

Frédéric Butin

Classes  
prépas  
scientifiques

1<sup>re</sup> et 2<sup>e</sup>  
années

# ITC

## Informatique tronc commun

Cours et exercices corrigés



ellipses

# Chapitre 1

## Tutoriel d'initiation à Python

Le langage Python<sup>1</sup> est un langage interprété<sup>2</sup>, créé en 1989 par Guido van Rossum, et utilisé dans de nombreux domaines. Sa syntaxe simple en fait un langage facile à lire et à comprendre. Il a en outre un intérêt tout particulier dans le domaine scientifique, grâce aux nombreux modules qui permettent d'utiliser et de développer des méthodes numériques, mais aussi de faire du calcul formel.

Ce chapitre présente une initiation à Python sous la forme d'un tutoriel qui, au fil de l'apprentissage, deviendra un aide-mémoire facilement consultable. De même qu'une première étape dans l'apprentissage d'une langue est la répétition de ce qui est entendu, de même ici la meilleure façon d'étudier ce tutoriel est de saisir au clavier tous les exemples donnés pour s'approprier ainsi les différentes commandes de Python.

### 1.1 Installation et premier exemple

#### 1.1.1 Installation de Python

Pour installer Python, on peut télécharger le fichier `Winpython64-3.9.10.0.exe` en cliquant sur le lien ci-dessous et l'exécuter.

```
https://sourceforge.net/projects/winpython/files/  
WinPython\_3.9/3.9.10.0/Winpython64-3.9.10.0.exe/download
```

Ce fichier est aussi accessible sur le site <https://winpython.github.io/>.

---

1. On trouve dans le langage Python de nombreuses références à la série des Monty Python, chère à l'inventeur de ce langage!

2. Comme son nom l'indique, un tel langage est interprété par un logiciel appelé **interpréteur** qui exécute ligne à ligne le code avec les données d'entrée et calcule les données de sortie. A l'inverse, dans un langage compilé, le code source est d'abord transformé en un code binaire par un logiciel appelé **compilateur**; ensuite, le système d'exploitation calcule les données de sortie grâce à ce code et aux données d'entrée. Un programme en langage compilé s'exécute en général plus rapidement qu'un programme en langage interprété. Cependant, un langage interprété présente aussi l'intérêt de fonctionner directement sur tout ordinateur dès qu'il possède l'interpréteur adéquat.

Cette distribution de Python contient tous les modules nécessaires au contenu du livre, en particulier `numpy` (pour les tableaux et l'algèbre linéaire), `scipy` (pour les méthodes numériques), `matplotlib` (pour les tracés graphiques), `sqlite3` (pour les bases de données), `sklearn` (pour l'intelligence artificielle), `pandas` (pour le traitement des données), `sympy` (pour le calcul formel), `cx_Freeze` (pour convertir un fichier Python en exécutable). Elle contient également les éditeurs `Idle`, `Spyder`, `Pyzo` et le notebook `Jupyter`<sup>3</sup>.

### 1.1.2 Premier exemple

Quand on ouvre le terminal `Idle`, le symbole `>>>` apparaît à l'écran. On peut alors saisir une instruction que l'on exécute avec la touche "Entrée".

Calculer  $4 \times 13 - 1$ .

Une nouvelle ligne commençant par `>>>` apparaît alors. Calculer  $\frac{4}{3}$ .

On remarque que l'on obtient une valeur approchée en utilisant le symbole `/`. Si l'on souhaite avoir la division euclidienne de 4 par 3, on devra écrire `4//3`. Quant au reste, il est obtenu en écrivant `4%3`. Notons aussi que la commande `divmod(a, b)` renvoie le couple formé du quotient et du reste de la division euclidienne de  $a$  par  $b$ . Pour accéder aux instructions précédemment écrites, on utilise simultanément les touches "alt" et "P".

La commande `print` permet d'afficher un résultat et la commande `input` de récupérer des données saisies au clavier.

```
prenom = input('Quel est votre prénom ? ')
```

On affiche alors le résultat.

```
print('Bonjour ' + str(prenom))
```

### 1.1.3 Modules

On peut importer des **modules** qui ne sont pas accessibles par défaut quand on lance Python. Pour calculer  $\sin(\frac{\pi}{2}) + \sqrt{2} - e$ , on importe par exemple le module `math` qui contient un certain nombre de fonctions mathématiques. Voici trois façons de procéder (et les trois sont utiles).

