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

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

Année 2019 – 2020

Dernière mise-à-jour
Vendredi 20 septembre 2019

Programme Indicatif

Semaine	CM (lundi)	TD (lundi)	TP (vendredi)
41	Ch. 1-2	–	–
42	–	Préparation TP 1-2 (ch. 1-2)	–
43	Ch. 3-4	–	1.1 - 1.2 - 1.3 - 1.4 - 1.5 - 1.6 - 1.7 - 1.8 - 1.9 - 1.10 - 1.11 - 1.12 - 1.13 - 1.14 - 1.15 1.16 - 1.17 - 1.18
45	Ch. 5-6	Préparation TP 3-4 (ch. 3-4)	2.1 - 2.2 - 2.3 - 2.4 3.1 - 3.2 - 3.3 - 3.4 - 3.5 - 3.6 - 3.7
46	–	–	4.1 - 4.2 - 4.3 - 4.4 - 4.5 - 4.6 - 4.7 - 4.8 - 4.9 - 4.10 - 4.11 - 4.12 - 4.13 - 4.14 - 4.15 - 4.16 4.17 - 4.18 - 4.19 - 4.20 - 4.21 - 4.22
47	–	Préparation TP 5-6 (ch. 5-6)	5.1 - 5.2 - 5.3 - 5.4 - 5.5 5.6 - 5.7 - 5.8 - 5.9 - 5.10 - 5.11 - 5.12 - 5.13 - 5.14 - 5.15 - 5.16
48	Ch. 7	–	6.1 - 6.2 - 6.3 - 6.4 - 6.5 - 6.6 - 6.7 - 6.8 - 6.9 - 6.10
49	–	Préparation TP 7 (ch. 7)	6.11 - 6.12 - 6.13 - 6.14 - 6.15 - 6.16 - 6.17 - 6.18 - 6.19 - 6.20 - 6.21 - 6.22 - 6.23 - 6.24 - 6.25 - 6.26 - 6.27 - 6.28 - 6.29 6.30 - 6.31 - 6.32 - 6.33 - 6.34 - 6.35 - 6.36 - 6.37
50	–	–	7.1 - 7.2 - 7.3 - 7.4 - 7.5 - 7.6 7.7 - 7.8 - 7.9 - A.1 - A.2 - A.3 - A.4 - A.5 - A.6 - A.7
51			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)
M43 au semetre 4 (L2) : *TP mathématiques* (notebook IPython)
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 de difficulté variée et dont la correction est disponible sur ma page web. Cependant, veuillez noter que le seul moyen de comprendre et d'apprendre est d'**essayer par soi-même !** 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.

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).

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/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.

On peut utiliser un simple éditeur de texte et un terminal ou des environnements spécialisés comme Idle ou Spyder. Ces derniers se composent d'une fenêtre appelée indifféremment *console*, *shell* ou *terminal* Python.

Chapitre 1.

Notions de base de Python

Python est un langage développé dans les années 1 980 (le nom est dérivé de la série télévisée britannique des *Monty Python's Flying Circus*). Il est disponible pour tous les principaux systèmes d'exploitation (Linux, Unix, Windows, Mac OS, etc.). Un programme écrit sur un système fonctionne sans modification sur tous les systèmes. Les programmes Python ne sont pas compilés en code machine, mais sont gérés par un interpréteur. Le grand 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.

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 `python` 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

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.

```
>>> 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 naviguer dans l'historique des instructions saisies dans l'INTERPRÉTEUR on peut utiliser les raccourcis ↑et ↓.
- ▷ 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.

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=10
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 = 10
```

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. 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.).

1.3. 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.

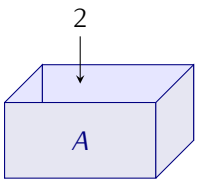
1.4. Variables et affectation

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 $A \leftarrow 2$).

ATTENTION
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)
```

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

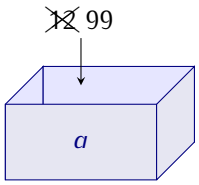
Les noms de variables peuvent être non seulement des lettres, mais aussi des mots ; ils peuvent contenir des chiffres (à condition toutefois de ne pas commencer par un chiffre), ainsi que certains caractères spéciaux comme le tiret bas «_» (appelé *underscore* en anglais).

Cependant, certains mots sont réservés :

```
and as assert break class continue def del elif else except False finally for from
global if import in is lambda not or pass raise 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 :

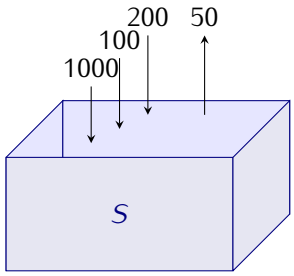
```
>>> a = 12
>>> a = 99
>>> print(a)
99
```



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(b)` 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 $2.1e-23$), 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
```

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
```

