chiffres un par un, on les converti en entier, enfin on les additionne :

```
n = 30071966
s = sum([int(c) for c in str(n)])
print(s)
```

32

Exercice 5.6 (Premières lettres)
Récupérer la première lettre des mots d'une liste commençant par une voyelle. La liste est

["maths", "info", "python", "exposant", "alpha", "fonction", "parabole", "equilateral", "orthogonal", "cercle", "isocèle"]

Correction
`mot[0]` récupère la première lettre de `mot`. Pour vérifier que c'est une voyelle, on vérifie qu'elle appartient à `"aeiouy"` :

```
>>> ma_liste_de_mots= ["maths", "info", "python", "exposant", "alpha", "fonction", "parabole",
 ↪  "equilateral", "orthogonal", "cercle", "isocèle" ]
>>> [ mot[0] for mot in ma_liste_de_mots if mot[0] in "aeiouy"]
['i', 'e', 'a', 'e', 'o', 'i']
```

Exercice 5.7 (Dernières lettres)
Afficher la liste des mots de la liste donnée qui se terminent par un `"s"`. La liste est

["maths", "info", "python", "exposant", "alpha", "fonction", "parabole", "equilateral", "orthogonal", "cercle", "isocèle"]

Correction
`mot[-1]` récupère la dernière lettre de `mot`.

```
>>> liste=["maths", "info", "python", "exposants", "alpha", "fonctions", "parabole",
 ↪  "equilateral", "orthogonal", "cercles", "isocèle" ]
>>> [ mot for mot in liste if mot[-1]=="s"]
['maths', 'exposants', 'fonctions', 'cercles']
```

Exercice 5.8 (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).

Correction

```
>>> n = 100
>>> [d for d in range(1,n) if (n%d==0)]
[1, 2, 4, 5, 10, 20, 25, 50]
```

Mieux :

```
>>> 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 5.9 (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,

$$\left.\begin{array}{rl} a & = 220 \text{ est divisible par } 1,2,4,5,10,11,20,22,44,55,110 \\ d(a) & = 1+2+4+5+10+11+20+22+44+55+110 = 284 > a \end{array}\right] \Longrightarrow \quad a = \text{ est abondant}$$

$$\left.\begin{array}{rl} b & = 284 \text{ est divisible par } 1,2,4,71,142 \\ d(b) & = 1+2+4+71+142 = 220 < b \end{array}\right] \Longrightarrow \quad b = \text{ est déficient}$$

$$\left.\begin{array}{rl} c & = 28 \text{ est divisible par } 1,2,4,7,14 \\ d(c) & = 1+2+4+7+14 = 28 = c \end{array}\right] \Longrightarrow \quad c = \text{ est parfait}$$

Classer tous les nombres $n \leq 100$ 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("n={:3d}, s={:3d}, diviseurs={}".format(n,s,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 5.10 (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 5.11 (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 ou égale 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 int(str(n)[-1])==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]
```

Deuxième méthode : on construit les nombres $n = m \times 10^2 + d \times 10 + u$:

$$n = \boxed{m}\boxed{d}\boxed{u}$$

L'entier se termine par 3 donc $u = 3$, le chiffre des dizaines est pair donc $d = 0, 2, 4, 6, 8$, la somme des chiffres est supérieure ou égale à 15 ainsi $m > 12 - d$:

```
L=[]
for d in range(0,10,2):
⟶for m in range(12-d+1,10):
⟶⟶L.append(m*10**2+d*10+3)
L.sort()
print(L)
```

```
[583, 683, 763, 783, 863, 883, 943, 963, 983]
```

qui s'écrit aussi comme

```
L=[m*10**2+d*10+3 for d in range(0,10,2) for m in range(12-d+1,10)]
L.sort()
print(L)
```

```
[583, 683, 763, 783, 863, 883, 943, 963, 983]
```

Exercice 5.12 (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']
```

Écrire un script qui génère et affiche la liste de tuples suivante :

```
[('Janvier',31),('Fevrier',28)...
```

Correction

```
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)]

Un dictionnaire est une structure un peu plus adaptée :

```
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 5.13 (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 5.14 (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 ?
- ▷ Que vaut après un an de l'achat ?
- ▷ 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 :

$$\begin{aligned} 40 + 300 \times (0.95)^n < 100 &\iff (0.95)^n < \frac{100-40}{300} = \frac{1}{5} = 5^{-1} \\ &\iff n\ln(0.95) < -\ln(5) \iff n > -\frac{\ln(5)}{\ln(0.95)} \simeq 31.377 \end{aligned}$$

Vérifions nos calculs :

```
uu=[430]+[40+300*0.95**n for n in range(1,50)]
indice=([u>100 for u in uu]).count(True)
print(f"u_{indice-1}={uu[indice-1]} et u_{indice}={uu[indice]}")
```

```
u_31=101.17204772373711 et u_32=98.11344533755026
```

Exercice 5.15 (Années bissextiles)

Depuis l'ajustement du calendrier grégorien, l'année sera bissextile si l'année est

$$\Big(\text{divisible par 4} \quad \text{et} \quad \text{non divisible par 100}\Big) \qquad \text{ou} \qquad \Big(\text{est divisible par 400.}\Big)$$

Ainsi,

- ▷ 2019 n'est pas bissextile car non divisible par 4
- ▷ 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]

Exercice 5.16 (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 : $[1, 3, 6, 10, 15, 21, 28, 36, 45, 55, \dots]$

Écrire la suite des premiers 100 nombres triangulaires.

Correction
Version naïve :

$$L_n = \sum_{i=1}^{n} i$$

```
L=[ sum(range(1,n+1)) for n in range(1,101)]
print(L)
```

[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]

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)
```

[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]

Encore mieux :

$$L_n = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

```
mylist=[int(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 5.17 (Table de multiplication)

Afficher la table de multiplication par $1, \ldots, 10$ suivante (l'élément en position (i, j) est égal au produit ij 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]]

Remarque : pour une affichage plus lisible on pourra écrire

```
for row in M:
⟶#print( '\t'.join([str(r) for r in row]) )
⟶print( '\t'.join(map(str,row)) )
```

Explications :

- ▷ `map(str,row)` signifie appliquer l'instruction `str` à `row`, ce qui équivaut à `[str(r) for r in row]`
- ▷ `'A'.join('1234')` donne `'1A2A3A4'`

Exercice 5.18 (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

$$\begin{aligned}\sum_{k=1}^{\lfloor\frac{2013-1}{5}\rfloor} 5k + \sum_{k=1}^{\lfloor\frac{2013-1}{7}\rfloor} 7k - \sum_{k=1}^{\lfloor\frac{2013-1}{35}\rfloor} 35k &= 5\sum_{k=1}^{402} k + 7\sum_{k=1}^{287} k - 35\sum_{k=1}^{57} k \\ &= 5\frac{402\times 403}{2} + 7\frac{287\times 288}{2} - 35\frac{57\times 58}{2} \\ &= 405015 + 289296 - 57855 = 636456\end{aligned}$$

Exercice 5.19 (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 $153 = 1^3 + 5^3 + 3^3$. On peut montrer qu'il n'existe que 4 nombres de Armstrong et qu'ils ont tous 3 chiffres. Écrire la liste des 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]
```

Exercice 5.20 (Nombre de chiffres)

Combien de fois le chiffre 4 apparaît en écrivant les nombres de 1 à 1234 inclus ?

Correction

```
print(sum( [str(i).count('4') for i in range(1,1235) ] ))
```

```
344
```

Exercice 5.21 (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 5.22 (Défi Turing n°48 – $1^1 + 2^2 + 3^3 + \cdots$)

$1^1+2^2+3^3+\cdots+10^{10} = 10405071317$. Donner les 10 premiers chiffres de la série $1^1+2^2+3^3+\cdots+2013^{2013}$.

Correction

```
print(str(sum([i**i for i in range(2013+1)]))[:10])
```

```
4341084564
```

Exercice 5.23 (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 5.24 (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://callicode.fr/pydefis/PokeNombresCommuns/txt

Exercice Bonus 5.25 (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://callicode.fr/pydefis/Herculito11Pommes/txt

Exercice Bonus 5.26 (Pydéfi – Constante de Champernowne)

La constante de Champernowne est un nombre compris entre 0 et 1, dont le développement décimal est obtenu en écrivant successivement les nombre entiers. Elle commence ainsi :

$$0,12345678910111213141515171819202122\ldots$$

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://callicode.fr/pydefis/Champernowne/txt

Exercice Bonus 5.27 (Défi Turing n° 40 – La constante de Champernowne)

La constante de Champernowne est un nombre irrationnel créé en concaténant les entiers positifs :

$$0,12345678910111213141515171819202021\ldots$$

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 5.28 (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). Les

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://callicode.fr/pydefis/ChiffresPreferes/txt

Exercice Bonus 5.29 (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://callicode.fr/pydefis/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 5.30 (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://callicode.fr/pydefis/Ordre66/txt

Exercice Bonus 5.31 (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://callicode.fr/pydefis/Desamorcage02/txt

☆ Exercice Bonus 5.32 (Défi Turing n°29 – puissances distincts)

Considérons a^b pour $2 \leq a \leq 5$ et $2 \leq b \leq 5$:

$$\begin{array}{llll} 2^2 = 4, & 2^3 = 8, & 2^4 = 16, & 2^5 = 32 \\ 3^2 = 9, & 3^3 = 27, & 3^4 = 81, & 3^5 = 243 \\ 4^2 = 16, & 4^3 = 64, & 4^4 = 256, & 4^5 = 1024 \\ 5^2 = 25, & 5^3 = 125, & 5^4 = 625, & 5^5 = 3125 \end{array}$$

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 $2 \leq a \leq 1000$ et $2 \leq b \leq 1000$?

Correction

```
>>> print (len(set(a**b for a in range(2,1001) for b in range(2,1001))))
977358
```

☆ Exercice Bonus 5.33 (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	$T_n = \frac{n(n+1)}{2}$	$1, 3, 6, 10, 15, \ldots$
pentagonal	$P_n = \frac{n(3n-1)}{2}$	$1, 5, 12, 22, 35, \ldots$
hexagonal	$H_n = n(2n-1)$	$1, 6, 15, 28, 45, \ldots$

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 5.34 (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 no 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]
```

☆ Exercice Bonus 5.35 (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)
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 5.36 (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 un 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://callicode.fr/pydefis/Monnaie

Exercice Bonus 5.37

Construire la liste des nombres de Fibonacci avec une liste de compréhension.

Correction

```
n=10
x=[0,1]
fibs = x[0:2] + [ x.append(x[-2]+x[-1]) or x[-1] for i in range(n-2)]
print(fibs)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Chapitre 6.

Fonctions

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.

Il est tout à fait possible de définir une fonction (au sens du langage Python) qui ressemble à une fonction mathématique. 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 [1] 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). 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.

```
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))
```

```
451
```

Si l'instruction `return` est absente, la fonction renvoi l'objet `None` (ce sera le cas lorsqu'on passe un objet mutable, par exemple une liste).

Attention

Ne pas confondre la fonction `print` et l'instruction `return` :

```
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
```

Voici un bêtisier pour mieux comprendre les règles :

- ▷ il manque les deux-points en fin de ligne :

```
>>> def f(x)
  File "<stdin>", line 1
    def f(x)
            ^
SyntaxError: invalid syntax
```

- ▷ il manque l'indentation :

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 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
```

▷ 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`.

▷ 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
```

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
m=2

def f(y):
⟶x = 7 # x est redefinie localement, m reste inchange
⟶return x*y*m

print(f(1))
print(x)
```

```
14
6
```

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 peu 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.
- ▷ 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.

Fonctions prédéfinies : `abs all any ascii bin bool bytearray bytes chr classmethod compile complex copyright credits delattr dict dir divmod enumerate eval exec exit filter float format frozenset getattr globals hasattr hash help hex id input int isinstance issubclass iter len license list locals map max memoryview min next object oct open ord pow print property quit range repr reversed round set setattr slice sorted staticmethod str sum super tuple type vars zip`

6.1. 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` :

$$\begin{array}{ccccc} & x & \mapsto & 2x \\ & \downarrow & \downarrow & \downarrow \\ \texttt{lambda} & \texttt{x} & \texttt{:} & \texttt{2*x} \end{array}$$

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 $f(x)$» 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 $f : x \mapsto x^2$ et $g : x \mapsto x - 1$ 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 :

```
>>> 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 choses plus compliquées, on utilise `def` (on peut toujours).

Les fonctions `lambda` sont surtout utiles pour passer une fonction en paramètre à une autre (par exemple, pour appliquer la fonction à tous les éléments d'une liste). Par exemple, la liste suivante est générée par une liste de compréhension :

```
>>> g = lambda x : x+1
>>> [g(x) for x in [1, 3, 42]]
[2, 4, 43]
```

6.1.1. ★ Fonctions lambda, map, listes en compréhensions, filter, zip

Les codes suivants affichent tous `[0,1,4,9,16]` :

avec une boucle

```
L=[]
for i in range(5):
    L.append(i**2)
print(L)
```

avec une liste en compréhension

```
L=[i**2 for i in range(5)]
print(L)
```

avec `map()`

```
L=list(map( lambda x:x**2 ,
 ↪  range(5) ))
print(L)
```

[0, 1, 4, 9, 16]

Les codes suivants affichent tous `[0,4,16]` :

avec une boucle

```
L=[]
for i in range(5):
    if i**2%2==0:
        L.append(i**2)
print(L)
```

avec une liste en compréhension

```
L=[i**2 for i in range(5) if
 ↪  i**2%2==0]
print(L)
```

avec `map()`

```
L=map( lambda x:x**2 ,
 ↪  range(5) )
L=list(filter( lambda x :
 ↪  x%2==0, L))
print(L)
```

[0, 4, 16]

1. La fonction invoquée dans une liste en compréhension ne doit pas forcement renvoyer un résultat stockable dans une liste, on peut utiliser n'importe quelle fonction !
   1.1. Par exemple, on peut écrire
   ```
   L=['sinus','cosinus','tangente']
   [print(i) for i in L]
   ```

```
sinus
cosinus
tangente
```

Cependant, la liste sera bien crée et elle contiendra des "None" :

```
L=['sinus','cosinus','tangente']
A=[print(i) for i in L]
print(A)
```

```
sinus
cosinus
tangente
[None, None, None]
```

1.2. Dans ces cas il est préférable d'utiliser `map` :

```
L=['sinus','cosinus','tangente']
list(map(print,L))
```

```
sinus
cosinus
tangente
```

2. On reprend le même exemple mais on veut rajouter une condition.

2.1. On affiche l'indice de tous les "e" dans un texte :

```
texte="On affiche l'indice de tous les 'e' dans ce texte"
[print(i) for i,c in enumerate(texte) if c=="e"]
```

```
9
18
21
29
33
42
45
48
```

On a le même problème : la liste sera crée et elle contiendra des "None".

2.2. Dans ces cas il est préférable d'utiliser `map` après un `filter` :

```
texte="On affiche l'indice de tous les 'e' dans ce texte"
list(map(print,filter(lambda c : c=="e", texte)))
```

```
e
e
e
e
e
e
e
e
```

3. Fusionner deux listes :

3.1. On peut utiliser une **boucle `for`** classique :

```
num=[1,2,3]
mot=['one','two','three']
L=[]
for i in range(len(num)):
⟶L.append((num[i],mot[i]))
print(L)
```

```
[(1, 'one'), (2, 'two'), (3, 'three')]
```

3.2. On peut utiliser une **liste en compréhensions** :

```
num=[1,2,3]
mot=['one','two','three']
L = [ (num[i],mot[i]) for i in range(len(num)) ]
print(L)
```

```
[(1, 'one'), (2, 'two'), (3, 'three')]
```

3.3. On peut utiliser **zip** :

```
num=[1,2,3]
mot=['one','two','three']
L=list(zip(num,mot))
print(L)
```

```
[(1, 'one'), (2, 'two'), (3, 'three')]
```

4. Pour une liste donnée, on veut extraire les éléments pairs de cette liste.

4.1. On peut utiliser une **boucle `for` classique** :

```
my_list = list(range(1,11))
sous_liste= []
for item in my_list:
    if item%2==0:
        sous_liste.append(item)
print(sous_liste)
```

```
[2, 4, 6, 8, 10]
```

4.2. On peut utiliser une **liste en compréhensions** :

```
my_list = list(range(1,11))
sous_liste = [item for item in my_list if item%2==0]
print(sous_liste)
```

```
[2, 4, 6, 8, 10]
```

4.3. On peut utiliser **`filter` et une fonction `lambda`** :

```
my_list = list(range(1,11))
sous_liste =  list(filter(lambda x : x%2==0, my_list))
print(sous_liste)
```

```
[2, 4, 6, 8, 10]
```

6.2. ☆ Fonctions récursives

Une fonction récursive est tous simplement une fonction qui s'appelle elle même.
Une fonction mathématique définie par une relation de récurrence et une condition initiale peut être programmée de manière récursive de façon naturelle. Par exemple :

$$\begin{cases} u_0 = 1, \\ u_{n+1} = 2u_n \end{cases}$$

Version récursive :

```
def UR(n):
    if n==0:
        return 1
    else:
        return 2*UR(n-1)

print(UR(6))
```

```
64
```

Version itérative :

```
def UI(u):
    return 2*u

N=5
u=1
for n in range(N+1):
    u=UI(u)
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 :

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
```

- ▷ 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 fait si $n = 1$ alors le programme va chercher à calculer `somme(1)` puis `somme(0)` puis `somme(-1)` sans jamais s'arrêter.
- ▷ 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 moyen 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)`.

6.3. Exercices

Exercice 6.1 (Devine le résultat - variables globales *vs* locales)

Cas 1 :
```
def test():
    x = "hello"

print(x)
```

Cas 2 :
```
x = "hello"
def test():
    print(x)

test()
```

Correction

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"
...
>>> 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.

```
>>> x = "hello"
>>> def test():
...     print(x)
...
>>> test()
hello
```

Exercice 6.2 (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 :

$$f(x,a,b) = \begin{cases} f(x,b,a) & \text{si } a > b \\ a & \text{si } x \leq a \text{ avec } a \leq b \\ b & \text{si } x \geq a \text{ avec } a \leq b \\ b & \text{si } (x-a) > (b-x) \text{ avec } a \leq b \\ a & \text{sinon} \end{cases}$$

Si $a > b$ on échange a et b : on peut donc restreindre l'étude au cas $a \leq b$ et étudier :

$$g(x,a,b) = \begin{cases} a & \text{si } x \leq a \text{ ou si } (x-a) \leq (b-x) \\ b & \text{si } x \geq a \text{ ou si } (x-a) > (b-x) \end{cases} = \begin{cases} a & \text{si } x \leq \frac{a+b}{2} \\ b & \text{si } x > \frac{a+b}{2} \end{cases}$$

et

$$f(x,a,b) = \begin{cases} g(x,b,a) & \text{si } a > b \\ g(x,a,b) & \text{sinon} \end{cases}$$

ainsi

$$f(0,0,0) = 0, \quad f(-2,0,3) = 0, \quad f(-2,3,0) = f(-2,0,3) = 0,$$
$$f(2,0,2) = 2, \quad f(1,0,2) = 0, \quad f(3,-1,-2) = (3,-2,-1) = -1.$$

Exercice 6.3 (Premières fonctions)

Écrire des fonctions sans paramètres ni sortie

▷ écrire une fonction appelée `affiche_table_de_7()` qui affiche la table de multiplication par 7 : `1×7=7, 2×7=14 ...`

Écrire des fonctions sans paramètres avec sortie

▷ écrire une fonction appelée `my_PI()` qui renvoi `3.1415`

Écrire des fonctions avec paramètres sans sortie

▷ écrire une fonction appelée `affiche_une_table(n)` qui dépend d'un paramètre n et qui affiche la table de multiplication par n. Par exemple, la commande `affiche_une_table(5)` affichera `1×5=5, 2×5=10 ...`

▷ 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"`. 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"]`

Écrire des fonctions avec paramètres et sortie

▷ écrire une fonction appelée `euros_vers_dollars(montant)` qui dépend d'un paramètre et qui pour une somme d'argent `montant`, 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 (sans afficher) 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.

Correction

Sans paramètres ni sortie

```
def  affiche_table_de_7():
    for i in range(10):
        print(f"{i}x7={i*7}")

# TEST
affiche_table_de_7()
```

```
0x7=0
1x7=7
2x7=14
3x7=21
4x7=28
5x7=35
6x7=42
7x7=49
8x7=56
9x7=63
```

Sans paramètres avec sortie

```
def  my_PI():
    return 3.1415

# Avec une lambda function on écrira
# my_PI = lambda : 3.1415

# TEST
print(my_PI())
```

```
3.1415
```

Avec paramètres sans sortie

```
def  affiche_une_table(n):
    for i in range(10):
        print(f"{i}x{n}={i*n}")

# TEST
affiche_une_table(5)
```

```
0x5=0
1x5=5
2x5=10
3x5=15
4x5=20
5x5=25
6x5=30
7x5=35
8x5=40
9x5=45
```

Avec paramètres sans sortie

```
def  modifier_liste(L):
    L.append("coucou")

# TEST
A=[1,5,10]
print(A)
modifier_liste(A)
print(A)
```

```
[1, 5, 10]
[1, 5, 10, 'coucou']
```

Avec paramètres et sortie

```
def  euros_vers_dollar(montant):
        return 1.13*montant

# Avec une lambda function on ecrira
# euros_vers_dollar = lambda montant :
 ↪  1.13*montant

# TEST
mE=10
mD=euros_vers_dollar(mE)
print(f"{mE} euro = {mD:.2f} dollar")
```

```
10 euro = 11.30 dollar
```

Avec paramètres et sortie

```
def  calcul_puissance(x,n):
    return x**n

# Avec une lambda function on ecrira
# calcul_puissance = lambda (x,n) : x**n

# TEST
print(calcul_puissance(2,10))
```

```
1024
```

Avec paramètres et sortie

```
def somme_produit(a,b):
    """
    Calcule somme et produit de deux nombres
    """
    return a+b, a*b
```

```
# Avec une lambda function on ecrira
# somme_produit = lambda (a,b) : a+b, a*b

# TEST
som, pro = somme_produit(6,7)
print(som)
```

```
print(pro)
```

```
13
42
```

Exercice 6.4 (Valeur absolue)

Implémenter une fonction `myabs` : $x \mapsto |x|$ sans utiliser la fonction valeur absolue `abs` du module `math`.