— Soit on écrit `from math import *` (cela importe tout le module), puis l'expression `sin(pi/2)+sqrt(2)-e`.

— Soit on écrit `from math import sin, pi, sqrt, e` (seules les commandes `sin`, `pi`, `sqrt`, `e` sont importées), puis `sin(pi/2)+sqrt(2)-e`.

— Soit on écrit `import math as mt` (cela importe tout le module et ses fonctions devront être préfixées par `mt`), puis `mt.sin(mt.pi/2)+mt.sqrt(2)-mt.e`.

Pour connaître toutes les commandes contenues dans le module `cmath`, on entre l'instruction `dir("cmath")`.

---

3. Les notebooks donnent une visualisation facile et une bonne présentation avec différents effets de style. Ils sont bien adaptés à l'utilisation en intelligence artificielle. Cependant, ils ne garantissent pas l'ordre d'exécution du code et rendent délicats la définition de modules et le lien avec d'autres outils : ils ne sont donc pas appropriés au développement de logiciels.

### 1.1.4 Utilisation de fichiers

Dans le menu de IDLE, sélectionner "File", puis "New File". Une nouvelle fenêtre s'ouvre, dans laquelle on peut entrer des instructions, avec des retours à la ligne sans que la ligne précédente soit exécutée. Toutes ces instructions peuvent cependant être exécutées en bloc en utilisant la touche "f5".

✓ Dans la pratique, on crée un fichier<sup>4</sup> dès que l'on souhaite écrire une procédure ou tracer un graphe. Le terminal utilisé jusqu'à présent sert alors à évaluer certaines fonctions du fichier pour des valeurs données des paramètres.

## 1.2 Types d'objets

### 1.2.1 Entiers et flottants

Différents types d'objets existent en Python : entiers, flottants, chaînes de caractères... La commande `type` donne le type d'un objet. Entrer `type(-44)` et `type(-44.0)`.

On peut convertir l'entier `-44` en un nombre flottant en écrivant `float(-44)`, de même que l'on peut convertir le flottant `-44.1` en entier (i. e. calculer  $\text{sgn}(x)E(|x|)$ ) en écrivant `int(-44.1)`. La partie entière est donnée par la commande `floor` du module `math`. Quant à la commande `round`, elle donne un arrondi, et l'on peut préciser le nombre de décimales souhaité. Exemple : `round(44.7318, 2)`.

### 1.2.2 Variables

Une **variable** est une *référence*<sup>5</sup> à une "case mémoire" dans laquelle on peut mettre un objet (nombre, polynôme, fonction, matrice, graphique...).

Par exemple, pour créer une variable `a` qui fait référence à l'objet `13` (on dit que l'on affecte la valeur `13` à la variable `a`), on écrit `a=13`. Évaluer alors `a` et `a2`.

On peut créer plusieurs variables simultanément. Par exemple `b, c=4, 7`. L'affectation des variables se fait en parallèle, ce qui évite le recours à une troisième variable quand on veut échanger leurs valeurs : les instructions `b, c=c, b` échangent le contenu des variables `b` et `c`. Attention cependant au cas des tableaux 2d dont on reparlera au

4. Le module `cxFreeze` transforme un fichier Python en exécutable : pour un fichier dont le nom est `bonjour.py`, on le place dans un dossier, appelé par exemple `Compilation`, situé sur le bureau. On crée dans ce dossier le fichier `setup.py` qui contient les trois lignes suivantes.

```
from cxFreeze import setup, Executable
setup(name='Salutation', version='1.0', description='Programme pour saluer',
      executables=[Executable('bonjour.py')])
```

Ensuite, on lance un terminal et l'on obtient une ligne semblable à `C:\Users\Fred>`.

Pour se placer dans le dossier `Compilation`, on entre l'instruction `cd Desktop\Compilation`.

On obtient alors la ligne `C:\Users\Fred\Desktop\Compilation>`.

On entre enfin la commande `python setup.py build` qui construit le fichier exécutable.

Un dossier `build` est alors créé dans le dossier `Compilation`, contenant un dossier qui à son tour comporte plusieurs fichiers dont le fichier exécutable `bonjour.exe`.

5. Et non directement une "case mémoire"! Cette distinction a toute son importance.

paragraphe 1.4.4. . .

Il est aussi possible d'affecter une même valeur à plusieurs variables en même temps : `a=b=c=4`. Observer également l'effet de l'instruction `a, *b=1, 2, 3, 4`.

Notons enfin que l'objet `None` exprime l'absence de valeur.