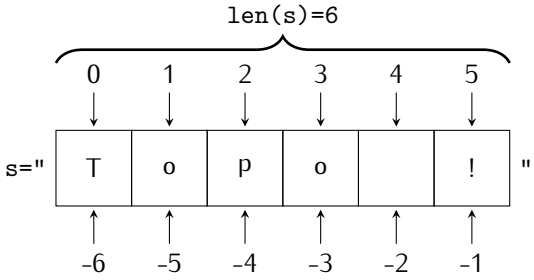
Si on tente d'extraire un élément avec un index dépassant la taille de la chaîne, Python renvoie un message d'erreur :

```
>>> s = 'Topolino'
>>> print(s[8])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

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
```

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'
```

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

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

`str(n)`	transforme un nombre `n` en une chaîne de caractères
`s1 + s2`	concatène la chaîne `s1` à la chaîne `s2`
`s.index("x")`	renvoie l'indice de la **première** occurrence de l'élément `"x"` dans la chaîne `s`
`s.count("x")`	renvoie le nombre d'occurrence de l'élément `"x"` dans la chaîne `s`
`len(s)`	renvoie le nombre d'éléments de la chaîne `s`
`"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

Exemples :

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

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

>>> s = string1 + string2
>>> print(s)
Press return to exitthe program !

>>> s = string1 + ' ' + string2
>>> print(s)
Press return to exit the program !

>>> print(s.index("r"))
1
>>> print(s.count("r"))
5
>>> print(len(s))
34
>>> print("g" in s)
True
>>> print("x" in s)
True
```

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]
```

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]
```

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},...')` Voici quelques exemples pour le choix dès règles de formatage :

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

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

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

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

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

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

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

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

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

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

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

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

>>> print('{:+.2f}'.format(a)) # affiche toujours le signe
-1234.57
>>> 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
>>> print(f'n={n:+.2f} a={a:+.2f}')
n=+9876.00 a=-1234.57
```

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. Exercices

Exercice 1.1 (Initiation Linux)

1. **Utiliser l'interface graphique** :
   - ▷ lancer un programme dont l'icône est sur le bureau
   - ▷ lancer un programme à partir du menu (Gedit, Geany, Firefox, Libre Office Calc)
   - ▷ gestion des fichiers (nautilus)
2. **Utiliser un terminal/console**.
   - ▷ Ouvrir un terminal. Taper la commande `gedit` et appuyez sur la touche "Entrée". 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`)
   - ▷ 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**ist 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
     - ▷ Utiliser l'auto-complétion : pour éviter d'avoir à saisir des noms de fichier en entier, il est possible d'utiliser la touche tabulation [TAB]. Un appui sur [TAB] 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 [TAB] pour les lister toutes.
   - ▷ Tester les commande ci-dessous :

```
mkdir PIM11-TP1
ls
cd PIM11-TP1
touch first.py
ls
```

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).

Voici, par exemple, une succession d'instructions à saisir en mode interactif. 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)
```

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 `python fichier.py`.

1. Écrire et exécuter un script par ligne de commande :
   1.1. Ouvrir un éditeur de texte pur (par exemple Gedit ou Geany, 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 `python first.py` puis appuyer sur la touche « Entrée ».
2. Rendre un script exécutable :
   2.1. Ouvrir le fichier `first.py` dans un éditeur de texte pur.
   2.2. Modifier ce fichier en ajoutant comme premières lignes :
   ```
   #! /usr/bin/env python3
   # coding: utf-8
   ```
   2.3. Sauvegarder le fichier.
   2.4. Ouvrir un terminal et y écrire `chmod +x first.py` puis appuyer sur la touche « Entrée ».
   2.5. Dans le même terminal écrire `./first.py` puis appuyer sur la touche « Entrée ». [Il n'est plus nécessaire d'écrire `python first.py`]
3. Utiliser un IDE :
   3.1. Lancer le logiciel Idle3, ouvrir le fichier `first.py` puis appuyer sur la touche « F5 » (on peut enlever les deux première lignes)

Exercice 1.4 (Devine le résultat – affectations)

Quel résultat donnent les codes suivants ?

```
a=1
b=100
b=a
a=b
print("a =",a," b =",b)
```

```
a=1
b=100
a=b
b=a
print("a =",a," b =",b)
```

Exercice 1.5 (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
y=2*x
print(x)
x=x+3*y
x=7
```

Exercice 1.6 (Devine le résultat – `string`)

Quel résultat donne le code suivant ?

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

```
print(s+" TEST")
```

Exercice 1.7 (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]`.

Exercice 1.8 (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.

Exercice 1.9 (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.

Exercice 1.10 (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.

Exercice 1.11 (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 !) :

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.

Exercice 1.12 (Nombre de chiffre)
Pour $n \in \mathbb{N}$ donné, calculer le nombre de chiffres qui le composent.

Exercice 1.13 (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.

Exercice 1.14 (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.

Exercice 1.15 (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 "

☆ Exercice Bonus 1.16 (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 "

☆ Exercice Bonus 1.17 (Changement de casse)

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()
```

☆ Exercice Bonus 1.18 (Comptage et recherche)