Correction

Deux implémentations :

1. ```
   def myabs(x):
   ⟶if x<0:
   ⟶⟶x=-x
   ⟶return x
   # TESTS
   print(myabs(-1),myabs(0),myabs(1))
   ```

   ```
   1 0 1
   ```

2. ```
   myabs = lambda x : (x>=0)*(x) + (x<0)*(-x)
   # TESTS
   print(myabs(-1),myabs(0),myabs(1))
   ```

   ```
   1 0 1
   ```

Exercice 6.5 (Premières fonctions λ)

Écrire les fonctions suivantes comme une fonction λ et avec le mot clé `def`.

$$f_1 : \mathbb{R} \to \mathbb{R} \qquad x \mapsto x^2 + 5$$

$$f_2 : \mathbb{R}^2 \to \mathbb{R} \qquad (x, y) \mapsto x + y - 2$$

$$f_3 : \mathbb{R} \to \mathbb{R}^2 \qquad x \mapsto (x^2, x^3)$$

$$f_4 : \mathbb{R}^2 \to \mathbb{R}^2 \qquad (x, y) \mapsto (x + y, x - y)$$

$$f_5 : \mathbb{R} \to \mathbb{R} \qquad x \mapsto \begin{cases} 3 & \text{si } x < 10 \\ -2x & \text{si } 10 \leq x < 15 \\ x^2 & \text{sinon} \end{cases}$$

Correction

▷ Avec des fonctions lambda :

```
f1 = lambda x : x**2+5
f2 = lambda x,y : x+y-2
f3 = lambda x : (x**2,x**3)
f4 = lambda x,y : (x+y,x-y)
f5 = lambda x : 3*(x<10)-2*x*(10<=x<15)+x**2*(x>=15)
# TESTS
print(f1(2))
print(f2(2,3))
print(f3(2))
print(f4(2,3))
print(f5(2),f5(12),f5(22))
```

```
9
3
(4, 8)
(5, -1)
3 -24 484
```

▷ Avec `def` :

```
def f1(x) :
    return x**2+5

def f2(x,y) :
    return x+y-2

def f3(x) :
    return (x**2,x**3)

def f4(x,y) :
    return (x+y,x-y)

def f5(x) :
    if x<10:
        return 3
    elif x<15:
        return -2*x
    else:
        return x**2

# TESTS
print(f1(2))
print(f2(2,3))
print(f3(2))
print(f4(2,3))
print(f5(2),f5(12),f5(22))
```

```
9
3
(4, 8)
(5, -1)
3 -24 484
```

Exercice 6.6 (Divisibilité)

Écrire une fonction qui retourne 1 si n est un multiple de 2014.

Correction

```
#def mul2014(n):
#     return 1 if n % 2014 == 0 else 0

#IDEM
mul2014 = lambda n : 1*(n%2014==0)+0

print(mul2014(2014), mul2014(2015))
```

```
1 0
```

Exercice 6.7 (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

```
#def v_retenue(v):
#    if v<100:
#        return v-5
#    else:
#        return v*0.95

# IDEM
v_retenue = lambda v : (v<100)*(v-5)+(v>=100)*(v*0.95)

# TESTS
for v in [60,100,110]:
    print(f"vitesse={v}, vitesse retenue = {v_retenue(v)}")
```

```
vitesse=60, vitesse retenue = 55.0
vitesse=100, vitesse retenue = 95.0
vitesse=110, vitesse retenue = 104.5
```

Exercice 6.8 (Validité d'un code)

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 $15 \mod 7 = 1 \neq 3$ 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"`.

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"

# TESTS
for N in [5739,5731]:
    print(f"N={N} {photocopieuse(N)}")
```

```
N=5739 NON VALIDE
N=5731 VALIDE
```

Exercice 6.9 (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 si il n'y a pas eu d'erreur en reportant le numéro sur un formulaire ou lors du transfert informatique par exemple. En effet,

$$c = 97 - r, \qquad \text{où } r = \text{reste de la division euclidienne de } n \text{ par } 97$$

Exemple : si le numéro de sécurité social est $N = 970000000010094$ alors $n = 9700000000100$ et le reste de la division euclidienne de n par 97 est 3. La clé est bien $97 - 3 = 94$.

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"

# 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 6.10 (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 $p \times (100 - x)/100$.

```
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 :

```
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 6.11 (Liste impairs)

Écrire une fonction 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=1
R=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]
```

Exercice 6.12 (Liste divisibles)

Écrire une fonction 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 6.13 (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.

Correction

```
>>> miroir = lambda n : int(str(n)[::-1])
>>> print(miroir(7423))
3247
```

Exercice 6.14 (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 $[v_{\text{min}}; v_{\text{max}}]$ en un ensemble de valeurs dans $[0; 1]$.

Générez une liste de valeurs aléatoires de cette façon :

```
import random
L=[random.randint(10,100)]
```

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=[43, 24, 37, 68, 83, 23, 91, 42, 30, 26]
G=[0.29411764705882354, 0.014705882352941176, 0.20588235294117646, 0.6617647058823529,
 ↪  0.8823529411764706, 0.0, 1.0, 0.27941176470588236, 0.10294117647058823,
 ↪  0.04411764705882353]
```

Exercice 6.15 (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 6.16 (Voyelles)

Écrire une fonction qui prend en entrée une chaîne de caractères donnée et renvoie le nombre de voyelles.

Correction

```
VOYELLES_FR = "AaÀàEeÈèÉéIiOoUuÙùYy"

#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", "aioli", "pèche", "Supercalifragilisticexpidélilicieux" ]:
⟶print("{} contient {} voyelles".format(s,voyelles(s)))
```

```
citron contient 2 voyelles
fraise contient 3 voyelles
aioli contient 4 voyelles
pèche contient 2 voyelles
Supercalifragilisticexpidélilicieux contient 16 voyelles
```

Exercice 6.17 (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 6.18 (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]

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 6.19 (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.

Correction

Une version naïve est la suivante :

```
def isPrime(n):
    if n==2 or n==3: return True
    if n%2==0 or n<2: return False # nombre pair ou égale à 1
    for i in range(3,int(n**0.5)+1,2):   # on s'arrete 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 6.20 (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 $x = \frac{\pi}{2}$

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 $f''\left(\frac{\pi}{2}\right) = 0$.

```
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(math.cos,math.pi/2,1.0e-5)
print('First derivative =', df)
print('Second derivative =', ddf)
```

```
First derivative = -0.9999999999898845
Second derivative = 1.6940658945086004e-11
```

Exercice 6.21 ($f : \mathbb{R} \to \mathbb{R}^2$)

Devine le résultat :

```
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) )
```

Correction

```
[2, 1]
[2, 1]
```

☆ Exercice Bonus 6.22 (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)])
n=1
test=True
while test:
    B=f(n)
    if 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
    else:
        test=False
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 6.23 (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
```

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):
    abc="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    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 6.24 (Pydéfi – La paranoïa de Calot)

Jean-René Calot, de retour en France après une très longue mission d'agent double dans un pays de l'Est est devenu complètement paranoïaque. En effet, de peur d'être empoisonné avec un agent Novitchok, il s'est mis en tête de chiffrer tous ses messages, même les plus anodins, à destination de ses collègues des services secrets. Après un temps d'exaspération, ses collègues ont fini par s'y faire et ont surtout fini par découvrir comment il chiffre ses messages. En effet, ils savent qu'il utilise les lignes suivantes de son clavier azerty :

a	z	e	r	t	y	u	i	o	p
q	s	d	f	g	h	j	k	l	m
w	x	c	v	b	n	,	;	:	!

et qu'il applique un décalage circulaire vers la droite (décalage positif) ou vers la gauche (décalage négatif) des touches du clavier. Ce décalage est compris entre -9 et +9. De plus, Calot supprime tous les espaces de ses textes.

Ainsi, à titre d'exemple, le texte "rendezvousouvoussavez" sera chiffré "zavqapxutmutxutmmoxap" si l'on applique un décalage de -2.

Merlaux, Jacquard et Moulinier ont reçu de Calot le message chiffré suivant :

```
hok:sopkhopgue;du:jeuploahtegrxropcdoyptytkojeoxtekxtekpoapte!
rtxkluxkexyoarapokauepuxaltxagokyguakxoktxayukmpux:sohoxawtxkh
urkjerugu!uxauqolowpterggopgokyrkaoklokohytrktxxoepkuhokaptekk
okvytephopoapte!opoxyupauxalewepoueuggoiatealptracatepxoiuque:
socatepxoiuque:socatepxoiulptraocuggoiatealptracyr!taoikep!tek
hohocpoxapoiluxkgoyup:cuppr!ougumtxaurxocohypexaoigoguwzprxaso
!oqoaugcmtx:oiatealptrau::pteyrkytepoxapopluxkgokatrgoaaokyewg
rjeokcuemtxlulptrao!tekapte!opoiexoytpaolokop!r:ojerltxxolrpo:
aohoxau::okue;:erkrxoklepokauepuxacgo:tloloxapoookajeupuxaoloe
;:rxjeuxaoseralrokocuaaoxartxoxte!puxaguytpaodurytkrartxxoexko
uelu:rlokyo:rugohoxarhytpaoleyup:zoggt,katxodekaouelokkeknn:og
ukopuralthhuqojeo!tekktzoilrkktekdekaou!uxalohopoapte!opnjeuxl
go:erkrxrop!teklohuxlopu:ojeo!tek!tegoihuxqoplraokgerblok:tpxr
:stxkue!rxurqpooaleqtgtewakzcrg!tek:txlerpuauhuauwgon
```

Validez le défi en indiquant d'où vient le piège de Calot.

Source : https://callicode.fr/pydefis/ParanoiaCalot

☆ Exercice Bonus 6.25 (Défi Turing n° 78 – $abcd = a^b c^d$)

Quel est le seul nombre ayant le motif suivant : $abcd = a^b c^d$?

Correction

On génère chaque chiffre séparément :

```
def pb78():
    for a in range(10):
        for b in range(10):
            for c in range(10):
                for d in range(10):
                    L=1000*a+100*b+10*c+d
                    R=(a**b)*(c**d)
                    if L==R:
                        return L

print(pb78())
```

2592

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). Variante : on génère les nombres $abcd$ (une seule boucle), on extrait les chiffres et on s'arrête dès qu'on trouve n avec l'instruction `break` :

```
for n in range(1000,10000):
⟶a,b,c,d=[int(c) for c in str(n)] # a,b,c,d=tuple(map(int,str(n)))
⟶if n==(a**b)*(c**d):
⟶⟶print(n)
⟶⟶break
```

2592

☆ Exercice Bonus 6.26 (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,

$a = 220$ est divisible par $1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110$ $\quad$ $b = 284$ est divisible par $1, 2, 4, 71, 142$

$d(a) = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284 = b$ $\quad$ $d(b) = 1 + 2 + 4 + 71 + 142 = 220 = a$

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 6.27 (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

- ▷ `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 :

```
>>> n=123
>>> list(map(int,str(n)))
[1, 2, 3]
>>> [int(x) for x in str(n)]
[1, 2, 3]
```

☆ Exercice Bonus 6.28 (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 14.90 €. Donc, si le nombre de tickets à acheter est supérieur ou égal à 10, on calculera le nombre de carnets et on complétera par des tickets achetés à l'unité. Cependant, déjà avec 8 tickets il convient d'acheter un carnet et garder 2 tickets inutilisés plutôt qu'acheter 8 tickets à l'unité car sans carnet on paye $1.90 \times 8 = 15.20$ € tandis qu'un carnet coûte 14.90 €.

Écrire une script qui calcule la somme minimale à payer si on prévoit `n` voyages ; nous dit 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 = 14.90

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 {nbTick
    else:
        if nbTicketsAAcheter>0:

            ↪ print(f"\tsi on achète {nbCarnets} carnet(s) et {nbTicketsAAcheter} ticket(s) à l'unit
        else:
            print(f"\tsi on achète {nbCarnets} carnet(s) on paye {somme_due:.2f}€")
    return

# TEST
for n in [7,8,9,10,11,12,20,21]: # nombres de voyages
⟶RATP(n)
```

```
Pour 7 voyage(s)
⟶on achète 7 ticket(s) à l'unité et on paye 13.30€
Pour 8 voyage(s)
⟶sans carnet on payerait 15.20
⟶si on achète 1 carnet(s) on paye 14.90€ et il reste 2 voyage(s)
Pour 9 voyage(s)
⟶sans carnet on payerait 17.10
⟶si on achète 1 carnet(s) on paye 14.90€ et il reste 1 voyage(s)
Pour 10 voyage(s)
⟶sans carnet on payerait 19.00
⟶si on achète 1 carnet(s) on paye 14.90€
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 16.80€
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 18.70€
Pour 20 voyage(s)
⟶sans carnet on payerait 38.00
⟶si on achète 2 carnet(s) on paye 29.80€
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 31.70€
```

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 {nbTicketsRe
    else:
        if nbTicketsAAcheter>0:

            ↪ print(f"\tsi on achète {nbCarnets} carnet(s) et {nbTicketsAAcheter} ticket(s) à l'unité on
        else:
            print(f"\tsi on achète {nbCarnets} carnet(s) on paye {somme_due:.2f}€")
    return
```

☆ Exercice Bonus 6.29 (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("\tsans carnet on paye {:.2f}€".format(p))

    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("Pour {} voyage(s) dont {} en bus et {} en mer".format(vBus+vBat,vBus,vBat))
⟶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:.
```

```
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("\tsans carnet on paye {:.2f}€".format(cBus*vBus+cBat*vBat))
    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 6.30 (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 6.31 (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://callicode.fr/pydefis/Herculito12Cerbere

☆ Exercice Bonus 6.32 (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) :

```
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)

print(cherche())
```

```
9744479
```

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).

☆ Exercice Bonus 6.33 (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 6.34 (Pydéfi – Les boeufs 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://callicode.fr/pydefis/Herculito10Boeufs>

Exercice Bonus 6.35 (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://callicode.fr/pydefis/Herculito01Lion

Exercice Bonus 6.36 (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.

Défi : donnant leur équivalent en Neutubykwa des nombres [363, 204, 108, 181, 326, 154, 259, 289, 332, 137].

Testez votre code : si la liste d'entrée était [118, 3, 24], il faudrait répondre `'BYNEUBYTU'`, `'NEU'`, `'BYTUKWA'`.

Source : https://callicode.fr/pydefis/Neutubykwa

Exercice Bonus 6.37 (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 6.38 (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

$$666 \xrightarrow[6\times6\times]{} 216 \xrightarrow[2\times1\times6]{} 12 \xrightarrow[1\times2]{} 2$$

Combien de nombres entre 10 et 10 millions ont comme image 6 ?

Correction

```
def prod(L):
⟶p=1
⟶for ell in L:
⟶⟶p*=ell
⟶return p

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 6.39 (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

$$\begin{aligned}
39 &\xrightarrow[\text{binaire}]{} 100111 \xrightarrow[\text{miroir}]{} 111001 \xrightarrow[\text{base 10}]{} 57 \xrightarrow[+2]{} \\
59 &\xrightarrow[\text{binaire}]{} 111011 \xrightarrow[\text{miroir}]{} 110111 \xrightarrow[\text{base 10}]{} 55 \xrightarrow[+2]{} \\
57 &\xrightarrow[\text{binaire}]{} 111001 \xrightarrow[\text{miroir}]{} 100111 \xrightarrow[\text{base 10}]{} 39 \xrightarrow[+2]{} \\
41 &\xrightarrow[\text{binaire}]{} 101001 \xrightarrow[\text{miroir}]{} 100101 \xrightarrow[\text{base 10}]{} 37 \xrightarrow[+2]{} 39
\end{aligned}$$

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://callicode.fr/pydefis/PistoletFury

Exercice Bonus 6.40 (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://callicode.fr/pydefis/NombresHeureux

Exercice Bonus 6.41 (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://callicode.fr/pydefis/BoitesSucres

☆ Exercice Bonus 6.42 (Set and filter – Devine le résultat)

```
nums = {1, 2, 3, 4, 5, 6}
nums = {0, 1, 2, 3} & nums
nums = filter(lambda x: x > 1, nums)
print(len(list(nums)))
```

Correction

2

En effet :

```
>>> nums = {1, 2, 3, 4, 5, 6}
>>> print(nums)
{1, 2, 3, 4, 5, 6}
>>> nums = {0, 1, 2, 3} & nums
>>> print(nums)
{1, 2, 3}
>>> nums = filter(lambda x: x > 1, nums)
>>> print(list(nums))
[2, 3]
>>> print(len(list(nums)))
0
```

Chapitre 7.

Modules

Un module est une collection de fonctions. 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).

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.

   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)`.

2. 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 `FunctionName(parameters)`.

   Parfois, il est plus facile d'importer tout le contenu d'un module plutôt que de lister toutes les fonctions dont on a besoin. Pour cela, on utilise un astérisque (`*`) au lieu de la liste de fonctions : `from ModuleName import *`.
   Dans ce cas les fonctions peuvent être utilisées directement par `FunctionName(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

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.

7.2. Quelques modules

Python offre par défaut une bibliothèque de plus de deux cents modules qui évite d'avoir à réinventer la roue dès que l'on souhaite écrire un programme. Ces modules couvrent des domaines très divers : mathématiques (fonctions mathématiques usuelles, calculs sur les réels, sur les complexes, combinatoire, matrices, manipulation d'images...), administration système, programmation réseau, manipulation de fichiers, etc. [1]

7.2.1. Le module `time`

Du module `time` nous n'utiliserons que la fonction `clock()` qui renvoie une durée de temps en secondes et qui permet par soustraction de mesurer des durées d'exécution.

```
from time import clock

N=5*10**7

debut=clock()
L=[]
for i in range(N):
        L+=[i]
fin=clock()
print(f"Avec += temps d'exécution={fin-debut}s")

debut=clock()
L=[]
for i in range(N):
        L.append(i)
fin=clock()
print(f"Avec append temps d'exécution={fin-debut}s")

debut=clock()
L=[x for x in range(N)]
fin=clock()
print(f"Avec comprehensions-list temps d'exécution={fin-debut}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.

```
Avec += temps d'exécution=5.481954999999999s
Avec append temps d'exécution=4.848286999999999s
Avec comprehensions-list temps d'exécution=1.9748989999999988s
```

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 un module :

```
import math
print(math.sin(math.pi))
```

1.2246467991473532e-16

```
from math import *
print(sin(pi))
```

1.2246467991473532e-16

Voici la liste des fonctions définies dans le module `math` :

```
import math
print(dir(math))
```

Notons que le module définit les deux constantes π et e.

7.2.3. 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 :

`random.randrange(p,n,h)`	choisit un éléments aléatoirement dans la liste `range(p,n,h)`
`random.randint(a,b)`	choisit un *entier* aléatoirement dans l'intervalle $[a;b]$
`random.random()`	renvoie un *décimal* aléatoire dans $[0;1[$
`random.uniform(a,b)`	choisit un *décimal* aléatoire dans $[a;b]$
`random.choice(seq)`	choisit un éléments aléatoirement dans la liste `seq`
`random.choices(seq,k=nombre)`	tirage de `nombre` éléments dans la liste `seq` avec répétition (on remet l'objet tiré avant d
`random.sample(seq,k=nombre)`	tirage de `nombre` ($<$ `len(seq)`) éléments dans la liste `seq` sans répétition (comme si on t

Voici quelques exemples :

1. Quelques tirages :

```
>>> import random
>>> random.randrange(50,100,5)
90
>>> random.randint(50,100)
58
```

```
>>> random.choice([1,7,10,11,12,25])
25
>>> random.random()
0.7560905045659553
>>> random.uniform(10,20)
13.161826264311639
```

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: 173, 2: 152, 3: 149, 4: 193, 5: 166, 6: 167}
```

3. On donne les résultats du jeu Pierre Caillou Ciseaux sous forme de liste. On veut simuler 20 résultats sous forme d'une liste.

```
>>> from random import *
>>> seq=["Pierre", "Caillou", "Ciseaux"]
>>> print([choice(seq) for _ in range(20)])
['Ciseaux', 'Caillou', 'Caillou', 'Pierre', 'Ciseaux', 'Pierre', 'Pierre', 'Pierre',
 ↪ 'Ciseaux', 'Ciseaux', 'Ciseaux', 'Ciseaux', 'Ciseaux', 'Ciseaux', 'Pierre', 'Caillou',
 ↪ 'Pierre', 'Caillou', 'Caillou', 'Ciseaux']
```

7.2.4. Le module `scipy` (calcul approché)

Cette librairie est un ensemble très complet de modules d'algèbre linéaire, statistiques et autres algorithmes numériques. 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 comme suit :

```
from math import cos
from scipy.optimize import fsolve

sol=fsolve(lambda x: x-cos(x), 1)[0] # On approche la solution de x = cos(x)
print(sol)
```

0.7390851332151607

`integrate.quad` Pour approcher la valeur numérique d'une intégrale on peut utiliser `integrate.quad` https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

```
from math import *
from scipy import 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] )

f = lambda x : exp(-x*x)
```

```
print(integrate.quad(f, 0,inf)[0]) # On approche ∫_0^∞ e^{-x^2} dx
print(sqrt(pi)/2) # Valeur exacte de cette intégrale
```

```
Integrale = 0.33333333333333337  Erreur = 3.700743415417189e-15
0.8862269254527579
0.8862269254527579
```

7.2.5. ☆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 :

```
from sympy import *
var('x,y')

# Calculs
s=(x+y)+(x-y)
print(s)

# Simplifications
expr=sin(x)**2 + cos(x)**2
print(expr)
print(simplify(expr))

expr=(x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
print(expr)
print(simplify(expr))

# Fractions
expr=(x**3-y**3)/(x**2-y**2)
print(expr)
print(cancel(expr))
```

$$2x$$

$$\sin^2(x) + \cos^2(x)$$

$$1$$

$$\frac{x^3 + x^2 - x - 1}{x^2 + 2x + 1}$$

$$x - 1$$

$$\frac{x^3 - y^3}{x^2 - y^2}$$

$$\frac{x^2 + xy + y^2}{x + y}$$

Encore un exemple :

```
from sympy import *
var('x,y')
functions = [sin(x), cos(x), tan(x)]
for f in functions:
    d = Derivative(f, x)
    i = Integral(f, x)
    print(d, "=" , d.doit(), "\t" , i , "=" , i.doit())
```

$$\frac{d}{dx}\sin(x) = \cos(x) \qquad \int \sin(x)\,dx = -\cos(x)$$

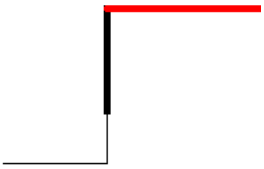
$$\frac{d}{dx}\cos(x) = -\sin(x) \qquad \int \cos(x)\,dx = \sin(x)$$

$$\frac{d}{dx}\tan(x) = \tan^2(x) + 1 \qquad \int \tan(x)\,dx = -\log(\cos(x))$$

Si vous utilisez spyder, vous pouvez remplacer `print(...)` par `display(...)`.