On peut ajouter une valeur à une variable donnée (c'est l'incrémement). L'instruction `a=a+4` peut aussi s'écrire `a+=4`; on a de même les opérateurs `-=`, `*=`, `/=`, `//=` et `%=`.

Pour connaître la place occupée en mémoire par une variable, on dispose de l'instruction `getsizeof`<sup>6</sup>.

```
import sys
a=range(10000000); sys.getsizeof(a) → 48
b=[k for k in a]; sys.getsizeof(b) → 81528056
```

### 1.2.3 Nombres complexes

Le nombre complexe  $i$  est représenté en Python par la constante `1j`. Pour définir le nombre  $z = 4 - 7i$ , on écrira `z=4-7j`. On accède alors aux parties réelle et imaginaire de  $z$  par les instructions `z.real` et `z.imag`. Quant au module de  $z$ , il est donné par `abs(z)`.

Pour utiliser certaines commandes relatives aux nombres complexes, on doit importer le module `cmath`. Après avoir écrit `import cmath`, on peut calculer l'argument de  $z$  (`cmath.phase(z)`) et son écriture exponentielle (`cmath.polar(z)`). Inversement, l'écriture cartésienne d'un nombre complexe  $z = \rho e^{i\theta}$  s'obtient par la commande `rect`. Par exemple pour  $e^{i\frac{\pi}{4}}$ , on écrit `cmath.rect(1, cmath.pi/4)`.

### 1.2.4 Booléens

Les **constantes booléennes** sont `True` et `False`. Par exemple, si l'on veut savoir si l'inégalité  $4 < 13$  est vraie, on entre `4<13` et Python renvoie `True`. On teste l'égalité (resp. la non égalité) de deux valeurs au moyen de l'opérateur `==` (resp. `!=`).

La conjonction (resp. la disjonction) est donnée par `and` (resp. `or`).

Quant à la négation, elle s'écrit `not`.

On convertit un booléen en un entier en écrivant par exemple `int(True)`.

Tester ces différentes commandes à partir des formules  $4 < 13$ ,  $4 > 13$ ,  $4 = 4$ ,  $4 \neq 4$ , en créant des variables qui les contiennent.

### 1.2.5 Chaînes de caractères

Une **chaîne de caractères** est une suite de caractères placée entre guillemets ou entre apostrophes. Par exemple `x='vacances'`. On peut concaténer des chaînes de caractères grâce à l'opérateur `+`. On écrit ainsi `'vive les '+x`.

6. Le résultat est exprimé en **bytes**, un byte étant la plus petite unité de mémoire allouable, et en général un byte vaut un octet.

Calculer aussi  $7 * x$ .

Pour concaténer des chaînes  $c_1, c_2, \dots, c_n$  on écrit  $c = ' '.join([c_1, c_2, \dots, c_n])$ .

La commande `str` convertit un entier ou un flottant en une chaîne de caractères, ce qui est utile pour les concaténations. Exemple : `' numéro ' + str(4)`. Réciproquement, on a les commandes `int` et `float` : `entrer int('44')` et `float('44.7')`.

La commande `%s` permet de placer une valeur dans une chaîne de caractères : par exemple, l'instruction `'%s éléphants et %s pandas'%(4,7)` renvoie la chaîne de caractères `'4 éléphants et 7 pandas'`.

### 1.2.6 Listes

Une **liste** en Python est une suite finie d'objets séparés par des virgules qui est placée entre crochets. Un exemple de liste est `L=[4, -13.7, 'vacances', True]`. Une liste est en fait un tableau (au sens où l'on peut accéder facilement à n'importe quelle case et en temps constant, indépendamment de la longueur de la liste) dynamique (au sens où l'on peut ajouter et retirer des éléments et modifier leur valeur).

Attention, toute liste est indexée à partir de 0 (et non 1). La liste vide est notée `[]`. La commande `len` retourne le nombre d'éléments d'une liste, et `L[j]` le  $j$ -ème élément de la liste  $L$ .

On peut ajouter un élément à la fin d'une liste grâce à la méthode `append`.

Exemple : `L.append(92160)`.

Pour concaténer deux listes, on utilise l'opérateur `+`. Exemple : `L+[2, False, 'ab']`.

On peut également construire des listes avec l'opérateur `*`. Exemples : `4*L` (i. e. "4 fois la liste  $L$ "), `7*[0]`.

L'instruction `L[i:j]` renvoie la sous-liste de  $L$  formée des éléments d'indices compris dans l'intervalle  $[[i, j[$ . Exemple : `L[0:3]`.