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')
```

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. Une liste n'est pas forcement homogène : elle peut contenir des objets de types différents les uns des autres. 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]`. Voici un exemple :

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

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

```
>>> print(fraise[2])
18
```

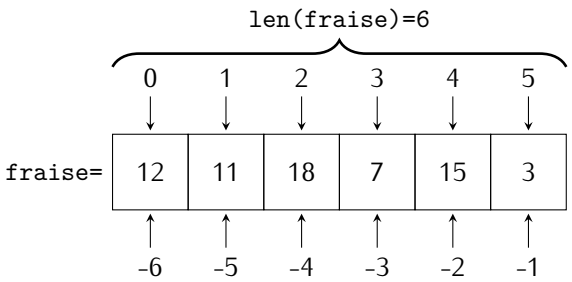
On peut modifier les éléments d'une liste :

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

Si on tente d'extraire un élément avec un index 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.

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

`len(a)`	renvoie le nombre d'éléments de la liste `a`
`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
`max(a)`	renvoi le plus grand élément de la liste `a`
`min(a)`	renvoi le plus petit élément de la liste `a`

Attention : **les instructions suivantes modifient la liste !**

`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
`a.reverse()`	modifie la liste `a` en inversant les éléments

```
>>> 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]
>>> print(len(a))
8
>>> a.insert(2,7)
>>> print(a)
[2, 17, 7, 20, 21, 34, 37, 83, 100]
>>> a[0] = 21
>>> print(a)
[21, 17, 7, 20, 21, 34, 37, 83, 100]
>>> a[2:4] = [-2,-5,-1978]
>>> print(a)
[21, 17, -2, -5, -1978, 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]
```

Attention

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 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.

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])
2.0
>>> 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.

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

2.2. 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. La fonction `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 `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["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. 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[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")
```

Exercice 2.2 (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.

Exercice 2.3 (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).

Exercice 2.4 (Range)

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

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

Chapitre 3.

Structure conditionnelle

Supposons vouloir calculer la valeur absolue d'un nombre x :

$$|x| = \begin{cases} x & \text{si } x \geq 0, \\ -x & \text{sinon.} \end{cases}$$

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 est la suivante :

```
if condition_1:
    instruction_1.1
    instruction_1.2
    ...
elif condition_2:
    instruction_2.1
    instruction_2.2
    ...
...
else:
    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`... 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 après le mot clé `else`.

Voici un exemple pour établir 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.

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) tel que le code affichera `Atchoum`.
3. Même question pour `Simplet`.

Exercice 3.3 (Calculer $|x|$)

Afficher la valeur absolue d'un nombre x sans utiliser la fonction `abs`.

Exercice 3.4 (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 $\texttt{IMC}=\frac{\texttt{masse}}{\texttt{taille}\times\texttt{taille}}$.

Exercice 3.5 (Note ECUE)

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

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

Écrire un script qui calcule la note finale dans les cas suivants (vérifier les résultats !) **sans utiliser la fonction `max`** :

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

Exercice 3.6 (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`

Exercice 3.7 ($ax^2 + bx + c = 0$)

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. $a = 1$, $b = 0$, $c = 4$

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`.

Chapitre 4.

Structures itératives

Les structures de répétition se classent en deux catégories : les *répétitions conditionnelles* pour lesquelles le bloc d'instructions est à répéter autant de fois qu'une condition est vérifiée, et les *répétitions inconditionnelles* pour lesquelles le bloc d'instructions est à répéter un nombre donné de fois.

4.1. 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`.

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

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
```

4.2. 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
```

4.3. ★ 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.

```
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)

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
```

Exercice 4.2 (Devine le résultat)

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
```

Exercice 4.3 (Puissance)

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

Exercice 4.4 (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)`.

Exercice 4.5 (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.

Exercice 4.6 (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 $u_n < 10^{-4}$?

Exercice 4.7 (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` :

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

Exercice 4.8 (Approximation de π)
Soit

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

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 obtenue comme suit

```
>>> from math import pi
>>> pi
3.141592653589793
```

Exercice 4.9 (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

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

.