7.2.6. Le module `turtle`

Le module `turtle` est adapté de la fameuse Tortue de Logo, un langage informatique en partie destiné à apprendre en créant (c'est l'ancêtre de *Scratch*). Il permet de réaliser des "graphiques tortue", c'est-à-dire des dessins géométriques correspondant à la trace laissée derrière elle par une tortue virtuelle, dont on contrôle les déplacements sur l'écran de l'ordinateur à l'aide d'instructions simples.



```
from turtle import *

forward(100)
left(90)
forward(50)
width(5)
forward(100)
color( 'red' )
right(90)
forward(200)

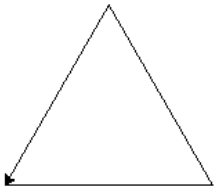
exitonclick()
```

Les principales fonctions mises à disposition dans le module `turtle` sont les suivantes :

- ▷ `clear()` Effacer les dessins
- ▷ `reset()` Effacer les dessins et réinitialiser le crayon
- ▷ `goto(x,y)` Aller au point de coordonnées (x, y)
- ▷ `forward(d)`, `backward(d)` Avancer / reculer d'une distance d
- ▷ `up()`,`down()` Relever / abaisser le crayon
- ▷ `left(alpha)`, `right(alpha)` Tourner à gauche / à droite d'un angle α (en degrés) sans avancer
- ▷ `home()` Aller au point de coordonnées $(0, 0)$ et réinitialiser l'orientation
- ▷ `color(couleur)`, `width(epaisseur)` Choisir la couleur (`"red"` , `"green"` , `"blue"` , `"orange"` , `"purple"` / l'épaisseur du trait
- ▷ `fill()` Remplir un contour fermé
- ▷ `hideturtle()`, `showturtle()` Cacher / afficher le curseur
- ▷ `pen(speed=v)` Vitesse, si $v = 0$ vitesse maximale
- ▷ `shape('turtle')` Changer le curseur
- ▷ `dot(r)` Tracer un disque de rayon r
- ▷ `stamp()` Imprimer la tortue
- ▷ `exitonclick()` Placée à la fin du code, l'exécution de cette instruction fermera la fenêtre seulement après un clique gauche

Attention

Ne pas appeler son fichier `turtle.py`. Ce nom est déjà utilisé et engendra des conflits avec l'importation du module ! Utiliser plutôt `myturtle.py` ou `turtle1.py` ou `tortue.py` etc.



```
from turtle import *

longueur_cote = 200
pen(speed=0)
forward(longueur_cote)  #1er côté
left(360/3)  ⟶⟶⟶#Angle
forward(longueur_cote)  #2ème côté
left(360/3)  ⟶⟶⟶#Angle
forward(longueur_cote)  #3ème côté

exitonclick()
```

7.3. Exercices

Exercice 7.1 (Module `math` – sin, cos, tan, π, e)
Calculer

$$\frac{\sin(3e^2)}{1+\tan\left(\frac{\pi}{8}\right)}.$$

Correction
Les trois stratégies suivantes sont équivalentes dans ce cas (car il n'y a pas de risque de conflits avec d'autres modules) :

```
from math import pi, e,sin,tan
print( sin(3*e**2)/(1+tan(pi/8)) )
```

```
-0.123823019762552
```

```
import math
print( math.sin(3*math.e**2)/(1+math.tan(math.pi/8)) )
```

```
-0.123823019762552
```

```
from math import *  # version du paresseux
print( sin(3*e**2)/(1+tan(pi/8)) )
```

```
-0.123823019762552
```

Au lieu d'utiliser la constante `math.e`, on aurait pu utiliser la fonction `math.exp` et donc écrire `math.exp(2)` au lieu de `math.e**2`.

Exercice 7.2 (Module `math` - Angles remarquables)
Soit R la liste des angles (en radians) `R=[0,pi/6,pi/4,pi/3,pi/2]`. Calculer une liste D avec les angles en degrés, une liste C avec l'évaluation du cosinus en ces angles et une liste sin avec l'évaluation du sinus.

Correction

```
>>> 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("R =",R)
R = [0, 0.5235987755982988, 0.7853981633974483, 1.0471975511965976, 1.5707963267948966]
>>> print("D =",D)
D = [0.0, 29.999999999999996, 45.0, 59.99999999999999, 90.0]
>>> print("C =",C)
C = [1.0, 0.8660254037844387, 0.7071067811865476, 0.5000000000000001, 6.123233995736766e-17]
>>> print("S =",S)
S = [0.0, 0.49999999999999994, 0.7071067811865475, 0.8660254037844386, 1.0]
```

Exercice 7.3 (Module `math` – Factorielle (version recursive))
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 récursive qui calcule la factorielle d'un entier naturel n. Pour vérifier le calcul, on pourra comparer à la fonction `factorial` du module `math`.

Correction

```
import math

def factorielle(n):
    if n==0:
        return 1
```

```
    else:
        return n*factorielle(n-1)

# TESTS
for n in [0,2,3,6,10,50]:
    print(math.factorial(n), factorielle(n))
```

```
1 1
2 2
6 6
720 720
3628800 3628800
30414093201713378043612608166064768844377641568960512000000000000
↪ 30414093201713378043612608166064768844377641568960512000000000000
```

Exercice 7.4 (Module `math` – Approximation de e)

On sait que

$$e = \lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{i!}$$

Créez un fonction `approx_e` qui prend en entrée un entier n et affiche l'approximation de e calculée avec la formule précédente.

Correction

```
>>> import math
>>> n=10
>>> sum([1/math.factorial(i) for i in range(n+1) ])
2.7182818011463845
```

Exercice 7.5 (Module `math` – Défi Turing n°6 et Projet Euler n° 20 – somme de chiffres)

$n!$ signifie $n \times (n-1) \times \cdots \times 3 \times 2 \times 1$. Par exemple, $10! = 10 \times 9 \times \cdots \times 3 \times 2 \times 1 = 3628800$. La somme des chiffres du nombre 10! 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, je ne vois qu'une seule solution, qui suit celle proposée dans l'énoncé, et qui consiste à 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 et les convertir en entier avec `int`, et enfin les additionner.

```
>>> import math
>>> print(sum([int(x) for x in str(math.factorial(2013))]))
24021

>>> print(sum([int(x) for x in str(math.factorial(100))]))
648
```

Exercice 7.6 (Module `random` - Matrice)

Construire une matrice carrée d'ordre 10 remplie de 0 et de 1 aléatoirement avec la probabilité d'avoir 1 égale à 0.2.

Compter le nombre d'éléments m_{ij} de la matrice vérifiant les conditions suivantes

$$m_{ij} = 0 \quad \text{et} \quad \left(m_{i-1,j} = 1 \text{ ou } m_{i+1,j} = 1 \text{ ou } m_{i,j-1} = 1 \text{ ou } m_{i,j+1} = 1\right)$$

Correction
Pour obtenir 1 avec une probabilité de 0.2, il suffit de tirer un nombre aléatoire a entre 1 et 5 inclus et de ne considérer que le cas où $a = 1$.

```
from random import randint
dim = 10
M = [ [ 1 if randint(1,5) == 1 else 0 for i in range(dim) ] for j in range(dim) ]

nb=0
for i in range(dim):
    for j in range(dim):
        if i>0 and M[i-1][j] == 1 : nb += 1
        elif i<dim-1 and M[i+1][j] == 1 : nb += 1
        elif j>0 and M[i][j-1] == 1 : nb += 1
        elif j<dim-1 and M[i][j+1] == 1 : nb += 1

print(nb)
```

```
47
```

Exercice 7.7 (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 10 fois et qui affiche "gagné" ou "perdu".

Simuler le jeu 10000 fois et compter combien de fois on a gagné.

Correction

```
from random import randint
T=10
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
Gagné!
Gagné!
Gagné!
Gagné!
Perdu
Gagné!
Perdu
Perdu
Perdu
```

```
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
print(Gagne)
# ou de maniere equivalente
print(sum([1 for t in Tirages if t in [1,5,6]]))
```

```
5062
5062
```

Exercice 7.8 (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 $[1/7, 6/7]$, soit $p = 5/6 - 1/7 = 29/42$.

```
from random import random
T=10000
Tirages=[random() for t in range(T)]
OuiA=[1 for t in Tirages if 1/7<t<5/6]
print("Fréquence théorique =",29/42, "Fréquence moyenne =",sum(OuiA)/T)
```

```
Fréquence théorique = 0.6904761904761905 Fréquence moyenne = 0.6963
```

Exercice 7.9 (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

```
50.39
```

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 7.10 (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. Pour avoir une approximation de π on utilisera le module `math` comme suit :

```
import math
print(math.pi)
```

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 7.11 (Formule d'Héron)
Pour le calcul de l'aire $\mathcal{A}(T)$ d'un triangle T de coté a, b et c, on peut utiliser la formule d'Héron :

$$\mathcal{A}(T) = \sqrt{p(p-a)(p-b)(p-c)}$$

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

```
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("Erone={:.5f}, Exacte={:.5f}".format(He,Ex))

# 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("Erone={:.5f}, Exacte={:.5f}, Erreur={:.5f}".format(He,Ex,Err))
```

```
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 7.12 (Formule de Kahan)

Pour le calcul de l'aire $\mathcal{A}(T)$ d'un triangle T de coté a, b et c avec $a \geq b \geq c$, William Kahan a proposé une formule plus stable que celle d'Héron :

$$S = \frac{1}{4}\sqrt{[a+(b+c)][c-(a-b)][c+(a-b)][a+(b-c)]}.$$

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):
    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("Khan={:.5f}, Exacte={:.5f}".format(He,Ex))

# 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=Khan(a,b,c)
    Ex=a/2*math.sqrt(b**2-(a/2)**2)
    Err=He-Ex
    print("Khan={:.5f}, Exacte={:.5f}, Erreur={:.5f}".format(He,Ex,Err))
```

```
Khan=6.00000, Exacte=6.00000
Khan=0.00000, Exacte=0.00000, Erreur=0.00000
Khan=4.14578, Exacte=4.14578, Erreur=0.00000
Khan=10.82532, Exacte=10.82532, Erreur=-0.00000
```

Exercice 7.13 (Module `random` - Kangourou)

Un kangourou fait habituellement des bonds de longueur aléatoire comprise entre 0 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(0,9)
        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 = 420  bonds.
La distance réelle parcourue est de  2003  mètres.
Pour parcourir d = 2000  mètres  100  fois il faut en moyenne N = 447.23  bonds
```

Exercice 7.14 (sin cos)

En important seulement les fonctions nécessaires du module `math`, écrire un script qui vérifie la formule suivante pour $n = 10$:

$$\sum_{k=0}^{n} \cos(kx) = \frac{1}{2} + \frac{\sin\left(\frac{2n+1}{2}x\right)}{2\sin\left(\frac{x}{2}\right)}$$

Correction

```
>>> from math import sin, cos, pi
>>> n = 50
>>> 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))
0.7423460401504868 0.7423460401504853
```

Exercice 7.15 (Distance)

▷ 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 $d(P,Q)$.

▷ 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 $P=(0,0)$, $Q=(0,1)$ et $R=(1,0)$ alors T = [[0, 0], [0, 1], [1, 0]] , distance(P,Q) = 1.0 , perimetre(T) = 3.414213562373095 , IsEquilateral(T) = False

Correction

```
import math

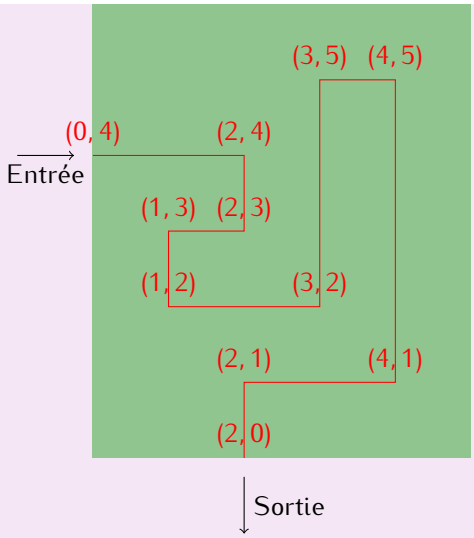
distance = lambda P,Q : math.sqrt((P[0]-Q[0])**2+(P[1]-Q[1])**2)
perimetre = lambda T : distance(T[0],T[1])+distance(T[0],T[2])+distance(T[1],T[2])
IsEquilateral = lambda T : True if abs(distance(T[0],T[1])-distance(T[0],T[2]))<1.e-10 and \
                                   abs(distance(T[0],T[1])-distance(T[1],T[2]))<1.e-1 \
                                else False
# TEST 1
P=[0,0]
Q=[0,1]
R=[1,0]
T=[P,Q,R]
print("T=",T)
print("distance(P,Q) renvoie",distance(P,Q))
print("perimetre(T) renvoie",perimetre(T))
print("IsEquilateral(T) renvoie",IsEquilateral(T))

# TEST 2
P=[0,0]
Q=[1,0]
R=[0.5,math.sqrt(1-0.25)]
T=[P,Q,R]
print("T=",T)
print("distance(P,Q) <",distance(P,Q))
print("perimetre(T) renvoie",perimetre(T))
print("IsEquilateral(T) renvoie",IsEquilateral(T))
```

```
T= [[0, 0], [0, 1], [1, 0]]
distance(P,Q) renvoie 1.0
perimetre(T) renvoie 3.414213562373095
IsEquilateral(T) renvoie False
T= [[0, 0], [1, 0], [0.5, 0.8660254037844386]]
distance(P,Q) < 1.0
perimetre(T) renvoie 3.0
IsEquilateral(T) renvoie True
```

Exercice Bonus 7.16 (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 decida 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 :

```
(0, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 3, 1, 4, 2, 4, 2, 3, 3, 3, 3, 4, 4, 4, 4, 3, 4, 2, 3, 2,
↪ 3, 1, 4, 1, 5, 1, 5, 2, 6, 2, 6, 1, 7, 1, 8, 1, 8, 2, 7, 2, 7, 3, 8, 3, 8, 4, 7, 4, 6,
↪ 4, 6, 3, 5, 3, 5, 4, 5, 5, 5, 6, 6, 6, 6, 5, 7, 5, 8, 5, 8, 6, 7, 6, 7, 7, 8, 7, 8, 8,
↪ 7, 8, 6, 8, 6, 7, 5, 7, 5, 8, 4, 8, 3, 8, 3, 7, 4, 7, 4, 6, 4, 5, 3, 5, 3, 6, 2, 6, 2,
↪ 5, 1, 5, 1, 6, 1, 7, 2, 7, 2, 8, 1, 8, 1, 9, 1, 10, 2, 10, 2, 9, 3, 9, 4, 9, 4, 10, 3,
↪ 10, 3, 11, 4, 11, 4, 12, 3, 12, 2, 12, 2, 11, 1, 11, 1, 12, 1, 13, 2, 13, 2, 14, 1, 14,
↪ 1, 15, 1, 16, 2, 16, 2, 15, 3, 15, 3, 16, 4, 16, 4, 15, 4, 14, 3, 14, 3, 13, 4, 13, 5,
↪ 13, 6, 13, 6, 14, 5, 14, 5, 15, 5, 16, 6, 16, 6, 15, 7, 15, 7, 16, 8, 16, 8, 15, 8, 14,
↪ 7, 14, 7, 13, 8, 13, 8, 12, 8, 11, 7, 11, 7, 12, 6, 12, 5, 12, 5, 11, 6, 11, 6, 10, 5,
↪ 10, 5, 9, 6, 9, 7, 9, 7, 10, 8, 10, 8, 9, 9, 9, 9, 10, 10, 10, 10, 9, 11, 9, 12, 9, 12,
↪ 10, 11, 10, 11, 11, 12, 11, 12, 12, 11, 12, 10, 12, 10, 11, 9, 11, 9, 12, 9, 13, 10,
↪ 13, 10, 14, 9, 14, 9, 15, 9, 16, 10, 16, 10, 15, 11, 15, 11, 16, 12, 16, 12, 15, 12,
↪ 14, 11, 14, 11, 13, 12, 13, 13, 13, 14, 13, 14, 14, 13, 14, 13, 15, 13, 16, 14, 16, 14,
↪ 15, 15, 15, 15, 16, 16, 16, 16, 15, 16, 14, 15, 14, 15, 13, 16, 13, 16, 12, 16, 11, 15,
↪ 11, 15, 12, 14, 12, 13, 12, 13, 11, 14, 11, 14, 10, 13, 10, 13, 9, 14, 9, 15, 9, 15,
↪ 10, 16, 10, 16, 9, 16, 8, 15, 8, 15, 7, 16, 7, 16, 6, 16, 5, 15, 5, 15, 6, 14, 6, 14,
↪ 5, 13, 5, 13, 6, 13, 7, 14, 7, 14, 8, 13, 8, 12, 8, 12, 7, 11, 7, 11, 8, 10, 8, 9, 8,
↪ 9, 7, 10, 7, 10, 6, 9, 6, 9, 5, 10, 5, 11, 5, 11, 6, 12, 6, 12, 5, 12, 4, 12, 3, 11, 3,
↪ 11, 4, 10, 4, 9, 4, 9, 3, 10, 3, 10, 2, 9, 2, 9, 1, 10, 1, 11, 1, 11, 2, 12, 2, 12, 1,
↪ 13, 1, 14, 1, 14, 2, 13, 2, 13, 3, 13, 4, 14, 4, 14, 3, 15, 3, 15, 4, 16, 4, 16, 3, 16,
↪ 2, 15, 2, 15, 1, 16, 1, 17, 1)
```

Aide Hercule en calculant pour lui la distance qu'il aura à parcourir dans le bois, à la poursuite de la biche.

Source : https://callicode.fr/pydefis/Herculito03Biche/txt

Exercice 7.17 (Approximation de π)

Soit

$$s(N) \overset{\text{déf}}{=} \sum_{n=1}^{N} \frac{1}{n^2}$$

On peut montrer que

$$\lim_{N\to+\infty} s(N) = \frac{\pi^2}{6}.$$

Écrire un script qui calcule $s(N)$ jusqu'à ce que cette somme soit une approximation de $\frac{\pi^2}{6}$ à 10^{-5} près. Que

vaut N ?
La valeur de π considérée comme exacte sera celle du module `math`.

Correction

```
from math import pi
n=0
somme=0
s = lambda n : 1/n**2
while abs(somme-pi**2/6)>1.e-5:
    n+=1
    somme+=s(n)
print(f"N={n}, somme={somme}, pi^2/6={pi**2/6}")
```

```
N=100000, somme=1.6449240668982423, pi^2/6=1.6449340668482264
```

Exercice 7.18 (Approximation de π)

Comme π vérifie

$$\pi = \lim_{N\to+\infty} \sum_{n=0}^{N} 16^{-n} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right)$$

on peut calculer une approximation de π en sommant les N premiers termes, pour N assez grand.

- ▷ Écrire une fonction pour calculer les sommes partielles de cette série (*i.e.* pour $n = 0 \dots N$ avec N donné en paramètre).
- ▷ Pour quelles valeurs de N obtient-on une approximation de π aussi précise que celle fournie par la variable π ?

Correction

```
from math import pi

def s(n):
    return ( 4/(8*n+1) - 2/(8*n+4) - 1/(8*n+5) - 1/(8*n+6) )*(1/16)**n

def piapproche(N):
    return sum( [ s(n) for n in range(N)] )

N=0
while abs(pi-piapproche(N))>0:
    N+=1

print("       N  = ",N)
print("pi approx = ",piapproche(N))
print(" math.pi  = ",pi)
```

```
       N  =  11
pi approx =  3.141592653589793
 math.pi  =  3.141592653589793
```

Pour $N = 11$ 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.

Exercice 7.19 (Approximation de π)

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 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 dans l'intervalle $[0,1]$, puis on calcule le nombre $m \leq N$ de ceux qui se trouvent dans le premier quart du cercle unité. π est la limite de la suite $4m/N$ lorsque $N \to +\infty$. Écrire un programme pour calculer cette suite et observer comment évolue l'erreur quand N augmente.

Correction

```
from math import pi
import random
N=1000 # Numéro du tir
m=0    # Nombre de tirs 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 ajoute 1
myPi=4*m/N
err=abs(pi-myPi)
print(f"pi={pi:g}, myPi={myPi:g}, error={err:g}")
```

On exécute le programme pour différentes valeurs de N. Plus N est grand, meilleure est l'approximation de π. Par exemple, pour $N = 1000$ on obtient

```
pi=3.14159, myPi=3.116, error=0.0255927
```

Naturellement, comme les nombres sont générés aléatoirement, les résultats obtenus pour une même valeur de N peuvent changer à chaque exécution. Cette méthode n'est pas très efficace, il faut beaucoup de tirs pour obtenir le deux premières décimales de π.

Exercice 7.20 (Méthode de Monte-Carlo)

La méthode de Monte-Carlo (du nom des casinos, pas d'une personne) est une approche probabiliste permettant d'approcher la valeur d'une intégrale. 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 détermine une approximation de J par la méthode de Monte-Carlo.
- ▷ 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

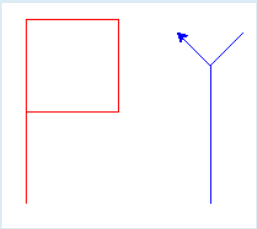
g = lambda x: 1
print("Calcul approché par Montecarlo =",montecarlo(g,0,1,100))
print("Calcul approché par ScyPy =", integrate.quad(g,0,1)[0])

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 = 0.27779293749982564
Calcul approché par ScyPy = 0.25
```

Exercice 7.21 (Turtle - lettres)

Écrire un script qui trace l'image suivante :



Correction

```
from turtle import *

color('red')
left(90)
forward(200)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)

up()

color('blue')
goto(200,0)
down()
right(90)
forward(150)
right(45)
forward(50)
up()
backward(50)
down()
left(90)
forward(50)

exitonclick()
```

Exercice 7.22 (Turtle - polygones)

Écrire une fonction `polygone(n,longueur)` qui trace un polygone régulier à n côtés et d'une longueur de côté donnée (l'angle pour tourner est alors de $360/n$ degrés). Tester la fonction pour $n = 3, 4, 5, \ldots, 14$.