Si l'on omet l'indice  $i$  (resp.  $j$ ), Python le remplace automatiquement par 0 (resp. la taille de la liste). Exemples : `L[:3]`, `L[1:]`.

On peut aussi accéder aux éléments d'une liste en utilisant des indices négatifs. Ainsi `L[-1]` renvoie le dernier élément de la liste.

La commande `del` supprime un élément d'une liste. Exemple : `del(L[2])`. Et la méthode `pop` supprime le dernier élément d'une liste et le récupère : `z=L.pop()`.

Attention, les variables sont des *références* : si l'on définit `x=[4, 7, 13, 0]` et `y=x`, alors la modification `x[1]=44` affectera aussi `y` ("car  $x$  et  $y$  pointent vers la même case mémoire"). En revanche si l'on définit `x=[4, 7, 13, 0]` et `y=[4, 7, 13, 0]`, alors la modification `x[1]=44` n'affectera pas `y` ("car  $x$  et  $y$  pointent vers deux cases mémoire différentes").

Etant donné une liste  $L$  comportant un élément  $a$ , l'instruction `L.index(a)` donne l'indice correspondant à la première occurrence de  $a$  dans  $L$ .

Quant à l'instruction `L.reverse()`, elle modifie la liste  $L$  en renversant l'ordre de ses éléments. On peut aussi créer une copie inversée de  $L$  en écrivant `M=L[::-1]`.

L'instruction `list(range(0,4))` renvoie la liste des entiers de l'intervalle  $[[0, 3[$ .

Autres exemples : `list(range(0,14,3)), list(range(6,1,-1))`.

Une méthode très importante pour définir une liste est donnée par l'exemple `[1/j for j in range(1,10)]`. On peut aussi spécifier une condition sur  $j$  : `[1/j for j in range(1,10) if j%2==0]`.

La commande `sum` permet de calculer des sommes finies. Ainsi on obtient la somme  $\sum_{k=1}^{10} k$  au moyen de `sum([k for k in range(1,11)])`. De même,  $\sum_{k=1}^{100} \frac{1}{k^2}$  s'obtient par `sum([1/k**2 for k in range(1,101)])`.

La commande `L.count(a)` donne le nombre d'occurrences de  $a$  dans  $L$ . On peut alors en déduire l'élément le plus fréquent dans  $L$ .

`L=[2,1,3,4,3,4,1,1,4,4,5,4,4,3,2,5]; max(L,key=L.count) → 4`

On peut également utiliser la fonction `Counter`.

`from collections import Counter`

`Counter(L) → Counter({4: 6, 1: 3, 3: 3, 2: 2, 5: 2})`

Il est utile, en particulier pour des données provenant de fichiers, de pouvoir transformer une chaîne de caractères en une liste. La commande `c.split('séparateur')` effectue cette opération.

`c='1.44,j,7.32,m'; L=c.split(',') → ['1.44', 'j', '7.32', 'm']`

## 1.2.7 Polynômes

Pour effectuer des calculs sur les polynômes, on importe le module `numpy`, dont nous reparlerons dans le paragraphe 1.4.1, en écrivant `import numpy as np`.

La commande `poly1d` permet de créer un polynôme en définissant ses coefficients dans l'ordre des degrés décroissants. Exemple : `P=np.poly1d([4,7,-1,2])`. On peut alors afficher ce polynôme en écrivant `print(P)`. Les opérations sur les polynômes sont obtenues grâce aux opérateurs classiques `+`, `-`, `*`. Le degré d'un polynôme est donné par la commande `order`. Exemple : `P.order`. Et l'on obtient la liste des coefficients de  $P$  (toujours dans l'ordre des degrés décroissants) en écrivant `P.c`. Pour avoir le coefficient de degré  $j$ , on écrit `P[j]`.

On obtient le polynôme dérivé (resp. primitive s'annulant en 0) grâce à `P.deriv()` (resp. `P.integ()`). Pour évaluer le polynôme en un point  $a$ , il suffit d'écrire `P(a)`. Pour calculer le quotient et le reste de la division euclidienne d'un polynôme  $A$  par un polynôme  $B$ , on procède ainsi.