Exercice 4.10 (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

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

Exercice 4.11 (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]]
    # L.append(L_1[i]).append(L_2[i])
print(L_1)
print(L_2)
print(L)
```

Exercice 4.12 (Happy Birthday)

Soit les chaînes de caractères suivantes :

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

Imprimer la chanson *Happy birthday*.

Exercice 4.13 (Cadeaux)

Vous recevez un cadeau de 1 centime aujourd'hui. 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.

Exercice 4.14 (Affichage)

Pour $n \in \mathbb{N}$ donné, afficher $1 + 2 + \cdots + n = N$ où N est à calculer.

Exercice 4.15 (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 impairs.

Exercice 4.16 (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`.

☆ Exercice Bonus 4.17 (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 affecter à la variable `j` la valeur que le joueur tape au clavier on écrira
`j=int(input('Quel nombre proposes-tu ?'))`

☆ Exercice Bonus 4.18 (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.

☆ Exercice Bonus 4.19 (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 grand grand nombre inférieur à 10 millions ayant la propriété : `miroir(n) = 4n` ?

☆ Exercice Bonus 4.20 (Défi Turing n°13)

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. Le plus petit carré palindrome ayant un nombre pair de chiffres est $698896 = 836^2$. Quel est le carré palindrome suivant ?

☆ Exercice Bonus 4.21 (Défi Turing n°43)

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.

☆ Exercice Bonus 4.22 (Défi Turing n°21)

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).

Chapitre 5.

List-comprehensions

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 :

$$\begin{array}{ccccccc} \{ & f(x) & | & x & \in & E & \} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \texttt{[} & \texttt{f(x)} & \texttt{for} & \texttt{x} & \texttt{in} & \texttt{E} & \texttt{]} \end{array}$$

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]
>>> [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 = 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] # idem que range(1,8)
>>> print([x**2 for x in E]) # carres
[1, 4, 9, 16, 25, 36, 49]
>>> print([x for x in E if x%2 == 0]) # pairs
[2, 4, 6]
>>> print([x**2 for x in E if x**2%2 == 0]) # carres pairs
[4, 16, 36]
>>> print([x for x in [a**2 for a in E] if x%2 == 0]) # idem
[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` :

```
>>> print( [1/n for n in range(1,5)] )
[1.0, 0.5, 0.3333333333333333, 0.25]
```

5.1. Exercices

Exercice 5.1 (Comprehensions-list : liste des diviseurs)
Pour un entier $n \in \mathbb{N}$ donné, calculer la liste de ses **diviseurs**.

Exercice 5.2 (Comprehensions-list : nombres parfaits)
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).

- ▷ 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,

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

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

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

Classer tous les nombres $n \leq 100$ en trois listes.

Exercice 5.3 (Comprehensions-list : 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.

Exercice 5.4 (Comprehensions-list : 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.

Exercice 5.5 (Comprehensions-list : 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.

Exercice 5.6 (Comprehensions-list : 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)...
```

Exercice 5.7 (Comprehensions-list : 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.

Exercice 5.8 (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 €.

Exercice 5.9 (Comprehensions-list : années bissextiles)

Depuis l'ajustement du calendrier grégorien, l'année sera bissextile si l'année est divisible par 4 et non divisible par 100, ou est divisible par 400. Ainsi,

- ▷ 2019 n'est pas bissextile car non divisible
- ▷ 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.

Exercice 5.10 (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.

Exercice 5.11 (Table de multiplication)

Afficher la table de multiplication par $1, \dots, 10$ 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

Exercice 5.12 (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.

Exercice 5.13 (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.

☆ Exercice Bonus 5.14 (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} ?

☆ Exercice Bonus 5.15 (Défi Turing n°29)

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

$2^2 = 4$,	$2^3 = 8$,	$2^4 = 16$,	$2^5 = 32$
$3^2 = 9$,	$3^3 = 27$,	$3^4 = 81$,	$3^5 = 243$
$4^2 = 16$,	$4^3 = 64$,	$4^4 = 256$,	$4^5 = 1024$
$5^2 = 25$,	$5^3 = 125$,	$5^4 = 625$,	$5^5 = 3125$

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 $2 \leq b \leq 1000$?

☆ Exercice Bonus 5.16 (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.

Chapitre 6.

Fonctions et Modules

6.1. 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`).

```
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 :

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

▷ il manque l'indentation :

```
>>> def f(x):
... return 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
  File "<stdin>", line 2
    return 2*x**7-x**6+5*x**5-x**4+9*x**3+7*x**2+8*x-1
         ^
IndentationError: expected an indented block
```

▷ 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))
print(x)
```

```
Traceback (most recent call last):
  File "py_stderr_6.1.py", line 6, in <module>
    print(x)
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.

6.1.1. 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` :

$$\underset{\texttt{lambda x}}{x} \quad \underset{\texttt{:}}{\mapsto} \quad \underset{\texttt{2*x}}{2x}$$

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]
```

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`

6.2. 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).

6.2.1. 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))
```

Les deux modules contiennent une fonction `sqrt`. Laquelle sera utilisée à la troisième instruction ? Python va simplement prendre 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 :

```
Traceback (most recent call last):
  File "py_stderr_6.2.py", line 3, in <module>
    print('Racine carrée de -4 =', sqrt(-4))
ValueError: math domain error
```

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.

6.2.2. 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.[2]

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")
```

```
Avec += temps d'exécution=7.089948s
Avec append temps d'exécution=6.534868s
Avec comprehensions-list temps d'exécution=2.272996000000001s
```

On en conclut que l'utilisation des comprehensions-list est à privilégier et que la méthode `append` est plus efficace que l'instruction `+=`.

Le module `math`

Dans Python seulement quelque fonction mathématique est prédéfinie :

2. 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.

`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.