Correction

J'ai ajouté une liste de couleurs pour voir changer la couleur de chaque polygone.

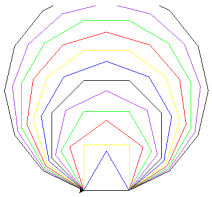
```
from turtle import *
import random

def polygone(n,longueur):
```

```
    for i in range(n):
        forward(longueur)
        left(360/n)

for n in range(3,15):
    color('#%06x' % random.randint(0, 2**24 - 1))
    polygone(n,100)

exitonclick()
```



Exercice 7.23 (Turtle - marcheur ivre à Manhattan)

Supposons qu'un promeneur complètement ivre se trouve à Manhattan, New-York (U.S.A.). Ce quartier de la ville est quadrillé par de longues avenues du nord au sud, et rues d'est en ouest, un peu comme les feuilles de mon bloc notes. À chaque fois que le marcheur arrive au niveau d'un croisement il fait une pause et comme il est vraiment ivre, il repart aléatoirement dans une des quatre directions possibles, avec une chance sur quatre pour chacune d'elle.

Simuler son déplacement aléatoire de la façon suivante.

1. Créer une variable `d` aléatoirement choisi dans $[1;4]$.
2. Si `d=1` le marcheur ne tourne pas et avance de 5,
3. Si `d=2` le marcheur tourne à gauche et avance de 5,
4. Si `d=3` le marcheur se retourne et avance de 5,
5. Si `d=4` le marcheur tourne à droite et avance de 5.
6. Répéter 1000 fois ce processus en affectant une couleur différente à chaque sens de déplacement.

Correction

```
from turtle import *
import random

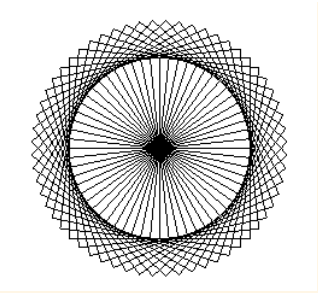
couleurs=['red','green','magenta','blue']

for n in range(1000):
    d=random.randint(1,5)
    color(couleurs[d-1])
    if d==1:
        right(0)
    elif d==2:
        right(-90)
    elif d==3:
        right(180)
    elif d==4:
        right(90)
    else:
        print('Error')
    forward(5)

exitonclick()
```

Exercice Bonus 7.24 (Turtle - carrés en cercle)

Écrire un script qui dessine 60 carrés, chacun tourné de 6 degrés, pour obtenir :



Correction

```
from turtle import *

def square():
    for i in range(4):
        forward(100)
        right(90)

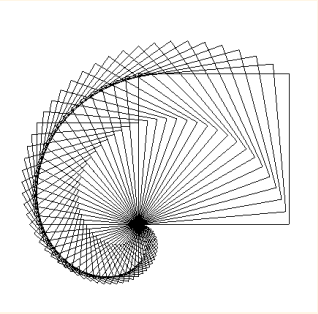
speed(0)

for n in range(60):
    right(6)
    square()

exitonclick()
```

☆ Exercice Bonus 7.25 (Turtle - spirale)

Écrire un script qui dessine 55 carrés, chacun tourné de 5 degrés, le premier de coté 5, le deuxième 10 etc. pour obtenir :



Correction

```
from turtle import *

def square(cote):
    for i in range(4):
        forward(cote)
        right(90)

speed(0)
cote=5
for n in range(54):
        right(5)
        cote+=5
        square(cote)

exitonclick()
```

☆ Exercice Bonus 7.26 (Sympy – devine le résultat)

Tester les codes suivants :

1.
```
from sympy import *
var('x')
expr=(x**3 + x**2 - x - 1)/(x**2 + 2*x
↪ + 1)
expr2=simplify(expr)
print(expr,"=",expr2)
```
2.
```
from sympy import *
var('x')
expr=x**4/2+5*x**3/12-x**2/3
expr2=factor(expr)
print(expr,"=",expr2)
```
3.
```
from sympy import *
var('x')
expr=x**4-1
sol=solve(expr)
print(sol)
```
4.
```
from sympy import *
var('x,a,b,c')
expr = a*x**2 + b*x + c
sol=solve(expr, x)
print(sol)
```
5.
```
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. $\dfrac{x^4}{2}+\dfrac{5x^3}{12}-\dfrac{x^2}{3} = \dfrac{x^2}{12}(2x-1)(3x+4)$
3. $[-1,\quad 1,\quad -i,\quad i]$
4. $\left[\dfrac{1}{2a}\left(-b+\sqrt{-4ac+b^2}\right),\quad -\dfrac{1}{2a}\left(b+\sqrt{-4ac+b^2}\right)\right]$
5. $1,\ \infty,\ 0,\ 1,\ 2xy,\ -\cos(x)$

☆ Exercice Bonus 7.27 (Sympy – calcul formel d'une dérivée)

Calculer $\partial_x f(x,y)$ pour

$$f(x,y) = \frac{ay-1}{(bx-1)^c}.$$

Correction

```
from sympy import *
var('x,y,a,b,c')
f=(a*y-1)/(b*x-1)**c
g=diff(f,x,1)
print(g)
```

$$-\frac{bc\,(bx-1)^{-c}}{bx-1}(ay-1)$$

☆ Exercice Bonus 7.28 (Sympy – composition de fonctions)

Calculer $f(m,f(n,m))$ si

$$f(a,b) = a^2-b.$$

Correction

```
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 7.29 (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=f.subs(x,1)-P_1
eq2=f.subs(x,2)-P_2
fp=sym.diff(f,x)
eq3=fp.subs(x,2)-1
sol=sym.solve([eq1,eq2,eq3],[a,b,c])
print( sol )
```

$$\{a : P_1 - P_2 + 1, \quad b : -4P_1 + 4P_2 - 3, \quad c : 4P_1 - 3P_2 + 2\}$$

☆ Exercice Bonus 7.30 (Sympy – calcul des paramètres)

Soit $f : \mathbb{R} \to \mathbb{R}$ 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 )
```

$$\{a : 2c, \quad b : 0\}$$

☆ Exercice Bonus 7.31 (Sympy – calcul de paramètres)

Soit $f : \mathbb{R} \to \mathbb{R}$ 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 )
```

$$\left[-\frac{a}{2}, \quad a\right]$$

Chapitre 8.

Tracé de courbes et surfaces

Le tracé de courbes scientifiques peut se faire à l'aide du package `matplotlib`. Ce package très complet contient différents modules, notamment les modules `pylab` et `pyplot` : `pylab` associe les fonctions de `pyplot` (pour les tracés) avec les fonctionnalité du module `numpy` pour obtenir un environnement très proche de celui de MATLAB/Octave, très répandu en calcul scientifique.

On peut importer le module

- ▷ soit par l'instruction `from matplotlib.pylab import *` (qui importe aussi le module `numpy` sans alias) ;
- ▷ soit par l'instruction `import matplotlib.pyplot as plt`.

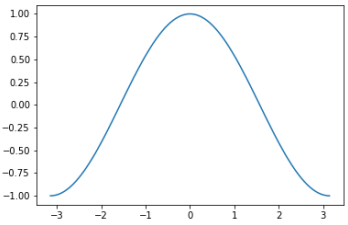
Dans le premier cas on pourra utiliser les commande "à la MATLAB", comme par exemple

```
from matplotlib.pylab import *
x=linspace(-pi,pi,101)
plot(x,cos(x))
```

Dans le deuxième cas il faudra faire précéder toutes les instruction `matplotlib` par `plt` (et celles `numpy` par `np`), comme par exemple

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-np.pi,np.pi,101)
plt.plot(x,np.cos(x))
```

Dans les deux cas on obtiendra l'image suivante :



Étant donné qu'avec les modules qu'on utilise dans cet ECUE il n'y a pas de risque de conflits des `namespace`, **dans ce polycopié on utilisera toujours `pylab`**.

8.1. Courbes

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.

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 :

```
from matplotlib.pylab import *

f = lambda x : # à compléter
a,b,n = # à compléter

xx=linspace(a,b,n+1)
yy=[f(x) for x in xx]
```

```
plot(xx,yy)
show()
```

Pour générer les points x_i on peut utiliser

- ▷ soit l'instruction `linspace(a,b,n+1)` qui construit la liste de $n+1$ éléments

$$[a, a+h, a+2h, \ldots, b = a+nh] \quad \text{avec } h = \frac{b-a}{n}$$

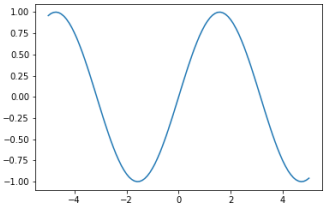
- ▷ soit l'instruction `arange(a,b,h)` qui construit la liste de $n = E(\frac{b-a}{h}) + 1$ éléments

$$[a, a+h, a+2h, \ldots, a+nh]$$

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.

Voici un exemple avec une sinusoïde :

```
from matplotlib.pylab import *
x = linspace(-5,5,101) # x = [-5,-4.9,-4.8,...,5]
# x = arange(-5,5,0.1) # idem
y = sin(x)             # operation is broadcasted to all
 ↪  elements of the array
plot(x,y)
show()
```



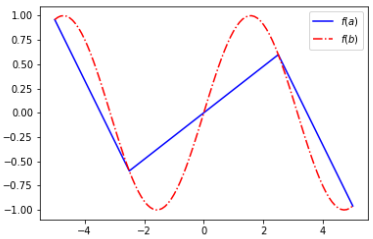
On obtient une courbe sur laquelle on peut zoomer, modifier les marges et sauvegarder dans différents formats.

8.1.1. Plusieurs courbes sur le même repère

On peut tracer plusieurs courbes sur la même figure.

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 :

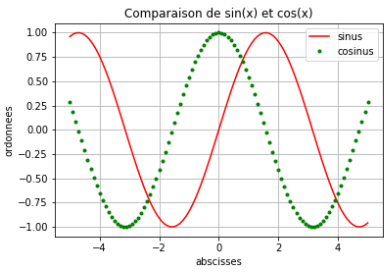
```
from matplotlib.pylab import *
a = linspace(-5,5,5)
fa = sin(a) # matplotlib importe numpy qui
            # redefinie les fonctions de base
            # pour pouvoir les appliquer
            # a un vecteur
plot(a,fa,'b-',label="$f(a)$")
b = linspace(-5,5,101)
fb = sin(b)
plot(b,fb,'r-.',label="$f(b)$" )
legend()
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.

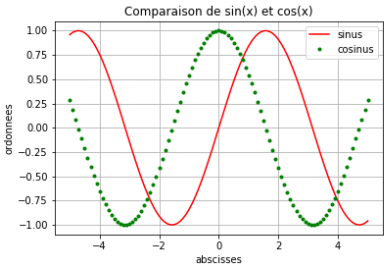
Par exemple, dans le code ci-dessous `"r-"` indique que la première courbe est à tracer en rouge (red) avec un trait continu, et `"g."` que la deuxième est à tracer en vert (green) avec des points.

```
from matplotlib.pylab import *
x = linspace(-5,5,101)
y1 = sin(x)
y2 = cos(x)
plot(x,y1,"r-",x,y2,"g.")
legend(['sinus','cosinus'])
xlabel('abscisses')
ylabel('ordonnees')
title('Comparaison de sin(x) et cos(x)')
grid(True)
show()
```



soit encore

```
from matplotlib.pylab import *
x = linspace(-5,5,101)
y1 = sin(x)
y2 = cos(x)
plot(x,y1,"r-",label=('sinus'))
plot(x,y2,"g.",label=('cosinus'))
legend()
xlabel('abscisses')
ylabel('ordonnees')
title('Comparaison de sin(x) et cos(x)')
grid(True)
show()
```



Quelques options de pylab :

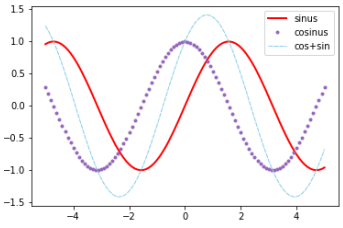
	linestyle=		color=		marker=
-	solid line	r	red	.	points
-	dashed line	g	green	,	pixel
:	dotted line	b	blue	o	filled circles
-.	dash-dot line	c	cyan	v	triangle down
(0,(5,1))	densely dashed	m	magenta	^	triangle up
		y	yellow	>	triangle right
		w	white	<	triangle left symbols
		k	black	*	star
				+	plus
				s	square
				p	pentagon
				x	x
				X	x filled
				d	thin diamond
				D	diamond

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

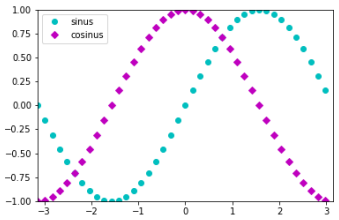
Voici un exemple d'utilisation :

```
from matplotlib.pylab import *
x = linspace(-5,5,101)
y1 = sin(x)
y2 = cos(x)
y3 = sin(x)+cos(x)
plot(x,y1,color='r'       ,ls='-'       ,linewidth=2           ,label=('sinus'))
plot(x,y2,color='purple'  ,ls=' '       ,lw=1        ,marker='.',label=('cosinus'))
plot(x,y3,color='skyblue',ls=(0,(5,1)),lw=1                    ,label=('cos+sin'))
legend()
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` :

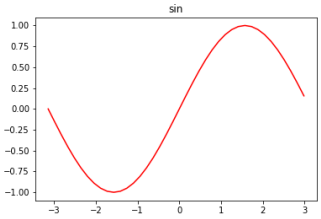
```
from matplotlib.pylab import *
x = arange(-pi,pi,0.05*pi)
plot(x,sin(x),'co',x,cos(x),'mD')
legend(['sinus','cosinus'],loc='upper left')
axis([-pi, pi, -1, 1]); # axis([xmin, xmax, ymin,
 ↪  ymax])
show()
```

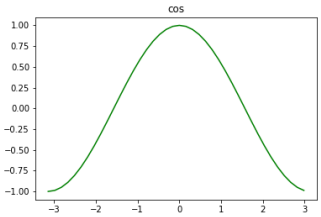


8.1.2. Plusieurs “fenêtres” graphiques

Avec `figure()` on génère deux fenêtres contenant chacune un graphe :

```
from matplotlib.pylab import *
x = arange(-pi,pi,0.05*pi)
figure(1)
plot(x, sin(x), 'r')
figure(2)
plot(x, cos(x), 'g')
show()
```





8.1.3. Plusieurs repères dans la même fenêtre

La fonction `subplot(x,y,z)` subdivise la fenêtre sous forme d'une matrice `(x,y)` et chaque case est numérotée, `z` étant 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.

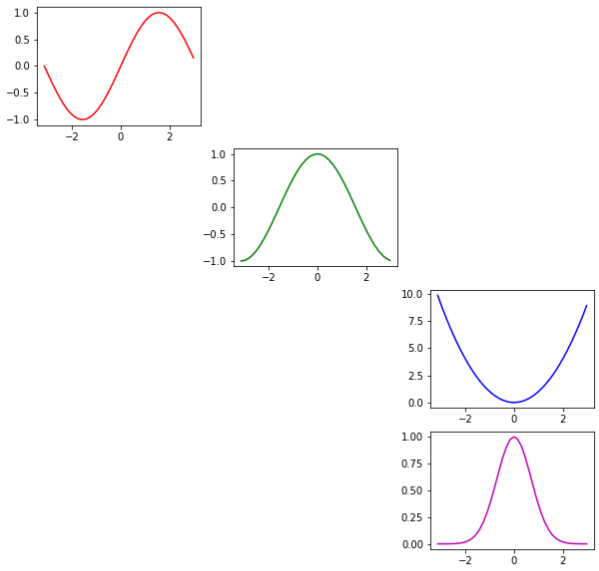
```
from matplotlib.pylab import *
x = arange(-pi,pi,0.05*pi)

subplot(4,3,1)
plot(x, sin(x), 'r')

subplot(4,3,5)
plot(x, cos(x), 'g')

subplot(4,3,9)
plot(x, x*x, 'b')

subplot(4,3,12)
plot(x, exp(-x*x), 'm')
```



8.1.4. Régression : polynôme de meilleure approximation

La fonction au cœur de la régression est `polyfit` du module `numpy`. Pour l'utiliser il faut donc importer le module numpy, ce qui est automatiquement fait avec `matplotlib.pylab`. Elle s'utilise de la façon suivante :

```
import numpy as np
a,b,c = 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. Ici on veut donc approcher le nuage par un polynôme qui sera de la forme $ax^2 + bx + c$. `np.polyfit` renvoie donc les coefficients a, b, c (dans cet ordre).

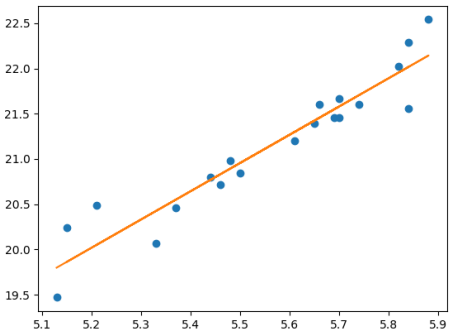
Exemple de régression linéaire :

```
from matplotlib.pylab import *

# 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=polyfit(xx,yy,1)
plot(xx,yy,"o")
plot(x,a*x+b)

show()
```



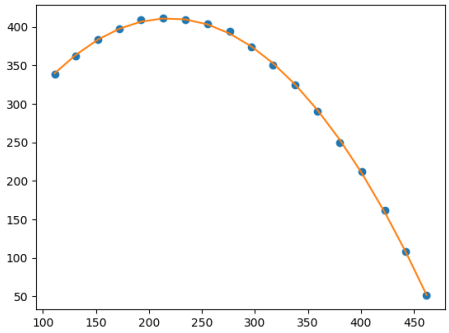
Exemple de régression d'ordre 2 :

```
from matplotlib.pylab import *

# Les listes des abscisses et ordonnées :
xx=[111, 131, 152, 172, 192, 213, 234, 255, 276, 297, 317, 338, 359, 380, 401, 423, 442, 462]
yy=[339, 362, 384, 398, 409, 411, 409, 404, 394, 374, 350, 325, 290, 250, 212, 162, 108, 51]

a,b,c=polyfit(xx,yy,2)
plot(xx,yy,"o")
plot(x,a*x**2+b*x+c)

show()
```



8.2. ★Surfaces

La représentation graphique de l'évolution d'une fonction f de deux variables x et y n'est pas une tâche facile, surtout si le graphique en question est destiné à être imprimé.

Commençons par considérer l'exemple simple d'une fonction de la forme :

$$f(x, y) = x^2 + y^2$$

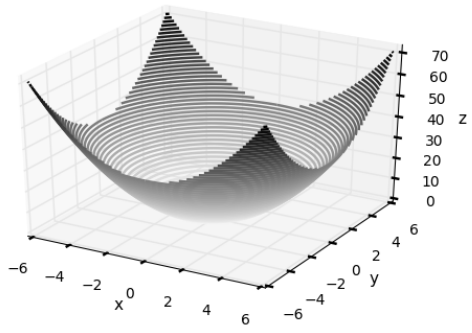
que l'on déclare en python par `f = lambda x,y : x**2+y**2`

Le tracé de cette fonction va nécessiter la création d'un maillage bidimensionnel permettant de stocker l'intervalle de chacune des variables. La fonction destinée à cela s'appelle `meshgrid` (incluse dans le module `numpy` comme les fonctions `linspace`, `arange` etc.). On construit donc le maillage en question sur le rectangle $[-6; 6] \times [-6; 6]$. La fonction `meshgrid` fait appel dans ce cas à deux fonctions `linspace` pour chacune des variables. `z` est ici un objet `array` qui contient les valeurs de la fonction f sur chaque nœud du maillage.

Pour tracer la surface de f on utilisera la fonction `contour3D` du module `mpl_toolkits` :

```
from matplotlib.pylab import *
xx=linspace(-6,6,101)
yy=linspace(-6,6,101)
X,Y = meshgrid(xx,yy)
Z = X**2+Y**2

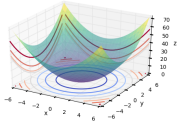
from mpl_toolkits import mplot3d
ax = axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```



On peut ajouter des courbes de niveau dans chaque direction :

```
from matplotlib.pylab import *
xx=linspace(-6,6,101)
yy=linspace(-6,6,101)
X,Y = meshgrid(xx,yy)
Z = X**2+Y**2

from mpl_toolkits import mplot3d
ax = axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=4,
 ↪  cstride=4,cmap='viridis',edgecolor='none', alpha=0.5)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
ax.contour(X, Y, Z, zdir='z', offset=0, cmap=cm.coolwarm)
ax.contour(X, Y, Z, zdir='x', offset=-2*pi, cmap=cm.coolwarm)
ax.contour(X, Y, Z, zdir='y', offset=2*pi, cmap=cm.coolwarm)
```



On peut changer l'angle de vue :

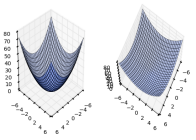
```
from matplotlib.pylab import *
xx=linspace(-6,6,101)
yy=linspace(-6,6,101)
X,Y = meshgrid(xx,yy)
Z = X**2+Y**2

from mpl_toolkits import mplot3d

ax1=subplot(1,2,1, projection='3d')
ax1.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
ax1.view_init(30, 45)

ax2=subplot(1,2,2, projection='3d')
ax2.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
ax2.view_init(70, 30)

fig.tight_layout()
```



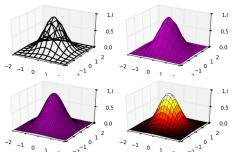
On peut changer l'apparence de la surface :

```
from matplotlib.pylab import *
xx=linspace(-6,6,101)
yy=linspace(-6,6,101)
X,Y = meshgrid(xx,yy)
Z = X**2+Y**2

from mpl_toolkits import mplot3d

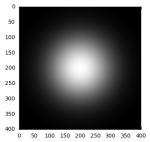
L, n = 2, 400
x = linspace(-L, L, n)
y = x.copy()
X, Y = meshgrid(x, y)
Z = exp(-(X**2 + Y**2))

fig, ax = subplots(nrows=2, ncols=2, subplot_kw={'projection': '3d'})
ax[0,0].plot_wireframe(X, Y, Z, rstride=40, cstride=40)
ax[0,1].plot_surface(X, Y, Z, rstride=40, cstride=40, color='m')
ax[1,0].plot_surface(X, Y, Z, rstride=12, cstride=12, color='m')
ax[1,1].plot_surface(X, Y, Z, rstride=20, cstride=20, cmap=cm.hot)
for axes in ax.flatten():
    axes.set_xticks([-2, -1, 0, 1, 2])
    axes.set_yticks([-2, -1, 0, 1, 2])
    axes.set_zticks([0, 0.5, 1])
fig.tight_layout()
```



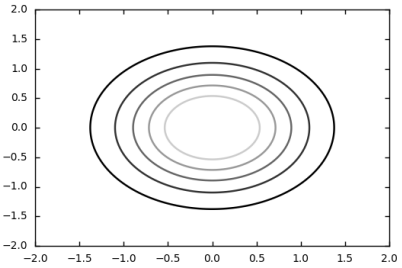
On peut regarder la projection de la surface sur le plan d'équation $z = 0$:

```
imshow(Z)
```

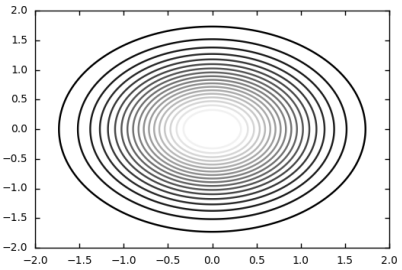


Pour tracer l'évolution de f en lignes de niveaux on utilisera les fonctions `contour` et `contourf` avec comme arguments les variables x et y, les valeurs de z correspondantes ainsi que le nombre de lignes de niveaux choisit.