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.choice(seq)`	choisit un éléments aléatoirement dans la liste `seq`
`random.random()`	renvoie un *décimal* aléatoire dans $[0; 1[$
`random.uniform(a,b)`	choisit un *décimal* aléatoire dans $[a; b]$

```
>>> import random
>>> random.randrange(50,100,5)
60
>>> random.randint(50,100)
83
>>> random.choice([1,7,10,11,12,25])
1
>>> random.random()
0.8640302573584399
>>> random.uniform(10,20)
11.096509325013416
```

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

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]
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)
print("Integrale =",integr[0], " Erreur =",integr[1] )

f = lambda x : exp(-x*x)
print(integrate.quad(f, 0,inf)[0])
print(sqrt(pi)/2)
```

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

⭐ 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 *
init_printing()
var('x,y')
# Calculs
s=(x+y)+(x-y)
print("$$",latex(s),"$$")
# Simplifications
expr=sin(x)**2 + cos(x)**2
print("$$",latex(expr),"$$")
print("$$",latex(simplify(expr)),"$$")
expr=(x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
print("$$",latex(expr),"$$")
print("$$",latex(simplify(expr)),"$$")
# Fractions
expr=(x**3-y**3)/(x**2-y**2)
print("$$",latex(expr),"$$")
print("$$",latex(cancel(expr)),"$$")
```

$$2x$$

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

$$1$$

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

$$x - 1$$

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

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

Encore un exemple :

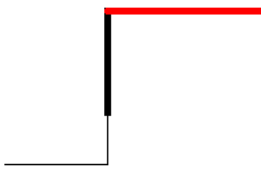
```
functions = [sin(x), cos(x), tan(x)]
print(r"\begin{align*}")
for f in functions:
    d = Derivative(f, x)
    i = Integral(f, x)
    print(latex(d) + "&=" + latex(d.doit()) +"&" + latex(i) + "&=" + latex(i.doit()) + r"\\")
print(r"\end{align*}")
```

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

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

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()
```

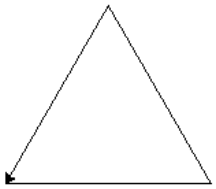
Les principales fonctions mises à disposition dans le module `turtle` sont les suivantes :

- ▷ `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()
```

6.3. Exercices

Exercice 6.1 (Module `math`)
Calculer

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

Exercice 6.2 (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.

Exercice 6.3 (Défi Turing n°6 – 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!

Exercice 6.4 (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.

Exercice 6.5 (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.

Exercice 6.6 (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.

Exercice 6.7 (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 $f(0,0,0)$, $f(-2,0,3)$, $f(-2,3,0)$, $f(2,0,2)$, $f(1,0,2)$, $f(3,-1,-2)$.

Exercice 6.8 (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.

Exercice 6.9 (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 ?

Exercice 6.10 (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)
```

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 a et b (inclus).

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 a et b (inclus) qui sont divisibles par n.

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.

Exercice 6.14 (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).

Exercice 6.15 (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).

Exercice 6.16 (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.

Exercice 6.17 (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.

Exercice 6.18 (Voyelles)

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

Exercice 6.19 (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'aï d'âge mûr et cætera !"

Exercice 6.20 (sin cos)

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

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

Exercice 6.21 (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]`.

Exercice 6.22 (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

Exercice 6.23 (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 ?

Exercice 6.24 (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.

Exercice 6.25 (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` ?

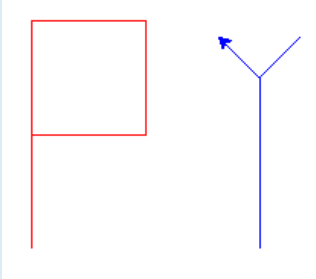
Exercice 6.26 (Approximation valeur ponctuelle dérivées)

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 $x = \frac{\pi}{2}$

Exercice 6.27 (Turtle - lettres)
Écrire un script qui trace l'image suivante :



Exercice 6.28 (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 $n = 3, 4, 5, \ldots, 14$.

Exercice 6.29 ($f : R \to \mathbb{R}^2$)
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) )
```

☆ Exercice Bonus 6.30 (Comprehensions-list : 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,

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

donc 220 et 284 sont amicaux.

Trouvez la somme de tous les nombres amicaux inférieurs à 10000.

☆ Exercice Bonus 6.31 (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.

☆ Exercice Bonus 6.32 (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.

☆ Exercice Bonus 6.33 (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 €

☆ Exercice Bonus 6.34 (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 $a + b + c = 1000$.

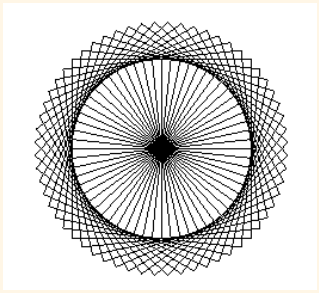
☆ Exercice Bonus 6.35 (Défi Turing n°4 – nombre palindrome)

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 ?

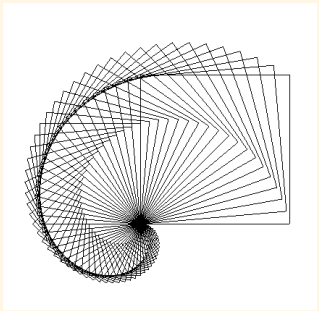
☆ Exercice Bonus 6.36 (Turtle - carrés en cercle)

Écrire un script qui dessine 60 carrés, chacun tourné de 6 degrés, pour obtenir :



☆ Exercice Bonus 6.37 (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 :



Chapitre 7.

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`.

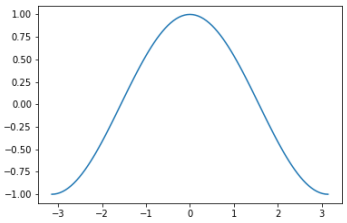
Dans le premier cas on pourra utiliser les commande "à la MATLAB", comme par exemple

```
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`**.

7.1. 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.

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

$$[a, a+h, a+2h, \ldots, b = a+nh] \quad \text{avec } h = \frac{b-a}{n}$$

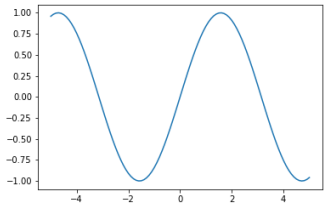
▷ soit l'instruction `arange(a,b,h)` qui construit la liste de $n = E(\frac{b-a}{h}) + 1$ éléments

$$[a, a+h, a+2h, \ldots, a+nh]$$

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()
```



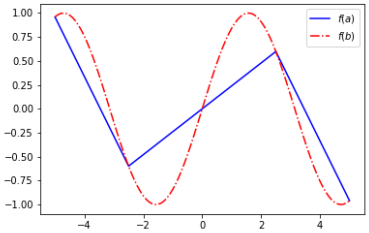
On obtient une courbe sur laquelle on peut zoomer, modifier les marges et sauvegarder dans différents formats.

7.1.1. 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 :

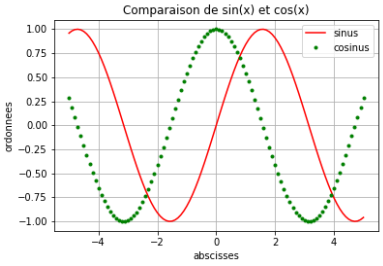
```
from matplotlib.pylab import *
a = linspace(-5,5,5)
fa = sin(a) # matplotlib importe numpy qui redefinie
            # les fonctions de base pour pouvoir
            # les appliquer a un vecteur
plot(a,fa,'b-',label="$f(a)$")
b = linspace(-5,5,101)
fb = sin(b)
plot(b,fb,'r-.',label="$f(b)$" )
legend()
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.

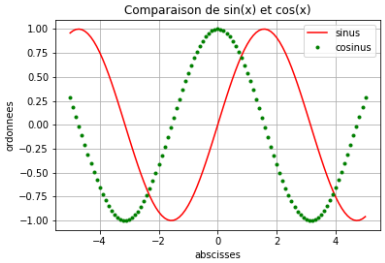
Par exemple, dans le code ci-dessous `"r-"` indique que la première courbe est à tracer en rouge (red) avec un trait continu, et `"g."` que la deuxième est à tracer en vert (green) avec des points.

```
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 :

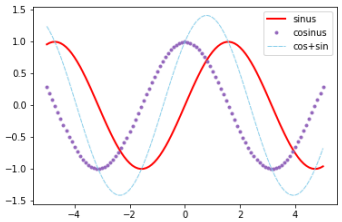
	linestyle=		color=		marker=
-	solid line	r	red	.	points
–	dashed line	g	green	,	pixel
:	dotted line	b	blue	o	filled circles
-.	dash-dot line	c	cyan	v	triangle down
(0,(5,1))	densely dashed	m	magenta	^	triangle up
		y	yellow	>	triangle right
		w	white	<	triangle left symbols
		k	black	*	star
				+	plus
				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

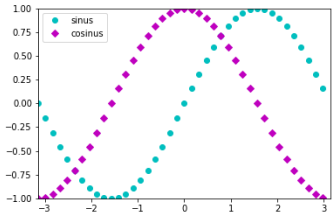
Voici un exemple d'utilisation :

```
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` :

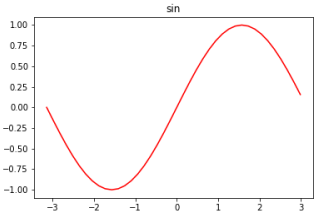
```
from matplotlib.pylab import *
x = arange(-pi,pi,0.05*pi)
plot(x,sin(x),'co',x,cos(x),'mD')
legend(['sinus','cosinus'],loc='upper left')
axis([-pi, pi, -1, 1]); # axis([xmin, xmax, ymin, ymax])
show()
```

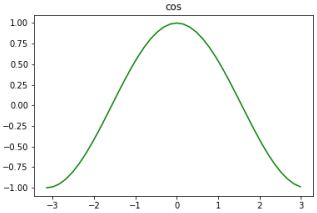


7.1.2. Plusieurs "fenêtres" graphiques

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()
```