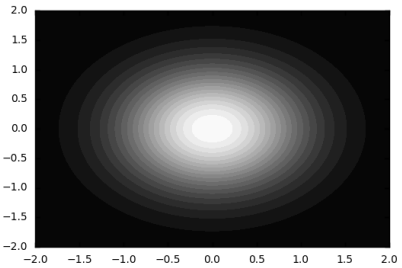
```
h = contour(X,Y,Z)
show()
```



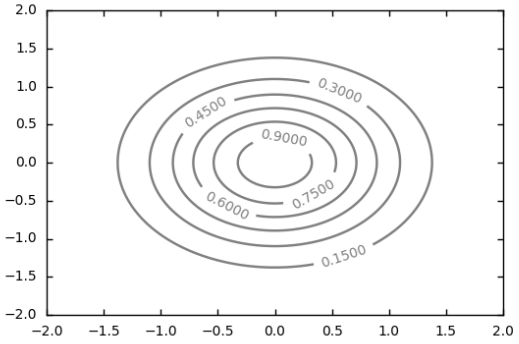
```
h = contour(X,Y,Z,20)
show()
```



```
h = contourf(X,Y,Z,20)
show()
```



Par défaut, ces courbes de niveaux sont colorées en fonction de leur “altitude” z correspondante. S'il n'est pas possible d'utiliser de la couleur, on peut imposer une couleur uniforme pour le graphe et utiliser des labels sur les lignes de niveaux afin de repérer leurs altitudes. La fonction en question s'appelle `clabel` et prend comme principal argument la variable graphe précédente. La première option `inline=1` impose d'écrire les valeurs sur les lignes de niveaux, les deux options suivantes gérant la taille de la police utilisée et le format d'écriture des valeurs.



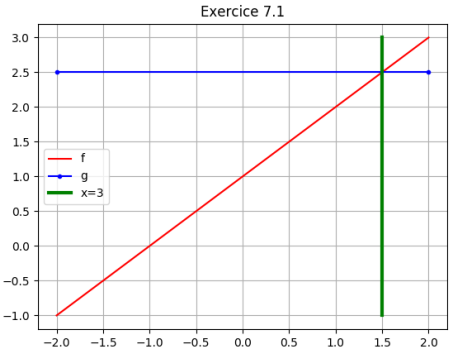

```
graphe4 = contour(x,y,z,20,colors='grey')
clabel(graphe4,inline=1,fontsize=10,fmt='%3.2f')
show()
```

8.3. Exercices

Exercice 8.1 (Tracer des droites)
Tracer dans le même repère les droites suivantes (sur $[-2;2]$) :

- ▷ $f(x) = x + 1$
- ▷ $g(x) = 2.5$ (en utilisant seulement deux points)
- ▷ $x = 3$

Ajouter une grille, la légende, un titre.




```
from matplotlib.pylab import *

xx=linspace(-2,2,101)

f = lambda x : x+1
ff=[f(x) for x in xx]
plot(xx,ff,'r-',label=('f'))

g = lambda x : 2.5
gg=[g(x) for x in xx]
plot(xx,gg,'r-',label=('g'))

yy=linspace(-2,2,101)
h = lambda y : 3
hh=[h(y) for y in yy]
plot(hh,yy,'g-',lw=3,label=('x=3'))

title("Exercice 7.1")
grid()
legend(loc="best")
show()
```

Étant donné que pour définir une droite il suffit d'imposer le passage par deux points, on peut tracer les mêmes graphes comme ci-dessous :

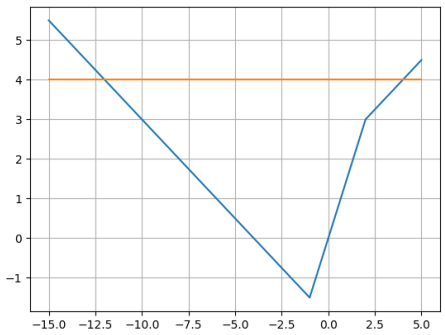
```
from matplotlib.pylab import *
plot([-2,2],[-1,3],'r-',label=('f'))
plot([-2,2],[2.5,2.5],'b.-',label=('g'))
plot([1.5,1.5],[-1,3],'g-',lw=3,label=('x=3'))
show()
```

Exercice 8.2 (Tracer une fonction définie par morceaux)
Tracer le graphe de la fonction

$$f : \mathbb{R} \to \mathbb{R}$$
$$x \mapsto |x+1| - \left|1 - \frac{x}{2}\right|$$

Résoudre d'abord graphiquement puis analytiquement $f(x) = 0$ et $f(x) \leq 4$.

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 $f(x) \leq 4$ pour $-12 \lesssim x \lesssim 4$. On affiche alors $f(-12)$ et $f(4)$.




```
from matplotlib.pylab import *
f = lambda x : abs(x+1)-abs(1-x/2)
xx = linspace(-15,5,101)
yy = [f(x) for x in xx]
plot(xx,yy)
plot([-15,5],[4,4])
grid()
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
```

Exercice 8.3 (Résolution graphique d'une équation)

Soit la fonction

$$f : [-10, 10] \to \mathbb{R}$$
$$x \mapsto \frac{x^3 \cos(x) + x^2 - x + 1}{x^4 - \sqrt{3}x^2 + 127}$$

1. Tracer le graphe de la fonction f en utilisant seulement les valeurs de $f(x)$ lorsque la variable x prend successivement les valeurs $-10, -9.2, -8.4, \ldots, 8.4, 9.2, 10$ (*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

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 $\alpha = 1.96289995$.

```
from matplotlib.pylab import *
f = lambda x: (x**3*cos(x)+x**2-x+1)/(x**4-sqrt(3)*x**2+127)

figure(num=None, figsize=(20, 10))

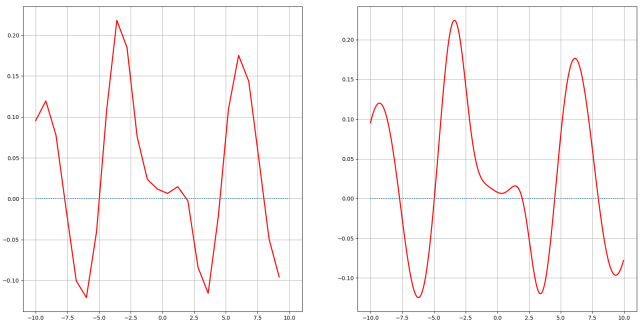
subplot(1,2,1)
xx=arange(-10,10,0.8)
yy=[f(x) for x in xx]
plot(xx,yy,'r-',lw=2)
plot([-10,10],[0,0],':')
grid()

subplot(1,2,2)
xx=arange(-10,10,0.05)
yy=[f(x) for x in xx]
plot(xx,yy,'r-',lw=2)
plot([-10,10],[0,0],':')
grid()

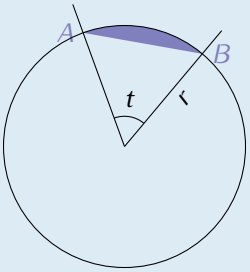
show()

from scipy.optimize import fsolve
print(fsolve(f,1.9))
```

```
[1.96289995]
```



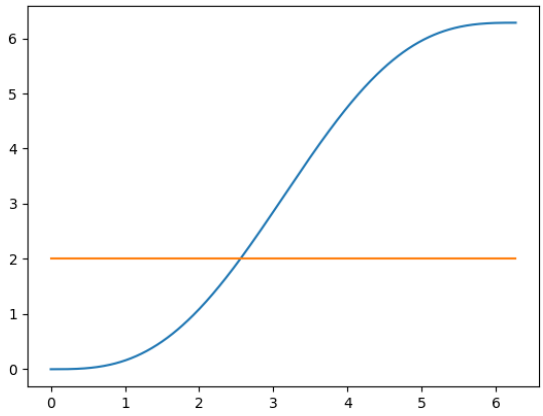
Exercice 8.4 (Résolution graphique d'une équation)



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 ?




```
from matplotlib.pylab import *
a=4
r=2
tt=arange(0,2*pi,pi/180)
rhs=[t-sin(t) for t in tt]
lhs=[2*a/r**2 for t in tt]
plot(tt,rhs,tt,lhs)
show()

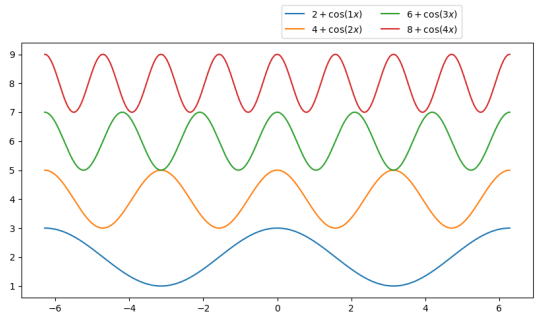
from scipy.optimize import fsolve
f=lambda t: t-sin(t)-2*a/r**2
print(fsolve(f,2.5))
```

```
[2.55419595]
```

Le graphe montre que la solution est entre 2 et 3. On peut alors calculer une solution approchée avec `fsolve`.

Exercice 8.5 (Tracer plusieurs courbes)

Tracer dans le même repère le graphe des fonctions $f_n(x) = 2n + \cos(nx)$ pour $n = 1, 2, 3, 4$.




```
from matplotlib.pylab import *

figure(num=None, figsize=(10, 5))

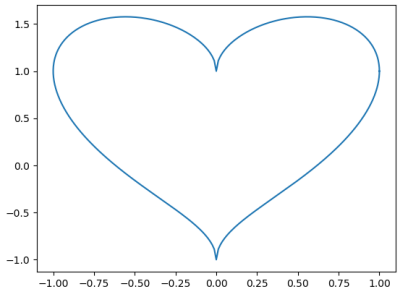
f = lambda x,n : 2*n+cos(n*x)
xx=linspace(-2*pi,2*pi,1001)
for n in range(1,5):
    yy=[f(x,n) for x in xx]
    plot(xx,yy, label=fr'${2*n}+\cos({n}x)$')

legend(bbox_to_anchor=(0.5, 1),ncol=2);
show()
```

Exercice 8.6 (Courbe parametrée)

Tracer la courbe mystère suivante pour $t \in [0; 2\pi]$:

$$\begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t) + \sqrt{|\cos(t)|} \end{cases}$$

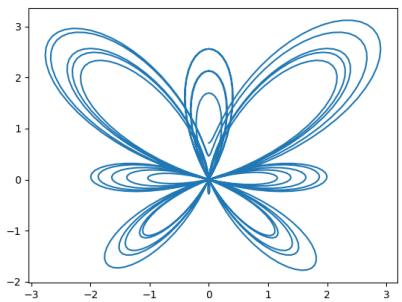


```
from matplotlib.pylab import *
x = lambda t : cos(t)
y = lambda t : sin(t)+sqrt(abs(cos(t)))
tt=linspace(0,2*pi,501)
xx=[x(t) for t in tt]
yy=[y(t) for t in tt]
plot(xx,yy)
show()
```

Exercice 8.7 (Courbe parametrée)

Tracer la courbe papillon ($t \in [0; 10\pi]$) :

$$\begin{cases} x(t) = \sin(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right) \\ y(t) = \cos(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right) \end{cases}$$

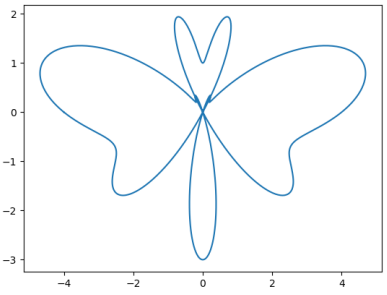


```
from matplotlib.pylab import *
x = lambda t :
 ↪  sin(t)*(exp(cos(t))-2*cos(4*t)-(sin(t/12))**2)
y = lambda t :
 ↪  cos(t)*(exp(cos(t))-2*cos(4*t)-(sin(t/12))**2)
tt=linspace(0,10*pi,1001)
xx=[x(t) for t in tt]
yy=[y(t) for t in tt]
plot(xx,yy)
show()
```

Exercice 8.8 (Courbe polaire)

Tracer la courbe papillon ($t \in [0; 2\pi]$) :

$$\begin{cases} x(t) = r(t)\cos(t) \\ y(t) = r(t)\sin(t) \end{cases} \quad \text{avec } r(t) = \sin(7t) - 1 - 3\cos(2t).$$



```
from matplotlib.pylab import *
r = lambda t : sin(7*t)-1-3*cos(2*t)
x = lambda t : r(t)*cos(t)
y = lambda t : r(t)*sin(t)
tt=linspace(0,2*pi,1001)
xx=[x(t) for t in tt]
yy=[y(t) for t in tt]
plot(xx,yy)
show()
```

Exercice 8.9 (Proies et prédateurs)

Le mathématicien Volterra a proposé en 1926 un modèle décrivant l'évolution conjointe des sardines et des requins constatée par des pêcheurs de l'Adriatique : les effectifs des deux espèces variaient de façon périodique en fonction du temps, avec la même période mais en étant décalées dans le temps. Plus généralement, il s'agit de l'étude de l'évolution de deux populations (proies et prédateurs) ayant une incidence l'une sur l'autre.

Les équations vérifiées par ces effectifs sont des équations différentielles. Nous ne traiterons ici que le problème discrétisé à l'aide de suites.

On note e le nombre de proies et c le nombre de prédateurs à l'instant t. En l'absence de prédateurs, on note A le taux de reproduction des proies entre les instants t_n et t_{n+1}. Il est supposé constant dans ce modèle : les proies sont supposées avoir une source illimitée de nourriture et se reproduire si elles ne sont soumises à aucune prédation. En l'absence des proies, on note C le taux de mortalité des prédateurs entre les instants t_n et t_{n+1} Il est supposé constant dans ce modèle : il représente la mort naturelle des prédateurs. Le taux de mortalité des proies due aux prédateurs est supposé, dans ce modèle, proportionnel au nombre de prédateurs : B est le coefficient de proportionnalité. Le taux de reproduction des prédateurs en fonction des proies mangées est supposé, là encore, proportionnel au nombre de proies : D est le coefficient de proportionnalité. On a donc :

$$\begin{cases} \frac{e_{n+1}-e_n}{e_n} = A - Bc_n, & \text{équation des proies,} \\ \frac{c_{n+1}-c_n}{c_n} = -C + Du_n, & \text{équation des prédateurs.} \end{cases}$$

La variation du nombre de proies est donnée par sa propre croissance moins le taux de prédation qui leur est appliqué, ce qui donne l'équation des proies. La variation de la population de prédateurs en tant que croissance de cette population, diminuée du nombre de morts naturelles, ce qui donne l'équation des prédateurs. On obtient ainsi le système d'équations du modèle de Volterra :

$$\begin{cases} e_{n+1} = e_n(1 + A - Bc_n), & \text{équation des proies,} \\ c_{n+1} = c_n(1 - C + Du_n), & \text{équation des prédateurs.} \end{cases}$$

Afficher l'évolution des deux populations entre $n = 0$ et $n = 1000$ si $e_0 = 1000$, $c_0 = 20$, $A = 0.1$, $B = C = 0.01$ et $D = 0.00002$.

Correction

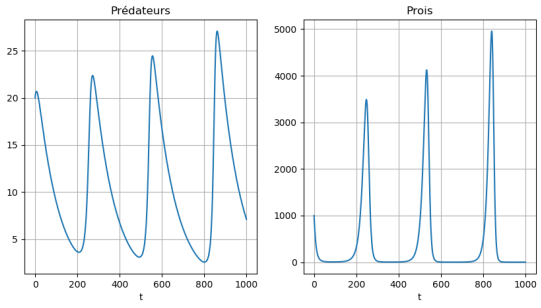
```
from matplotlib.pylab import *
e,c=1000,20
Er=[e]
Ca=[c]
A,B,C,D=0.1,0.01,0.01,0.00002
```

```
for n in range(1000):
    e,c =  e*(1+A-B*c), c*(1-C+D*e)
    Ca.append(c)
    Er.append(e)

subplot(1,2,1)
plot(range(len(Ca)),Ca,label="Predateurs")
xlabel('t')
grid()
title('Prédateurs')

subplot(1,2,2)
plot(range(len(Er)),Er,label="Prois")
title('Prois')
xlabel('t')
grid()

show()
```



Exercice 8.10 (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 $\frac{364}{365} \times \frac{363}{365} 5 \times \frac{362}{365} \times \cdots \times \frac{365-k+1}{365}$.

On obtient la formule de récurrence

$$\begin{cases} P_1 = 1, \\ P_{k+1} = \frac{365-k+1}{365} P_k. \end{cases}$$

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%.

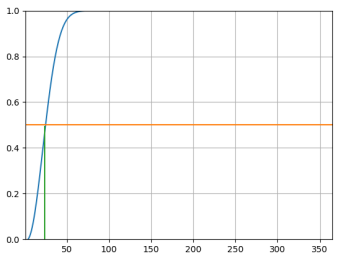
Source : blog http ://eljjdx.canalblog.com/archives/2007/01/14/3691670.html

Correction

Montrons que 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 !)

```
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")
plot(nn,nP, [min(nn),max(nn)],[seuil,seuil],[indice,indice],[0,P[indice]])
axis([1,tot,0,1])
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 8.11 (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 :

```
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]}")
```

On peut alors tracer les graphes demandés (on a ajouté des annotations) :

```
from matplotlib.pyplot import *
figure(num=None,figsize=(30,10))

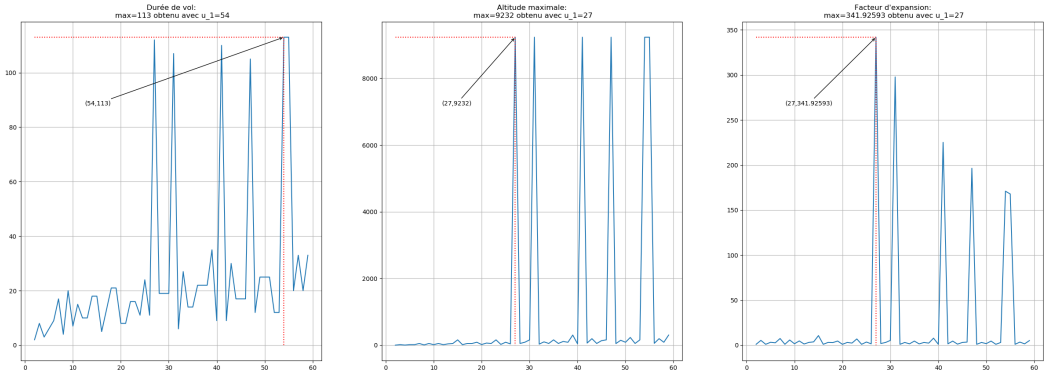
subplot(1,3,1)
plot(Uinit,L)
maxL=max(L)
indicemaxL=L.index(maxL)+2
plot([Uinit[0],indicemaxL,indicemaxL],[maxL,maxL,0],'r:')
title(f"Durée de vol:\n max={maxL} obtenu avec u_1={indicemaxL}")
annotate(f'({indicemaxL},{maxL})',
        xy=(indicemaxL, maxL), xycoords='data' ,
         xytext=(0.3, 0.75), textcoords='axes fraction',
        arrowprops=dict(arrowstyle="->"),
        horizontalalignment='right',
        verticalalignment='bottom'
        )
grid()

subplot(1,3,2)
plot(Uinit,M)
maxM=max(M)
indicemaxM=M.index(maxM)+2
plot([Uinit[0],indicemaxM,indicemaxM],[maxM,maxM,0],'r:')
title(f"Altitude maximale:\n max={maxM} obtenue avec u_1={indicemaxM}")
annotate(f'({indicemaxM},{maxM})',
        xy=(indicemaxM, maxM), xycoords='data' ,
         xytext=(0.3, 0.75), textcoords='axes fraction',
        arrowprops=dict(arrowstyle="->"),
        horizontalalignment='right',
```

```
        verticalalignment='bottom'
        )
grid()

subplot(1,3,3)
plot(Uinit,F)
maxF=max(F)
indicemaxF=F.index(maxF)+2
plot([Uinit[0],indicemaxF,indicemaxF],[maxF,maxF,0],'r:')
title(f"Facteur d'expansion:\n max={maxF:.5f} obtenu avec u_1={indicemaxF}")
annotate(f'({indicemaxF},{maxF:.5f})',
        xy=(indicemaxF, maxF), xycoords='data' ,
         xytext=(0.3, 0.75), textcoords='axes fraction',
        arrowprops=dict(arrowstyle="->"),
        horizontalalignment='right',
        verticalalignment='bottom'
        )
grid()

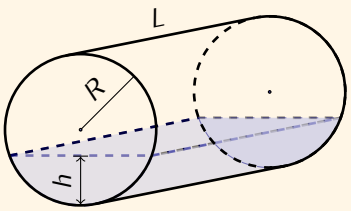
#savefig("Images/exo-7-6.png")
show()
```



☆ Exercice Bonus 8.12 (Cuve de fioul enterrée)

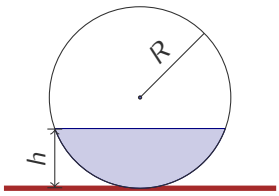
J'ai acheté une ancienne maison qui utilise du mazout pour le chauffage. Le constructeur avait enterré une cuve cylindrique dont je ne connais pas la capacité, je peux juste mesurer la hauteur du mazout dans la cuve et le rayon qui est de 0.80 m . Sachant qu'au départ la hauteur du mazout dans la cuve 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.

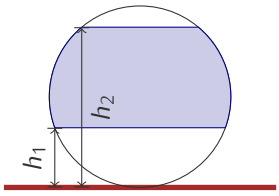


Correction

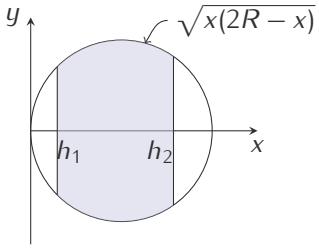
1. **Calcul de la capacité de la cuve.**
   Notons L la longueur de la cuve, R son rayon et h la hauteur du mazout. On a $0 \leq h \leq 2R$. Considérons une coupe verticale de la cuve :



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 $L \times \mathcal{A}$ où $\mathcal{A}$ est la surface coloriée :



Pour calculer l'aire coloriée, on "tourne" le cercle et on le plonge dans un repère orthonormé :



On peut alors calculer l'aire grâce au calcul d'une intégrale :

$$\mathcal{A} = 2\int_{h_1}^{h_2} \sqrt{x(2R-x)}\,\mathrm{d}x.$$

Comme $L \times \mathcal{A} = 3$ m^3, on trouve ensuite L (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 math import *
from scipy import integrate

R =0.8  # en metres
h1=0.36 # en metres
h2=1.35 # en metres

g = lambda x : sqrt(x*(2*R-x))

A_m2 = 2*integrate.quad(g,h1,h2)[0]  # en metres carres
L_m  = 3/A_m2                        # en metres
V_m3 = pi*R**2*L_m                   # en metres cubes
V_l  = V_m3*1.e3                     # en litres

print(f'A = {A_m2} m^2')
print(f'L = {L_m} m')
print(f'V = {V_m3} m^3 = {V_l} litres')
```

```
A = 1.4713582865283241 m^2
L = 2.038932343989792 m
V = 4.0995167187487676 m^3 = 4099.516718748768 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 :

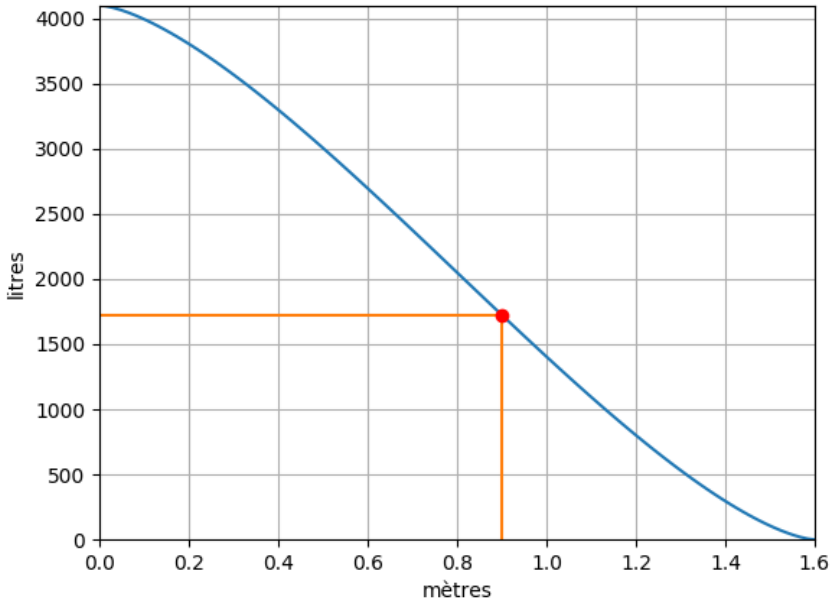
$$\begin{aligned} f : [0;2R] &\to [0;4000] \\ h &\mapsto 10^3 \times \left(V_{m^3} - 2 \times L_m \times \int_0^h \sqrt{x(2R-x)}\,\mathrm{d}x \right) \end{aligned}$$

```
from matplotlib.pyplot import *

f = lambda h : 1.e3*(V_m3-2*L_m*integrate.quad(g,0,h)[0])

hh=linspace(0,2*R,101)
yy=[f(h) for h in hh]
plot(hh,yy)

htest = 0.9
ftest = f(htest)
plot([htest,htest,0],[0,ftest,ftest])
plot([htest],[ftest],lw=0.5,marker='o',color='red')
xlabel("mètres")
ylabel("litres")
grid()
axis([0,2*R,0,V_l])
show()
```



3. **Calcul analytique** Calculons tous d'abord les primitives de $\sqrt{x(2R-x)}$:

$$\begin{aligned}\int \sqrt{x(2R-x)}\,\mathrm{d}x &\overset{\substack{x=R-y\\ \mathrm{d}x=-\mathrm{d}y}}{=} -\int \sqrt{R^2-y^2}\,\mathrm{d}y \overset{\substack{y=R\sin(t)\\ \mathrm{d}y=R\cos(t)\,\mathrm{d}t}}{=} -R^2\int \cos^2(t)\,\mathrm{d}t\\ &= -\frac{R^2}{2}\left(\sin(t)\cos(t)+t\right)+c = -\frac{R^2}{2}\left(\sin(t)\sqrt{1-\sin^2(t)}+t\right)+c\\ &\overset{t=\arcsin\left(\frac{y}{R}\right)}{=} -\frac{R^2}{2}\left(\frac{y}{R}\sqrt{1-\frac{y^2}{R^2}}+\arcsin\left(\frac{y}{R}\right)\right)+c\\ &\overset{y=R-x}{=} -\frac{R-x}{2}\sqrt{x(2R-x)}-\frac{R^2}{2}\arcsin\left(1-\frac{x}{R}\right)+c.\end{aligned}$$

Par conséquent

$$\mathcal{A} = 2\int_{h_1}^{h_2}\sqrt{x(2R-x)}\,\mathrm{d}x \approx 1.48 \quad \mathrm{m}^2 \quad .$$

Comme $L\times\mathcal{A} = 3 \quad \mathrm{m}^3$, on trouve ensuite $L \approx 2 \quad \mathrm{m}$ et le volume totale de la cuve est $\pi R^2 L \approx 4 \quad \mathrm{m}^3$,

i.e. environs 4000 L . Enfin

$$\begin{aligned} f(h) &= 10^3 \times \left(V_{m^3} - 2 \times L_m \times \int_0^h \sqrt{x(2R-x)}\,\mathrm{d}x \right) \\ &= 10^3 \times \left(V_{m^3} - 2 \times L_m \times \left(-\frac{R-h}{2}\sqrt{h(2R-h)} - \frac{R^2}{2}\arcsin\left(1-\frac{h}{R}\right) + \frac{R^2}{2}\arcsin(1) \right) \right) \\ &= 10^3 \times \left(V_{m^3} + L_m \left((R-h)\sqrt{h(2R-h)} + R^2 \arcsin\left(1-\frac{h}{R}\right) - \frac{\pi R^2}{2} \right) \right) \end{aligned}$$

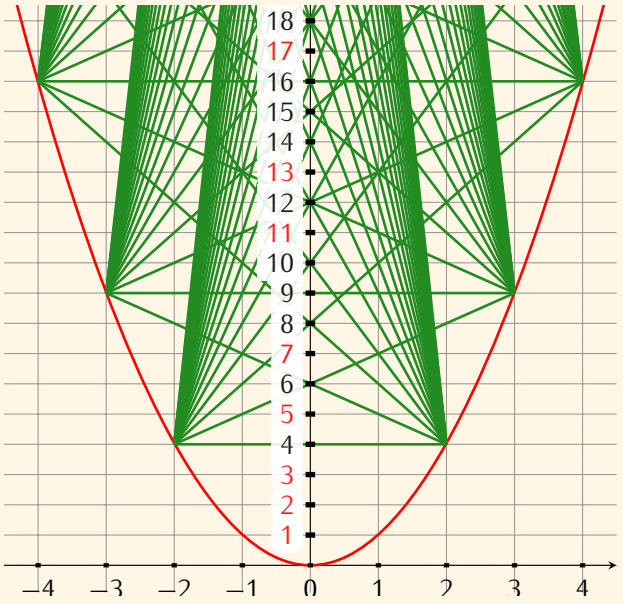
- ▷ $f(0) = V_\ell$, $f(2R) = 0$;
- ▷ $f'(h) = -2 \times 10^3 \times L_m \sqrt{h(2R-h)}$;
- ▷ $f'(h) = 0$ ssi $h = 0$ ou $h = 2R$;
- ▷ f est décroissante pour tout $h \in [0; 2R]$;
- ▷ $f''(h) = -2 \times 10^3 \times L_m \frac{h-R}{\sqrt{h(2R-h)}}$;
- ▷ $h = R$ est un point d'inflexion ;
- ▷ f convexe pour $h \in [R; 2R]$, concave pour $h \in [0; R]$.

☆ Exercice Bonus 8.13 (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. ```
   from matplotlib.pylab import *
   # la parabole
   xx=linspace(-5,5,101)
   yy=[x**2 for x in xx]
   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
axis([-5,5,0,19])
xticks(range(-5,5,1), size='small')
yticks(range(0,19,1), size='small')
grid()
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 Bonus 8.14 (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 2018 est partagé en tranches et l'impôt à payer en 2019 est calculé en prenant un certain pourcentage de chaque tranche. Pour les revenus de 2018 (impôts 2019), 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 :

Tranche Du Revenu 2018	Taux d'imposition 2019
• jusqu'à 9964 € :	0%
• de 9964 € à 27519 € :	14%
• de 27519 € à 73779 € :	30%
• de 73779 € à 156244 € :	41%
• plus de 156244 € :	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 9964 €, on ne paye pas d'impôt.
- ▷ Si le revenu est compris entre 9964 € et 27519 €, on ne paye pas d'impôt sur les premiers 9964 € de son revenu, et on paye 14% de la partie qui excède 9964 €. Par exemple, si le revenu est 9964 €, on payera 14% de 1 €, c'est-à-dire 14 centimes.
- ▷ Si le revenu est compris entre 27519 € et 73779 €, on ne paye rien sur la première tranche de 9964 €, puis 14% sur la deuxième tranche, allant de 9964 € à 27519 € (soit 14% de $(27519 - 9964) = 2457.70$ €), et enfin 30% sur la partie du revenu qui excède 27519 €. Par exemple : si le revenu est de 72 000 €, l'impôt sera de $0 + 2457.70 + 30\% \times (72000 - 27519) = 15\,802$ €.

Prenons l'exemple concret d'un contribuable, célibataire, qui en 2019 déclare 100000 € de revenu pour l'année 2018. 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

$$0 \times (9964 - 0) + 14\% \times (27519 - 9964) + 30\% \times (73779 - 27519) + 41\% \times (100000 - 73779) = 27086.31\,€$$

et est ainsi redevable de 27.09% de ses revenus (et non 41%).

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.

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).

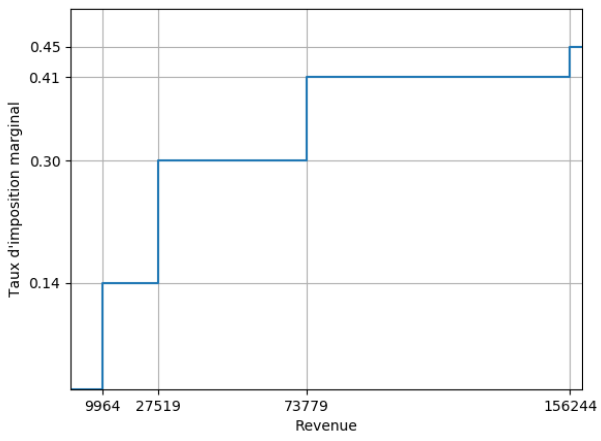
Correction

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, qu'on appelle le taux marginal.

```
from matplotlib.pylab import *

tranche=[9964,27519,73779,156244]
taux=[0.14,0.30,0.41,0.45]

plot([0,tranche[0],tranche[0],tranche[1],tranche[1],tranche[2],tranche[2],tranche[3],tranche[3],
 ↪  160000],
     [0,          0,    taux[0],    taux[0],    taux[1],    taux[1],    taux[2],    taux[2],
      ↪  taux[3],taux[3]],
     lw=1.5)
axis([0,160000,0,0.5])
xticks(tranche)
yticks(taux)
xlabel('Revenue')
ylabel("Taux d'imposition marginal")
grid()
savefig("Images/margin.png")
show()
```



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 $I\colon \mathbb{R}_+ \to \mathbb{R}_+$ est une fonction continue mais pas sa dérivée $I'\colon \mathbb{R}_+ \to \mathbb{R}_+$ qui est définie par

$$I'(R) = \begin{cases} 0 & \text{si } R < 9964\,€ \\ 0.14 & \text{si } 9964\,€ < R < 27519\,€ \\ 0.30 & \text{si } 27519\,€ < R < 73779\,€ \\ 0.41 & \text{si } 73779\,€ < R < 156244\,€ \\ 0.45 & \text{si } R > 156244\,€ \end{cases}$$

Ce graphique indique le pourcentage d'imposition qui serait appliqué *sur chaque euro supplémentaire* qui viendrait s'ajouter au revenu : si votre revenu est par exemple de 80000 €, alors pour chaque euro supplémentaire que vous gagnez, 41 centimes seront pour les impôts.

2. Il s'agit de la fonction $I\colon \mathbb{R}_+ \to \mathbb{R}_+$ définie par

$$\begin{aligned} I(R) &= \int_0^R I'(r)\,\mathrm{d}r \\ &= \begin{cases} \int_0^R 0\,\mathrm{d}r & \text{si } R < 9964\,€ \\ \int_0^{9964} 0\,\mathrm{d}r + \int_{9964}^R 0.14\,\mathrm{d}r & \text{si } 9964\,€ < R < 27519\,€ \\ \int_0^{9964} 0\,\mathrm{d}r + \int_{9964}^{27519} 0.14\,\mathrm{d}r + \int_{27519}^R 0.30\,\mathrm{d}r & \text{si } 27519\,€ < R < 73779\,€ \\ \int_0^{9964} 0\,\mathrm{d}r + \int_{9964}^{27519} 0.14\,\mathrm{d}r + \int_{27519}^{73779} 0.30\,\mathrm{d}r + \int_{73779}^R 0.41\,\mathrm{d}r & \text{si } 73779\,€ < R < 156244\,€ \\ \int_0^{9964} 0\,\mathrm{d}r + \int_{9964}^{27519} 0.14\,\mathrm{d}r + \int_{27519}^{73779} 0.30\,\mathrm{d}r + \int_{73779}^{156244} 0.41\,\mathrm{d}r + \int_{156244}^R 0.45\,\mathrm{d}r & \text{si } R > 156244\,€ \end{cases} \\ &= \begin{cases} 0 & \text{si } R < 9964\,€ \\ 0 + 0.14(R - 9964) & \text{si } 9964\,€ < R < 27 \\ 0 + 0.14(27519 - 9964) + 0.30(R - 27519) & \text{si } 27519\,€ < R < 7 \\ 0 + 0.14(27519 - 9964) + 0.30(73779 - 27519) + 0.41(R - 73779) & \text{si } 73779\,€ < R < 1 \\ 0 + 0.14(27519 - 9964) + 0.30(73779 - 27519) + 0.41(156244 - 73779) + 0.45(R - 156244) & \text{si } R > 156244\,€ \end{cases} \\ &= \begin{cases} 0 & \text{si } R \leq 9964\,€ \\ 0.14R - 1394.96 & \text{si } 9964\,€ < R \leq 27519\,€ \\ 0.30R - 5798 & \text{si } 27519\,€ < R \leq 73779\,€ \\ 0.41R - 13913.69 & \text{si } 73779\,€ < R \leq 156244\,€ \\ 0.45R - 20163.45 & \text{si } R > 156244\,€ \end{cases} \end{aligned}$$

Voici la représentation graphique de la fonction Impôt en fonction du revenu avec le revenu imposable en horizontale et l'impôt à payer en verticale.

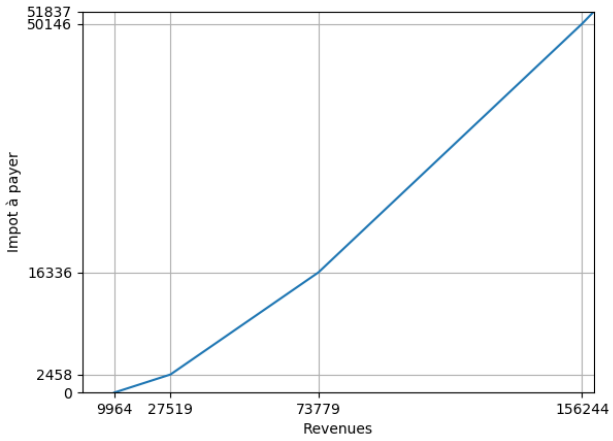
```
from matplotlib.pylab import *

tranche=[9964,27519,73779,156244]
taux=[0.14,0.30,0.41,0.45]