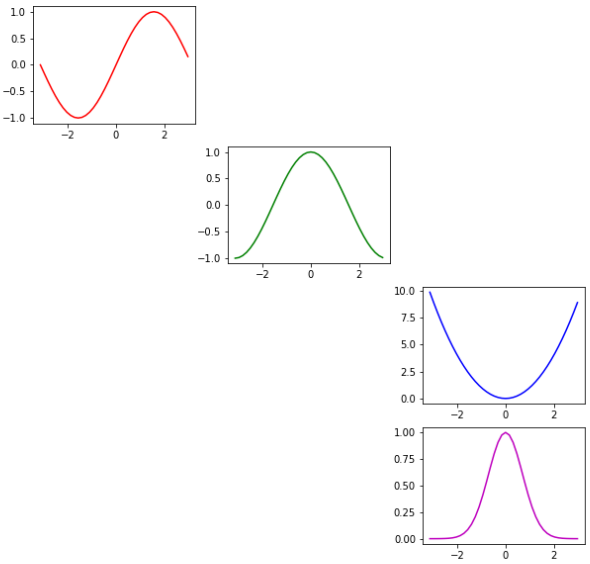




7.1.3. Plusieurs repères dans la même fenêtre

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')
```



7.2. 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 :

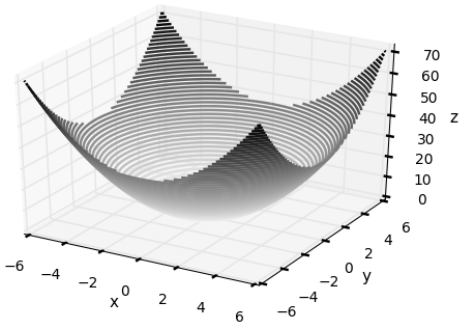
$$f(x, y) = x^2 + y^2$$

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.

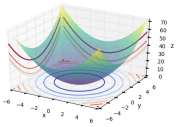
Pour tracer la surface de f on utilisera la fonction `contour3D` du module `mpl_toolkits` :

```
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')
```



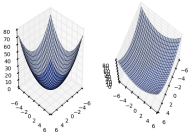
On peut ajouter des courbes de niveau dans chaque direction :

```
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='non
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)
```



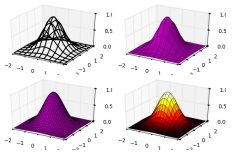
On peut changer l'angle de vue :

```
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()
```



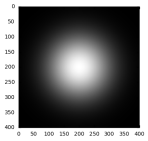
On peut changer l'apparence de la surface :

```
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()
```



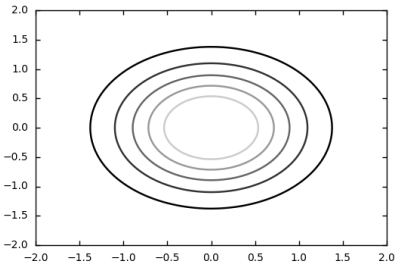
On peut regarder la projection de la surface sur le plan d'équation $z = 0$:

```
imshow(Z)
```

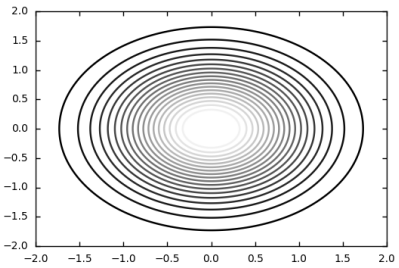


Pour tracer l'évolution de f en lignes de niveaux on utilisera les fonctions `contour` et `contourf` avec comme arguments les variables x et y, les valeurs de z correspondantes ainsi que le nombre de lignes de niveaux choisit.

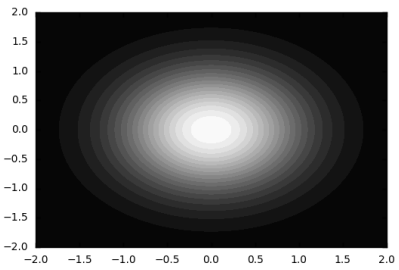
```
h = contour(X,Y,Z)
show()
```



```
h = contour(X,Y,Z,20)
show()
```



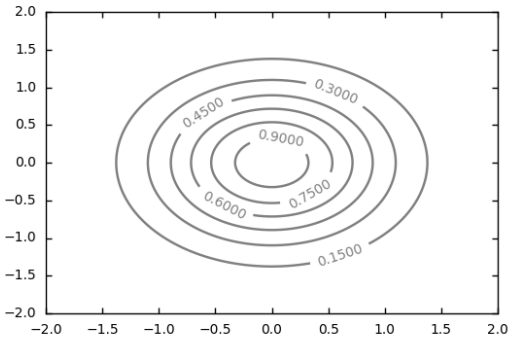
```
h = contourf(X,Y,Z,20)
show()
```



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()
```



7.3. Exercices

Exercice 7.1 (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.