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-tranc
        else:
                return
                 ↪ 0+taux[0]*(tranche[1]-tranche[0])+taux[1]*(tranche[2]-tranche[1])+taux[2]*(tranche

rr=tranche+[160000]
```

```
yy=[Impot(r) for r in rr]
plot(rr,yy,lw=1.5)
axis([0,160000,0,yy[-1]])
xticks(tranche)
yticks(yy)
xlabel('Revenues')
ylabel("Impot à payer")
grid()
savefig("Images/apayer.png")
show()
```



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 2019 déclare 100000 € de revenu pour l'année 2018 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. Dans notre exemple, l'impôt payé est de $0 \times (9964 - 0) + 14\% \times (27519 - 9964) + 30\% \times (73779 - 27519) + 41\% \times (100000 - 73779) = 27086.31$ € pour un revenu de 100000 €, soit un taux réel d'imposition de $27086.31/100000 \simeq 27.08\%$ (et non 41%). Depuis quelques années le taux effectif d'imposition est indiqué sur l'avis d'imposition.

   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 9202 €, si bien que le taux global est de $9202/50000 = 0.18404$, c'est-à-dire $\simeq 18.40\%$.

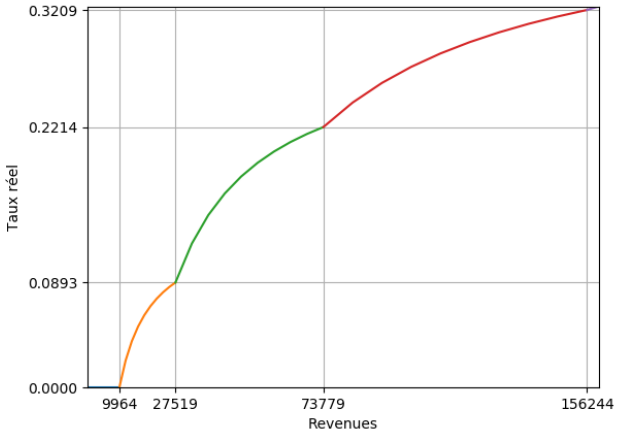
```
from matplotlib.pylab import *

tranche=[9964,27519,73779,156244]
taux=[0.14,0.30,0.41,0.45]

def Taux_Reel(R):
        if R<=tranche[0]:
                return 0
        elif R<=tranche[1]:
                return 0+taux[0]*(R-tranche[0])/R
        elif R<=tranche[2]:
                return 0+taux[0]*(tranche[1]-tranche[0])/R+taux[1]*(R-tranche[1])/R
        elif R<=tranche[3]:
                return
                 ↪  0+taux[0]*(tranche[1]-tranche[0])/R+taux[1]*(tranche[2]-tranche[1])/R+taux[2]
        else:
                return
                 ↪  0+taux[0]*(tranche[1]-tranche[0])/R+taux[1]*(tranche[2]-tranche[1])/R+taux[2]

rr=linspace(0,tranche[0],10)
```

```
yy=[Taux_Reel(r) for r in rr]
plot(rr,yy,lw=1.5)
rr=linspace(tranche[0],tranche[1],10)
yy=[Taux_Reel(r) for r in rr]
plot(rr,yy,lw=1.5)
rr=linspace(tranche[1],tranche[2],10)
yy=[Taux_Reel(r) for r in rr]
plot(rr,yy,lw=1.5)
rr=linspace(tranche[2],tranche[3],10)
yy=[Taux_Reel(r) for r in rr]
plot(rr,yy,lw=1.5)
rr=linspace(tranche[3],160000,10)
yy=[Taux_Reel(r) for r in rr]
plot(rr,yy,lw=1.5)
axis([0,160000,0,yy[-1]])
xticks(tranche)
yticks([Taux_Reel(r) for r in tranche])
xlabel('Revenues')
ylabel("Taux réel")
grid()
savefig("Images/reel.png")
show()
```



Il s'agit de la fonction $G\colon \mathbb{R}_+ \to \mathbb{R}_+$ définie par

$$G(R) = \frac{I(R)}{R} = \begin{cases} 0 & \text{si } R \leq 9964\,€ \\ 0.14 - \frac{1394.96}{R} & \text{si } 9964\,€ < R \leq 27519\,€ \\ 0.30 - \frac{5798}{R} & \text{si } 27519\,€ < R \leq 73779\,€ \\ 0.41 - \frac{13913.69}{R} & \text{si } 73779\,€ < R \leq 156244\,€ \\ 0.45 - \frac{20163.45}{R} & \text{si } R > 156244\,€ \end{cases}$$

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.

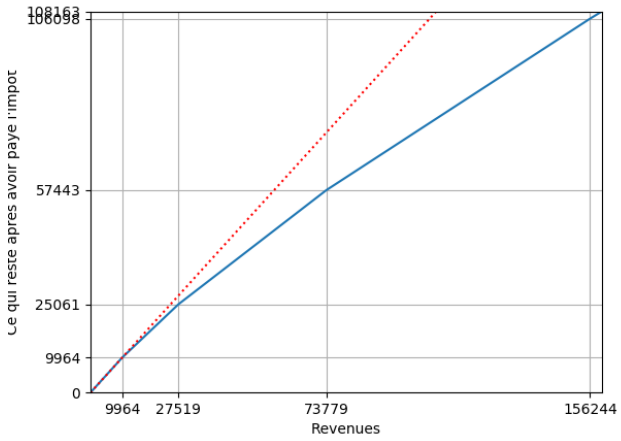
```
from matplotlib.pylab import *

tranche=[9964,27519,73779,156244]
taux=[0.14,0.30,0.41,0.45]

def Reste(R):
```

```
        if R<=tranche[0]:
                return R
        elif R<=tranche[1]:
                return R-taux[0]*(R-tranche[0])
        elif R<=tranche[2]:
                return R-taux[0]*(tranche[1]-tranche[0])-taux[1]*(R-tranche[1])
        elif R<=tranche[3]:
                return
                 ↪ R-taux[0]*(tranche[1]-tranche[0])-taux[1]*(tranche[2]-tranche[1])-taux[2]*(R-
        else:
                return
                 ↪ R-taux[0]*(tranche[1]-tranche[0])-taux[1]*(tranche[2]-tranche[1])-taux[2]*(tra

figure(4)
rr=[0]+tranche+[160000]
yy=[Reste(r) for r in rr]
plot(rr,yy,lw=1.5)
plot([0,160000],[0,160000],'r:')
axis([0,160000,0,yy[-1]])
xticks(tranche)
yticks(yy)
xlabel('Revenues')
ylabel("Ce qui reste après avoir payé l'impôt")
grid()
savefig("Images/reste.png")
show()
```



Il s'agit de la fonction $A\colon \mathbb{R}_+ \to \mathbb{R}_+$ définie par

$$A(R) = R - I(R) = \begin{cases} 0 & \text{si } R \leq 9964\,€ \\ (1-0.14)R - 1359.4 & \text{si } 9964\,€ < R \leq 27519\,€ \\ (1-0.30)R - 5650.28 & \text{si } 27519\,€ < R \leq 73779\,€ \\ (1-0.41)R - 13559.06 & \text{si } 73779\,€ < R \leq 156244\,€ \\ (1-0.45)R - 19649.46 & \text{si } R > 156244\,€ \end{cases}$$

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 (/, +, ...) 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. Il ne faut jamais se fier trop vite au résultat d'un calcul obtenu avec un ordinateur...

L'expérimentation numérique dans les sciences est un sujet passionnant et un outil fort utile, devenu indispensable pour certains scientifiques. Malgré la puissance vertigineuse de calcul de nos ordinateurs aujourd'hui, et encore

plus de certains centres de calculs, on aurait tort d'oublier complètement la théorie et de trop se moquer de comment fonctionne la machine, au risque d'avoir quelques surprises...

Observons des calculs quelque peu surprenants :

```
>>> 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.

Erreurs d'arrondis

```
>>> 1 / 3 - 1 / 4 - 1 / 12
-1.3877787807814457e-17
```

Non-commutativité

```
>>> 1 + 1e-16 - 1
0.0
>>> -1 + 1e-16 + 1
1.1102230246251565e-16
```

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
```

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
print( "-"*9,"a" ,"-"*9, "|" , "-"*9, "b", "-"*9, "|" , "ab-((a+b)/2)^2-((a-b)/2)^2" )
```

```
for i in range(6):
⟶produit = prod(a,b)
⟶difference = diff(a,b)
⟶print( "%1.15e | %1.15e |%1.15e" % (a,b,produit-difference) )
⟶a, b = produit, a+1
```

On constate que la divergence est spectaculaire :

```
--------- a --------- | --------- b --------- | ab-((a+b)/2)^2-((a-b)/2)^2
6.553990000000000e+03 | 6.554990000000000e+03 |0.000000000000000e+00
4.296133891010000e+07 | 6.554990000000000e+03 |-5.859375000000000e-02
2.816111469423164e+11 | 4.296133991010000e+07 |1.667072000000000e+06
1.209839220626197e+19 | 2.816111469433164e+11 |-4.798256640190465e+21
3.407042105375514e+30 | 1.209839220626197e+19 |1.046648870315659e+44
4.121973165408151e+49 | 3.407042105375514e+30 |1.404373613177356e+80
```

Le module `fractions` Pour corriger ce problème on peut utiliser le module `fractions` :

```
from fractions import Fraction
print(0.1 + 0.1 + 0.1 - 0.3, "\t vs ", Fraction(1,10) + Fraction(1,10) + Fraction(1,10) -
↪  Fraction(3,10))
print(1.1 + 2.2, "\t vs ", Fraction(11,10) + Fraction(22,10))
print(1.0 / 3 - 1.0 / 4 - 1.0 / 12, " vs ", Fraction(1,3) + Fraction(1,4) - Fraction(1,12))
print( 1 + 1e-16 - 1, " vs ", Fraction(1,1) + Fraction(1,10**16) - Fraction(1,1))
print(-1 + 1e-16 + 1, " vs ", -Fraction(1,1) + Fraction(1,10**16) + Fraction(1,1))
print(1.2 - 1.0 - 0.2, " vs ", Fraction(6,5) - Fraction(1,1) - Fraction(1,5))
```

```
5.551115123125783e-17 ⟶ vs  0
3.3000000000000003 ⟶ vs  33/10
-1.3877787807814457e-17  vs  1/2
0.0  vs  1/10000000000000000
1.1102230246251565e-16  vs  1/10000000000000000
-5.551115123125783e-17  vs  0
```

Revenons sur notre exemple :

```
from fractions import Fraction
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
print( "-"*9,"a" ,"-"*9, "|" , "-"*9, "b", "-"*9, "|" , "ab-((a+b)/2)^2-((a-b)/2)^2" )
for i in range(6):
⟶produit = prod(a,b)
⟶difference = diff(a,b)
⟶print( "%1.15e | %1.15e |%1.15e" % (a,b,produit-difference) )
⟶a, b = produit, a+1
```

```
--------- a --------- | --------- b --------- | ab-((a+b)/2)^2-((a-b)/2)^2
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, à Dharan (Arabie Saoudite), a échoué dans l'interception d'un missile Scud irakien. Le Scud a frappé un baraquement de l'armée américaine et a tué 28 soldats. La commission d'enquête a conclu à un calcul incorrect du temps de parcours, dû à un problème d'arrondi. Les nombres étaient représentés en virgule fixe sur 24 bits. Le temps était compté par l'horloge interne du système en 1/10 de seconde. Malheureusement, 1/10 n'a pas d'écriture

finie dans le système binaire : 1/10 = 0,1 (dans le système décimal) = 0,0001100110011001100110011… (dans le système binaire). L'ordinateur de bord arrondissait 1/10 à 24 chiffres, d'où une petite erreur dans le décompte du temps pour chaque 1/10 de seconde. Au moment de l'attaque, la batterie de missile Patriot était allumée depuis environ 100 heures, ce qui avait entraîné une accumulation des erreurs d'arrondi de 0,34 s. Pendant ce temps, un missile Scud parcourt environ 500 m, ce qui explique que le Patriot soit passé à côté de sa cible.

▷ Le 4 juin 1996, une fusée Ariane 5 a explosé 40 secondes après l'allumage. La fusée et son chargement avaient coûté 500 millions de dollars. La commission d'enquête a rendu son rapport au bout de deux semaines. Il s'agissait d'une erreur de programmation dans le système inertiel de référence. À un moment donné, un nombre codé en virgule flottante sur 64 bits (qui représentait la vitesse horizontale de la fusée par rapport à la plate-forme de tir) était converti en un entier sur 16 bits. Malheureusement, le nombre en question était plus grand que 32768, le plus grand entier que l'on peut coder sur 16 bits, et la conversion a été incorrecte.

A.2. Exercices

Exercice A.1 (Devine le résultat)

Quel résultat donne le code suivant ?

```
print(0.2+0.3+0.4)
print(0.3+0.4+0.2)
print(0.4+0.2+0.3)
print(0.2+0.4)
```

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.

```
>>> 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
```

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
```

Exercice A.2 (Qui est plus grand ?)

Parmi A et B, qui est plus grand si $A = \frac{2^{2019}-1}{4^{2019}-2^{2019}+1}$ et $B = \frac{2^{2019}+1}{4^{2019}+2^{2019}+1}$?

Correction

Naïvement on pourrait essayer ceci :

```
>>> A=(2**2019-1)/(4**2019-2**2019+1)
>>> B=(2**2019+1)/(4**2019+2**2019+1)
>>> print(A-B)
0.0
```

et en déduire que $A = B$, mais regardons un peu mieux :

```
>>> A=(2**2019-1)/(4**2019-2**2019+1)
>>> B=(2**2019+1)/(4**2019+2**2019+1)
>>> print(A)
0.0
>>> print(B)
0.0
```

Or, ceci ce n'est pas possible car $2^{2019} \pm 1 \neq 0$.

Posons $x = 2^{2019} > 0$ alors $A = \dfrac{x-1}{x^2-x+1}$ et $B = \dfrac{x+1}{x^2+x+1}$ ainsi

$$\begin{aligned} A - B &= \frac{(x-1)(x^2+x+1) - (x+1)(x^2-x+1)}{(x^2-x+1)(x^2+x+1)} \\ &= \frac{(x^3+x^2+x-x^2-x-1) - (x^3-x^2+x+x^2-x+1)}{(x^2-x+1)(x^2+x+1)} \\ &= \frac{-2}{(x^2-x+1)(x^2+x+1)} < 0 \end{aligned}$$

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**2019-1,4**2019-2**2019+1)
B=Fraction(2**2019+1,4**2019+2**2019+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 (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

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 :

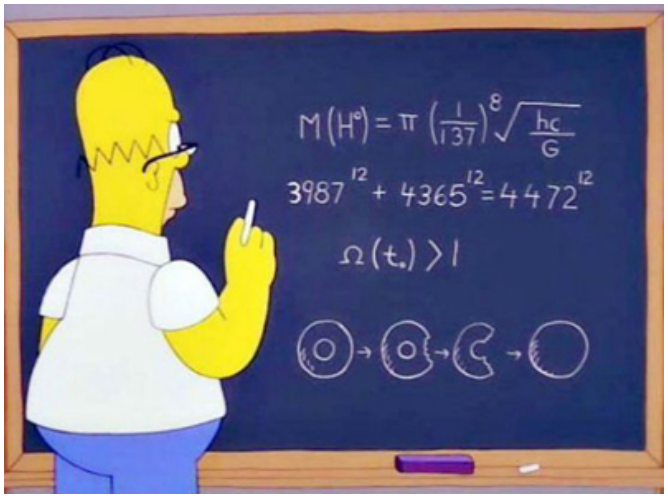
```
>>> 844487**5 + 1288439**5 - 1318202**5
-235305158626
```

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 = 398024523518563901305561949 7406 Arrondis = 3.980245235185639e+30
>>> print("Exact =", 1318202**5, "Arrondis =", 1318202.**5)
Exact = 398024523518563901329092465 6032 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,

$$3987^{12}+4365^{12}=6397665634969861261623623095315448789698710 6$$
$$4472^{12}=6397665634848672580686235832216857578412441 6$$

soit une différence de

$$121188680937387263098591211286269 0.$$

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.4 (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("u_{}={}".format(n,u))
for n in range(1,30):
⟶u = 5*u-1
⟶print("u_{}={}".format(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("v_{}={}".format(n,v))
for n in range(1,30):
⟶v = 6*v-1
⟶print("v_{}={}".format(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_10=0.20000000179015842    v_20=0.3082440341822803
v_1=0.20000000000000018    v_11=0.20000001074095053    v_21=0.8494642050936818
v_2=0.20000000000000107    v_12=0.20000006444570317    v_22=4.096785230562091
v_3=0.2000000000000064     v_13=0.20000038667421904    v_23=23.580711383372545
v_4=0.20000000000003837    v_14=0.20000232004531426    v_24=140.48426830023527
v_5=0.20000000000023022    v_15=0.20001392027188558    v_25=841.9056098014116
v_6=0.2000000000013813     v_16=0.2000835216313135     v_26=5050.43365880847
v_7=0.20000000000828777    v_17=0.20050112978788093    v_27=30301.60195285082
v_8=0.20000000004972662    v_18=0.20300677872728556    v_28=181808.6117171049
v_9=0.20000000029835974    v_19=0.21804067236371338    v_29=1090850.6703026295
```

De même

```
n=0
w = 1/3
print("w_{}={}".format(n,w))
for n in range(1,41):
⟶w = 4*w-1
⟶print("w_{}={}".format(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("v_{}={}".format(n,v))
for n in range(1,30):
⟶v = 6*v-1
⟶print("v_{}={}".format(n,v))

n=0
w = Fraction(1,3)
print("v_{}={}".format(n,v))
for n in range(1,41):
⟶w = 4*w-1
⟶print("v_{}={}".format(n,w))
```

```
v_0=1/5
v_1=1/5
v_2=1/5
v_3=1/5
v_4=1/5
v_5=1/5
v_6=1/5
v_7=1/5
v_8=1/5
v_9=1/5
v_10=1/5
v_11=1/5
v_12=1/5
v_13=1/5
v_14=1/5
v_15=1/5
v_16=1/5
v_17=1/5
v_18=1/5
v_19=1/5
v_20=1/5
v_21=1/5
v_22=1/5
v_23=1/5
v_24=1/5
v_25=1/5
v_26=1/5
v_27=1/5
v_28=1/5
v_29=1/5
v_0=1/5
v_1=1/3
v_2=1/3
v_3=1/3
v_4=1/3
v_5=1/3
v_6=1/3
v_7=1/3
v_8=1/3
v_9=1/3
v_10=1/3
v_11=1/3
v_12=1/3
v_13=1/3
v_14=1/3
v_15=1/3
v_16=1/3
v_17=1/3
v_18=1/3
v_19=1/3
v_20=1/3
v_21=1/3
v_22=1/3
v_23=1/3
v_24=1/3
v_25=1/3
v_26=1/3
v_27=1/3
v_28=1/3
v_29=1/3
v_30=1/3
v_31=1/3
v_32=1/3
v_33=1/3
v_34=1/3
v_35=1/3
v_36=1/3
v_37=1/3
v_38=1/3
v_39=1/3
v_40=1/3
```

Exercice A.5 (Suite de Muller)

Considérons la suite

$$\begin{cases} x_0 = 4, \\ x_1 = 4.25, \\ x_{n+1} = 108 - \frac{815 - \frac{1500}{x_{n-1}}}{x_n}. \end{cases}$$

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("x_{}={}".format(0,x[0]))
print("x_{}={}".format(1,x[1]))
for i in range(2,30):
⟶x.append(108-(815-(1500/x[-2]))/x[-1])
⟶print("x_{}={}".format(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("x_{}={}".format(0,x[0]))
print("x_{}={}".format(1,x[1]))
for i in range(2,30):
⟶x.append(108 - Fraction((815 - Fraction(1500, x[-2])), x[-1]))
⟶print("x_{}={} ≈ {}".format(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.6 (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("x_{}={}".format(0,x[0]))
print("x_{}={}".format(1,x[1]))
for i in range(2,2001):
⟶x.append(2003-6002/x[-1]+4000/(x[-1]*x[-2]))
print("x_{}={}".format(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 :

```
from fractions import Fraction
x=[Fraction(3,2),Fraction(5,3)]
print("x_{}={}".format(0,x[0]))
print("x_{}={}".format(1,x[1]))
for i in range(2,2001):
⟶x.append(2003-Fraction(6002,x[-1])+Fraction(4000,x[-2]*x[-1]))
print("x_{}= ≈ {}".format(i,float(x[i])))
```

```
x_0=3/2                    x_1=5/3                    x_2000= ≈ 2.0
```

Exercice A.7 (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.8 (Calcul d'intégrale par récurrence)

On veut approcher numériquement l'intégrale $I_n = \int_0^1 x^n e^{\alpha x} \mathrm{d}x$ 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

$$I_n = \int_0^1 x^n e^{\alpha x} \mathrm{d}x \tag{A.2}$$

$$= \frac{1}{\alpha} e^{\alpha} - \frac{n}{\alpha} I_{n-1} \tag{A.3}$$

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

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("I_{}={}".format(0,I[-1]))
for n in range(20):
⟶I.append(exp(alpha)/alpha -(n+1)*I[-1]/alpha)
⟶print("I_{}={}".format(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=0.2490280313165915
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=0.14236307722318344
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 $\frac{n!}{\alpha^n}|\varepsilon_0|$.
Ici on ne peut pas utiliser le module `fraction` car $e \notin \mathbb{Q}$.

Annexe B.

Manipulation de fichiers

B.1. Ouverture et fermeture d'un fichier

```
f = open(nomfichier[,mode][,encoding='xxx'])}
```

- le premier argument est une chaîne contenant le nom du fichier à ouvrir ;
- le second argument est une chaîne précisant le mode d'ouverture :
  - `"w"` (write) indique un mode en écriture,
  - `"r"` (read) indique un mode en lecture (mode par défaut si argument omis),
  - `"a"` (append) indique un mode en ajout ;
- le troisième argument est une chaîne indiquant l'encodage des caractères du fichier (par défaut spécifique au système d'exploitation).

Notons qu'en mode `"w"`, le fichier est créé et que s'il préexistait, l'ancien est effacé et remplacé par le nouveau (d'où l'utilité du mode `"a"`).

Lorsque les opérations sur un fichier sont terminées, il faut le fermer par un appel à sa méthode `close()` qui permet de libérer les ressources allouées pour la gestion de l'accès au fichier et de s'assurer que les données stockées par le système d'exploitation dans des zones mémoire tampons sont effectivement écrites sur le disque.

```
f.close()
```

B.2. Écriture séquentielle dans un fichier

Si on lance le script suivant

```
f = open("monFichier1.txt","w",encoding='utf-8')
f.write("Hé bien ça démarre sec.")
f.write("Là, on a 0123456789 valeurs !")
f.close()
```

et on ouvre le fichier `monFichier1.txt` dans le répertoire courant, on obtient les deux chaînes écrites et concaténées, car elles ont été écrites l'une après l'autre sans séparateur. Pour qu'il aille à la ligne on écrira

```
f = open("monFichier.txt","w",encoding='utf-8')
f.write("Hé bien ça démarre sec.\n")
f.write("Là, on a 0123456789 valeurs !")
f.close()
```

- Ne pas confondre le nom du fichier (chaîne) sur disque avec le nom de variable de l'objet-fichier qui y donne accès (objet retourné par `open()`).
- Le fichier est sur le disque dur alors que l'objet-fichier est dans la mémoire vive de l'ordinateur (celle dont le contenu est perdu lorsqu'on éteint la machine).
- La méthode `write()` réalise l'écriture proprement dite. Son argument est une chaîne de caractères, le plus souvent formatée. Elle retourne le nombre de caractères écrits. Elle n'ajoute pas de retour à la ligne (caractère `'\n'`), il faut le spécifier dans la chaîne à écrire.
- La méthode `close()` referme le fichier. En raison des caches d'optimisation du système d'exploitation, il n'est pas garanti que les modifications faites soient écrites physiquement sur le disque tant que le fichier n'est pas fermé.

B.3. Lecture séquentielle dans un fichier

On peut maintenant relire les données que l'on vient de stocker.

```
f= open("monFichier.txt",'r',encoding='utf-8')
t = f.read()
print(t)
f.close()
```

```
Hé bien ça démarre sec.
Là, on a 0123456789 valeurs !
```

- ▷ La méthode `read()` lit l'ensemble des données présentes dans le fichier et les affecte à une variable de type chaîne de caractères.
- ▷ La méthode `read()` peut être utilisée avec un argument entier qui indique alors le nombre de caractères à lire à partir de la position courante dans le fichier. Ceci permet de lire le fichier par morceaux.
- ▷ La méthode `close()` referme le fichier. Ceci permet de s'assurer qu'on libère les ressources allouées par le système d'exploitation pour l'accès au fichier.

B.4. Réécriture dans un fichier

On va ajouter une ligne dans le fichier existant `monFichier.txt` :

```
f = open("monFichier.txt","a",encoding='utf-8')
f.write("Une ligne de texte\n")
f.close()
```

Si on exécute plusieurs fois les instructions suivantes, que se passe-t-il ?

```
import time
f = open("monFichier.txt","a",encoding="utf-8")
f.write("On est à "+str(time.time())+"\n")
f.close()
```

B.5. Boucle de lecture dans un fichier

Les objets-fichiers sont des objets itérables (que l'on peut utiliser comme séquence dans une boucle for par exemple). À chaque itération ils fournissent la ligne de texte suivante dans le fichier. On peut ainsi écrire :

```
# Afficher les lignes d'un fichier qui contiennent une chaîne particulière:
f = open("monFichier.txt")
for line in f :
⟶if 'texte' in line :
⟶⟶print(line)
f.close()
```

B.6. Écriture de données

Jusqu'ici, nous avons simplement écrit des chaînes de caractères. Le script suivant crée un fichier qui sauvegarde d'autres données : un nombre flottant et une liste de chaînes. Cependant, il faut d'abord les convertir en chaînes de caractères.

```
s = "Une phrase"
x = 23.432343
lst = [ "Uno","Dos","Tres" ]
f = open("data.txt",'w',encoding='utf-8')
f.write(s)
f.write("\n")
f.write(str(x))
f.write("\n")
f.write(str(lst))
f.write("\n")
f.close()
```

Annexe C.

Autres ressources

Pour compléter votre apprentissage de Python, n'hésitez pas à consulter d'autres ressources complémentaires à ce polycopié. D'autres auteurs abordent l'apprentissage de Python d'une autre manière.

Il existe de nombreux sites Web sur Internet qui proposent des cours en ligne, des livres électroniques et des tutoriels (gratuits et payants). Vous trouverez ci-dessous une (très petite) liste de ressources qui pourraient vous être utiles.

- Gerard Swinnen. *Apprendre à programmer avec Python 3*. 2012. URL : http://inforef.be/swi/download/apprendre_python3_5.pdf
  Le code source des exemples de ce livre peut être téléchargé à partir du site de l'auteur : http://inforef.be/swi/python.htm
- Arnaud Bodin. *Python au lycée (tome 1) : Algorithmes et programmation (Livres Exo7) (French Edition)*. Avr. 2018. ISBN : 1091267510. URL : http://exo7.emath.fr/cours/livre-python1.pdf
- Patrick Fuchs et Pierre Poulain :
  "Cours de Python", https://python.sdv.univ-paris-diderot.fr/cours-python.pdf
- Laurent Pointal :
  "Une introduction à Python 3", notes de cours https://perso.limsi.fr/pointal/python:courspython3
  "Python 3 Exercices corrigés", https://perso.limsi.fr/pointal/_media/python:cours:exercices-python3.pdf
- Débuter avec Python au lycée : http://python.lycee.free.fr/index.html
- Alexandre Casamayou-Boucau, Pascal Chauvin et Guillaume Connan. *Programmation en Python pour les mathématiques-2e éd.* Dunod, 2016

- Recueil d'exercices pour apprendre Python au lycée
  https://www.codingame.com/playgrounds/17176/
- Apprendre le langage de programmation python
  https://python.doctor/
- Apprendre Python et s'initier à la programmation :
  https://python.developpez.com/tutoriels/apprendre-programmation-python/les-bases/
- Adaptation en français du Python Tutorial : https://docs.python.org/fr/3/tutorial/
- "How to Think Like a Computer Scientist : Interactive Edition"
  https://runestone.academy/runestone/books/published/thinkcspy/index.html
- http://www.france-ioi.org Cours et problèmes

- Pydéfis
  PyDéfis vous propose de petits défis de programmation, sous la forme d'énoncés (plus ou moins) courts. Vous pouvez résoudre ces défis par le moyen de votre choix, même si l'objectif est ici de réaliser un programme informatique. Vous pouvez consulter les énoncés tout de suite (il n'est pas nécessaire de s'identifier), mais il vous faudra ouvrir un compte pour pouvoir proposer des réponses.
- Défi Turing: 256 exercices de programmation : https://apprendre-en-ligne.net/turing
  Le Défi Turing est une série d'énigmes mathématiques qui pourront difficilement être résolues sans un programme informatique. Attention ! Votre programme devra trouver la réponse en moins d'une minute !
- projecteuler.net
  Le projet Euler est une série de défis, de difficulté croissante, mêlant mathématiques, algorithmique, et programmation. Chaque problème possède une unique solution qu'il s'agit de découvrir par soi-même, ce qui permet d'accéder à un forum consacré aux différentes approches menant à sa résolution.
- www.pythonchallenge.com

- codingame

- Si vous voulez apprendre Python en regardant des vidéos, je vous conseille de visualiser les “Pythonneries” du site du zéro disponibles à cette adresse Pythonneries
  https://www.dailymotion.com/playlist/x22t3u

- Matplotlib :
  - 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

Annexe D.

Annales

1. AA 2019-2020
   1.1. CC du 20 décembre 2019 à la page 198
   1.2. CT du 13 janvier 2020 à la page 201
   1.3. CR du __ juin 2020

D.1. Contrôle Continu du 20 décembre 2019

CATALOGUE DE TOUTES LES QUESTIONS AVEC SOLUTION

L1 MATHS – PIM-11 – CC – 20 décembre 2019 Durée : 2h

- **Une feuille A4 recto-verso manuscrite autorisée, tout autre document interdit.**
- Vous pouvez ouvrir seulement Idle3 ou un éditeur de texte et un terminal. **L'ouverture de tout autre logiciel (navigateur, calculatrice, lecteur pdf ou autre) est strictement interdite.**
- Toutes les questions ont une et une seule bonne réponse. **Des points négatifs seront affectés aux mauvaises réponses.**

Nom et prénom :	Cochez votre ID (par exemple, si ID=15 alors on coche 1 sur la première ligne et 5 sur la deuxième) :
.....	☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
.....	☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

Q. [note] La note d'un examen est calculée selon la formule

$$\frac{3}{10}TP + \max\left\{\frac{7}{10}CT; \frac{1}{2}CT + \frac{1}{5}\overline{CC}\right\}$$

où $\overline{CC}$ est la moyenne arithmétique des notes de CC. Si les notes obtenues sont $CT = 2$, $TP = 8$ et les notes des CC sont $9, 12, 8, 11, 16$, que vaut la note finale (arrondie à 10^{-2} près)?

☐ 3.8 ■ 5.64 ☐ 5.2 ☐ 6.6 ☐ 14.6 ☐ 4.76 ☐ Autre réponse

Solution :

```
TP, CT =8, 2
CClist=[9,12,8,11,16]
CCmoy=sum(CClist)/5
round(0.3*TP+max(0.7*CT,0.5*CT+0.2*CCmoy),2)
```

Q. [suite1] Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_n = \frac{1}{3^n}$ (donc $u_0 = 1$, $u_1 = \frac{1}{3}$ etc.). Quel est le plus petit n tel que $u_n < 10^{-7}$?

☐ 16 ■ 15 ☐ 14 ☐ 17 ☐ 18 ☐ 13 ☐ Autre réponse

Solution : $u_n = \frac{1}{b^n}$ Il s'agit d'une suite géométrique de raison $0 < q = \frac{1}{b} < 1$: elle est donc décroissante. On a

$$u_n < 10^{-p} \iff \frac{1}{b^n} < 10^{-p} \iff \log_{10} b^{-n} < -p \iff n > \frac{p}{\log_{10}(b)} \simeq 14.671322920025693.$$

```
from math import *
p, b =7, 3
n=int(p/log(b,10))+1
TEST:
(1/b)**n =6.969171937625627e-08
(1/b)**(n-1)=2.090751581287688e-07
```

Q. [suite2] Calculer u_{99} si $(u_n)_{n\in\mathbb{N}}$ est la suite définie par $u_0 = 1018$ et $u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ \frac{1+3u_n}{2} & \text{sinon.} \end{cases}$

☐ 7 ☐ 3 ■ 2 ☐ 1 ☐ -1 ☐ 12 ☐ Autre réponse

CATALOGUE DE TOUTES LES QUESTIONS AVEC SOLUTION

Q. [suite4] Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 = 99$, $u_1 = 101$ et $u_{n+2} = u_{n+1} + (-1)^n u_n$. Que vaut u_{77}?

☐ 81943743 ☐ 3899589435 ■ 6309668248 ☐ 10209257683 ☐ Autre réponse

Q. [suite3] Soient $(u_n)_{n\in\mathbb{N}}$ et $(w_n)_{n\in\mathbb{N}}$ les suites définies par $u_0 = 10$, $w_0 = 10$ et $u_{n+1} = w_n - \frac{1}{4}$, $w_{n+1} = u_n + \frac{1}{2}$. Que vaut $u_{97} + w_{97}$?

■ 44.25 ☐ 44.5 ☐ 68.0 ☐ 56.0 ☐ 56.25 ☐ Autre réponse

Q. [comp] Calculer $\sum_{i=0}^{66} i^4$.

☐ 241074977 ■ 260049713 ☐ 4888521 ☐ 280200834 ☐ Autre réponse

Solution :

```
n, p = 66, 4
s=sum([i**p for i in range(n+1)])
```

Q. [comp23] Si on liste tous les entiers naturels compris entre 4 et 20 (inclus) qui sont multiples de 5 ou de 17, on obtient [5,10,15,17,20]. La somme de ces nombres est 67. Trouver la somme de tous les multiples de 5 ou de 7 compris entre 266 et 17787 (inclus).

☐ 4524030 ■ 49710686 ☐ 49692899 ☐ 49710420 ☐ Autre réponse

Q. [comp4] $2^{15} = 32768$ et la somme de ses chiffres vaut $3+2+7+6+8 = 26$. Que vaut la somme des chiffres composant le nombre 3^{940}?

☐ 1944 ■ 1935 ☐ 1953 ☐ 3870 ☐ 1932 ☐ Autre réponse

Solution :

```
b, n = 3, 940
s=sum([int(x) for x in str(b**n)])
```

Q. [comp5] On appellera "miroir d'un nombre n" le nombre n écrit de droite à gauche. Par exemple, `miroir(7423) = 3247`. Si on calcule `7423-miroir(7423)` on obtient 4176 et son avant dernière chiffre est 7.

Si $n = 3^{853}$, que vaut l'avant dernière chiffre de `n-miroir(n)`?

■ 5 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 6 ☐ 7 ☐ 8 ☐ 9

Solution :

```
b, p = 3, 853
n=b**p
f = lambda n : str(n-int(str(n)[::-1]))
solution=f(n)[-2]
```

Q. [fun] Calculer $f(f(f(0.4)))$ si $f(x) = \left(1+\sin\left(\frac{1}{x^9}\right)\right)^{1/2}$.

☐ 1.31148290225 ■ 1.27835967856 ☐ 1.4918246976412703
☐ 1.04259940571 ☐ 2.718281828459045 ☐ Autre réponse

Solution :

```
from math import sin
```

CATALOGUE DE TOUTES LES QUESTIONS AVEC SOLUTION

```
a, b = 0.4, 9
f = lambda x : (1+sin(x**(-b)))**(0.5)
solution=f(f(f(a)))
```

Q. [fun2] Calculer $F(F(F(14,10)))$ si $F(x,y)=(x+y,x-y)$.

■ [48, 8] ☐ [48, -8] ☐ [28, 28] ☐ [20, -20] ☐ [28, 20] ☐ Autre réponse

Solution :

```
a, b = 14, 10
F = lambda x: [x[0]+x[1],x[0]-x[1]]
solution=F(F(F([a,b])))
```

Q. [dess] Soit la fonction $f:\mathbb{R}\to\mathbb{R}$ définie par $f(x)=\dfrac{x^3\cos(x)+x^2-x+16}{x^4-\sqrt{3}x^2+127}$. L'équation $f(x)=0$ a une solution α voisine de 17. Tracer le graphe de la fonction f et, en utilisant le zoom, choisir la meilleure approximation de α parmi les suivantes.

■ 17.220897 ☐ 17.230897 ☐ 17.120897 ☐ 17.270897 ☐ Autre réponse

Solution : Il faut choisir un pas inférieur à l'erreur entre les solutions proposées, à savoir inférieur à 0.01.

Q. [Open]

① Écrire une λ-`function` qui implémente la fonction mathématique $f(x)=\exp(x)$.

② Écrire une `comprehensions-list` qui génère l'ensemble $\{f(x)\mid x=1,2,\ldots,150 \text{ si } x^2 \text{ est divisible par } 7\}$.

③ Calculer la somme de ses éléments.

■-0.25 ■0 ■0.5 ■1 ■1.5 ■2 ■2.5 ■3 Cases réservées au correcteur

.....

.....

.....

.....

D.2. Contrôle Terminal du 13 janvier 2020

CATALOGUE DE TOUTES LES QUESTIONS AVEC SOLUTION

L1 MATHS – PIM-11 – CT – 13 janvier 2020 Durée : 2h

- **Une feuille A4 recto-verso manuscrite autorisée, tout autre document interdit.**
- Vous pouvez ouvrir seulement Idle3 ou un éditeur de texte et un terminal. **L'ouverture de tout autre logiciel (navigateur, calculatrice, lecteur pdf ou autre) est strictement interdite.**
- Toutes les questions ont une et une seule bonne réponse. **Des points négatifs seront affectés aux mauvaises réponses.**

Nom et prénom :	Cochez votre ID (par exemple, si ID=15 alors on coche 1 sur la première ligne et 5 sur la deuxième) :
.....	☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
Questions avec réponses doublées :	☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
.....	

Q. [suiteEXP1] Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_n = \left(1+\frac{2}{n}\right)^n$ (donc $u_1 = 1$, $u_2 = (1+2/2)^2$ etc.). On sait que $\lim_{n\to+\infty} u_n = e^2$. Quel est le plus petit n tel que $|u_n - e^2| < 10^{-2}$?

☐ 1474 ☐ 1475 ■ 1476 ☐ 1477 ☐ 1478 ☐ 1479 ☐ Autre rép.

Q. [suiteH] Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 = 3$ et $u_{n+1} = \frac{1}{2}\left(u_n + \frac{8}{u_n}\right)$. Calculer u_{14} à 5 chiffres significatifs.

☐ 1.82843 ■ 2.82843 ☐ 2.92843 ☐ 3.32843 ☐ 3.57843 ☐ Autre réponse

Q. [suitePAR] Considérons deux espèces: une proie (des lièvres par exemple) et un prédateur (des lynx par exemple). On suppose qu'à ce jour il y a 60000 proies et 9000 prédateurs et on se demande comment vont évoluer les populations de ces deux espèces. On suppose qu'il n'y a aucune autre intervention extérieur. Une modélisation possible pour ce genre de système a été proposé par Lotka et Volterra. Si on note u_n la population de proies et v_n la population de prédateurs à l'instant n, alors on peut modéliser leur évolution par

$$\begin{cases} u_{n+1} = u_n(1+a-bv_n), & \text{équation des proies,} \\ v_{n+1} = v_n(1-c+du_n), & \text{équation des prédateurs.} \end{cases}$$

Nous allons prendre pour valeur $a = 0.09$, $b = 0.000010$, $c = 0.25$ et $d = 0.000005$.
Que vaut u_{29} arrondi à l'entier le plus proche?

■ 45781 ☐ 5778 ☐ 40002 ☐ 46872 ☐ 6559
☐ Autre réponse

Q. [newton1] Soit $f(x) = x^3 - 3$ et $g(x,h) = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h}$. On prend $h = 0.3$. Calculer u_4 à 6 chiffres significatifs si $(u_n)_{n\in\mathbb{N}}$ est la suite définie par $u_0 = 3$ et

$$u_{n+1} = u_n - \frac{f(u_n)}{g(u_n,h)}$$

■ 1.44265 ☐ 1.464123 ☐ 1.632926 ☐ 2.111851 ☐ 1.44225 ☐ Autre réponse

Q. [comp3] Combien d'entiers naturels compris entre 869 et 5232 (inclus) sont divisibles par 24?

☐ 171 ☐ 172 ☐ 181 ■ 182 ☐ 219 ☐ Autre réponse

Q. [comp4] Combien de fois le chiffre 2 apparaît en écrivant les nombres de 47 à 8403 inclus?

☐ 3564 ■ 3566 ☐ 3041 ☐ 3581 ☐ 3585 ☐ Autre réponse

Q. [comp2] Si on liste tous les entiers naturels compris entre $L = 0$ et $R = 999$ (inclus) qui vérifient toutes les propriétés suivantes: l'entier se termine par $t = 3$, la somme des chiffres est supérieure ou égale à $s = 15$, le chiffre des dizaines est pair, on obtient $[483, 583, 663, 683, 763, 783, 843, 863, 883, 943, 963, 983]$. La somme de ces nombres est 9436.

Que vaut cette somme si $L = 26$, $R = 1370$, $t = 5$, $s = 11$?

■ 35805 ☐ 18135 ☐ 9436 ☐ 27871 ☐ 35815 ☐ Autre réponse

Q. [comp] Considérons la série harmonique $S(n) = \sum_{i=1}^{n} \frac{1}{i}$. Soit $T(n)$ la série obtenue en excluant de $S(n)$ toutes les fractions qui contiennent le chiffre 2 au dénominateur (par exemple 1/2 et 1/326). Que vaut $T(382)$ à 10 chiffres significatifs?

☐ 4.7474625154 ■ 4.795860689 ☐ 4.8853755136 ☐ 5.595860689 ☐ Autre réponse

Q. [comp5] Soit `myf(m,n)` une fonction qui prend en entrée deux nombres entiers m et n et renvoie la liste des nombres entiers compris entre 0 et m (inclus) tels que n n'apparaît pas dans l'écriture du nombre. Exemples: si $n = 1$ et $m = 22$, la liste sera $[0, 2, 3, 4, 5, 6, 7, 8, 9, 20, 22]$ car on enlève tous les nombres où apparaît le chiffre 1; si $n = 13$, dans la liste n'apparaîtront pas les nombres qui contiennent 13 comme 130, 4139 ou 313. Par contre les nombres 310 ou 123 seront dans la liste.

Que vaut la somme des éléments de la liste obtenue avec $n = 8$ et $m = 523$?

☐ 42139 ☐ 72302 ☐ 111062 ■ 110565 ☐ 110068 ☐ Autre réponse

Q. [fun1] Soit deux fonctions f et d définies comme suit:

$$f : \mathbb{R} \to \mathbb{R} \qquad d : \mathbb{R}^2 \to \mathbb{R}^2$$
$$x \mapsto f(x) = \cos(x) - x \qquad (a,b) \mapsto d(a,b) = \begin{cases} \left(a, \frac{a+b}{2}\right) & \text{si } f(a) f\left(\frac{a+b}{2}\right) < 0, \\ \left(\frac{a+b}{2}, b\right) & \text{sinon.} \end{cases}$$

Calculer $d(d(d(d(0,2))))$.

☐ $(0, 1.0)$ ☐ $(0.5, 0.75)$ ☐ $(0.6875, 0.75)$
☐ $(0.5, 1.0)$ ■ $(0.625, 0.75)$ ☐ Autre réponse

Q. [fun2] Soit deux fonctions f et d définies comme suit:

$$f : \mathbb{R} \to \mathbb{R} \qquad d : \mathbb{R}^2 \to \mathbb{R}^2$$
$$x \mapsto f(x) = \cos(x) - x \qquad (a,b) \mapsto d(a,b) = \begin{cases} \left(a, \frac{a+b}{2}\right) & \text{si } f(a) f\left(\frac{a+b}{2}\right) < 0, \\ \left(\frac{a+b}{2}, b\right) & \text{sinon.} \end{cases}$$

Calculer $d(d(d(d(0,1))))$.

☐ $(0.5, 1)$ ☐ $(0.625, 0.75)$ ☐ $(0.71875, 0.75)$
☐ $(0.5, 0.75)$ ■ $(0.6875, 0.75)$ ☐ Autre réponse

Q. [fun3] Soit deux fonctions f et d définies comme suit:

$$f : \mathbb{R} \to \mathbb{R} \qquad d : \mathbb{R}^2 \to \mathbb{R}^2$$
$$x \mapsto f(x) = \cos(x) - x \qquad (a,b) \mapsto d(a,b) = \begin{cases} \left(a, \frac{a+b}{2}\right) & \text{si } f(a) f\left(\frac{a+b}{2}\right) < 0, \\ \left(\frac{a+b}{2}, b\right) & \text{sinon.} \end{cases}$$

Calculer $d(d(d(d(0.5, 1.5))))$.

CATALOGUE DE TOUTES LES QUESTIONS AVEC SOLUTION

☐ $(0.5, 1.0)$ ☐ $(0.625, 0.75)$ ☐ $(0.71875, 0.75)$
☐ $(0.5, 0.75)$ ■ $(0.6875, 0.75)$ ☐ Autre réponse

Q. [cesar1] 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 "USHALSDWALSDAW" et la clé de codage est $n = 18$, quelle est la réponse à la question?

■ Rome ☐ Paris ☐ Madrid ☐ Athènes ☐ Bruxelles ☐ Vienne ☐ Berlin

Q. [Open] Écrire les instructions pour:

① implémenter la fonction mathématique $f(x) = \cos(x)$ avec une fonction lambda,

② affecter à la variable `A` la liste de points equidistribuée de $-\pi$ à π avec 651 points,

③ écrire une `comprehensions-list` qui génère l'ensemble $\{ f(x) \mid x \in \mathtt{A} \}$,

④ utiliser les listes précédentes pour afficher le graphe de la fonction $x \mapsto f(x)$.

Écrire exactement les instructions à taper dans un fichier (suggestion: écrivez le script dans un fichier et vérifiez l'exécution). L'utilisation de `def` ou d'autres structures qu'une comprehensions-list sera considérée une faute.

■-0.25 ■0 ■0.5 ■1 ■1.5 ■2 Cases réservées au correcteur

Liste des Exercices

Exercices

Exercices ☆

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 réponse par vous même ☺