Exercice 7.2 (Tracer une fonction définie par morceaux)

Tracer le graphe de la fonction

$$\begin{aligned} f : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto |x+1| - \left|1 - \frac{x}{2}\right| \end{aligned}$$

Résoudre d'abord graphiquement puis analytiquement $f(x) = 0$ et $f(x) \leq 4$.

Exercice 7.3 (Résolution graphique d'une équation)

Soit la fonction

$$\begin{aligned} f : [-10, 10] &\to \mathbb{R} \\ x &\mapsto \frac{x^3\cos(x) + x^2 - x + 1}{x^4 - \sqrt{3}x^2 + 127} \end{aligned}$$

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.

Exercice 7.4 (Tracer plusieurs courbes)

Tracer dans le même repère le graphe des fonctions $f_n(x) = 2n + \cos(nx)$ pour $n = 1, 2, 3, 4$.

Exercice 7.5 (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

Exercice 7.6 (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.

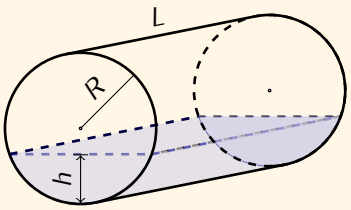
Ce problème est couramment appelé Conjecture de Syracuse (mais aussi problème de Syracuse, algorithme de Hasse, problème de Ulam, problème de Kakutani, conjecture de Collatz, conjecture du $3n+1$). Vous pouvez lire l'article de vulgarisation https://automaths.blog/2017/06/20/la-conjecture-de-syracuse/amp/

a. Dès que $u_i = 1$ pour un certain i, la suite devient périodique de valeurs 4, 2, 1

☆ Exercice Bonus 7.7 (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.

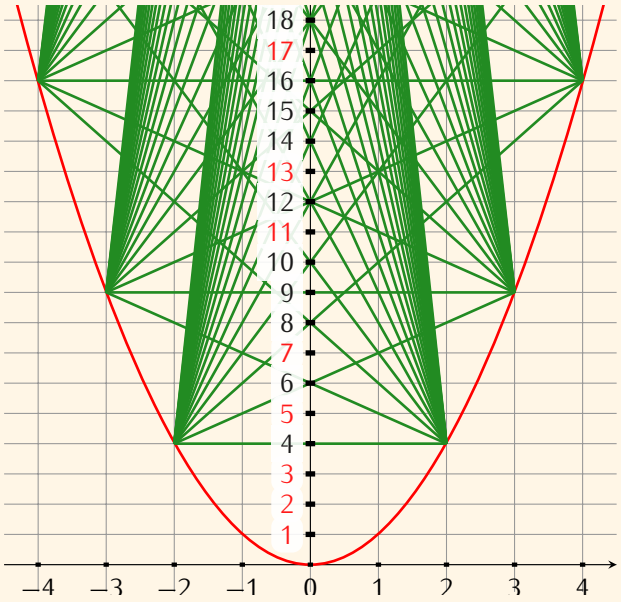


⭐ Exercice Bonus 7.8 (Le crible de Matiyasevitch)

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.

⭐ Exercice Bonus 7.9 (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).

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) $= 0{,}0001100110011001100110011\ldots$ (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)
```

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 $B = \frac{2^{2019}+1}{4^{2019}+2^{2019}+1}$?

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
```

Exercice A.4 (Suites)
Calculer analytiquement et numériquement les premiers 100 termes des suites suivantes :

$$\begin{cases} u_0 = \frac{1}{4}, \\ u_{n+1} = 5u_n - 1, \end{cases} \qquad \begin{cases} v_0 = \frac{1}{5}, \\ v_{n+1} = 6v_n - 1, \end{cases} \qquad \begin{cases} w_0 = \frac{1}{3}, \\ w_{n+1} = 4v_w - 1. \end{cases}$$

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 ?

Exercice A.6 (Évaluer la fonction de Rump)
Évaluer au point $(x, y) = (77617, 33096)$ la fonction de deux variables suivante :

$$f(x,y) = \frac{1335}{4}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11}{2}y^8 + \frac{x}{2y}$$

Exercice A.7 (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 $n \to +\infty$.

Annexe B.

Ressources

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 liste de ressources qui pourraient vous être utiles.

- Gerard Swinnen. *Apprendre à programmer avec Python 3*. 2012. ISBN : 978-2-212-13434-6. URL : http://inforef.be/swi/download/apprendre_python3_5.pdf et compléments http://inforef.be/swi/python.htm
- Arnaud Bodin. *Python au lycée (tome 1) : Algorithmes et programmation (Livres Exo7) (French Edition)*. 2018. ISBN : 1091267510. URL : http://exo7.emath.fr/cours/livre-python1.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

- 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
- 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
- www.pythonchallenge.com

- 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