`A=np.poly1d([4,7,-1,2]); B=np.poly1d([3,-5])`

`Q,R=A/B → poly1d([ 1.33333333, 4.55555556, 7.25925926]),  
poly1d([ 38.2962963])`

## 1.3 Graphes de fonctions et courbes paramétrées

### 1.3.1 Le module `matplotlib`

Pour tracer des graphiques avec Python, on doit importer le sous-module `pyplot` du module `matplotlib`. Notons aussi que le module `numpy` est souvent nécessaire. On commencera donc le fichier contenant les instructions graphiques par les deux lignes ci-dessous.

```
import numpy as np
import matplotlib.pyplot as plt
```

### 1.3.2 Graphes de fonctions

- Pour tracer le graphe de la fonction  $f$  sur l'intervalle  $[a, b]$ , on discrétise cet intervalle en une suite  $t_0, \dots, t_n$  d'abscisses. On trace alors la ligne brisée joignant les points de coordonnées  $(t_j, f(t_j))$ . Puis la commande `show()` (toujours placée à la fin) affiche le graphique. Notons que la commande `clf()` efface la fenêtre graphique. Pour tracer la fonction  $\cos$  sur  $[0, 10]$ , on écrit donc

```
t=np.linspace(0,10,100) (ici on choisit 100 points de discrétisation)
y=np.cos(t)
plt.plot(t,y); plt.show()
```

- De nombreuses options peuvent être ajoutées. Pour avoir un quadrillage, on écrit `grid()`. La commande `axis` spécifie une fenêtre d'affichage : on obtient la fenêtre  $[0, 2\pi] \times [-1.5, 1.5]$  en écrivant `plt.axis([0, 2*np.pi, -1.5, 1.5])`. On peut indiquer un troisième argument dans `plot` pour préciser un style de tracé ou une couleur. On peut aussi choisir l'épaisseur du trait grâce à la commande `linewidth`. L'instruction `plt.axis('equal')` permet d'avoir un repère orthonormé. On peut aussi indiquer des informations sur les axes.

Exemple : `plt.xlabel('temps'); plt.ylabel('signal')`.

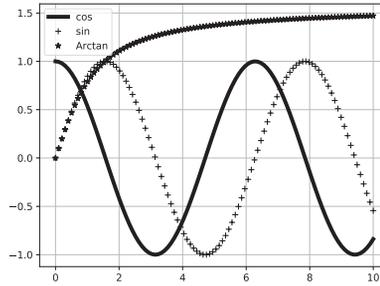
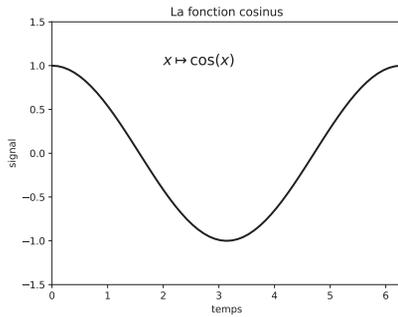
L'instruction `plt.title('La fonction cosinus')` ajoute un titre au graphe.

Il est aussi possible d'écrire du texte sur le graphique, éventuellement en Latex :

```
plt.text(2,1,r'$x \mapsto \cos(x)$', fontsize=15, color='b')
```

- Pour superposer plusieurs graphes, on procède comme ci-après. La légende (facultative) `plt.legend(('cos', 'sin', 'Arctan'))` (dans l'ordre indiqué pour le tracé) permet de les repérer.

```
t=np.linspace(0,10,100)
y=np.cos(t); z=np.sin(t); w=np.arctan(t)
plt.plot(t,y,'r',t,z,'k',t,w,'*g',linewidth=4)
plt.legend(('cos', 'sin', 'Arctan')); plt.grid(); plt.show()
```



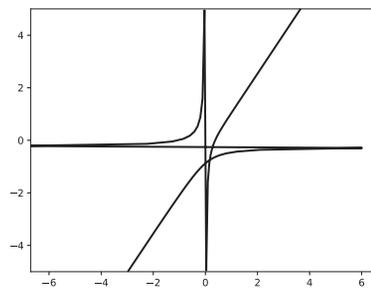
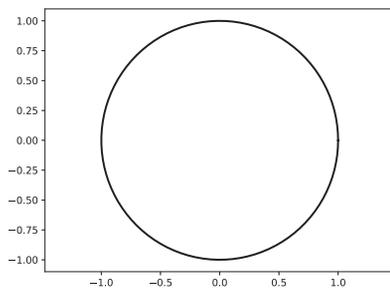
### 1.3.3 Courbes paramétrées

Pour tracer la courbe paramétrée  $t \mapsto (x(t), y(t))$ , on utilise toujours la commande `plot`. Exemple :

```
t=np.linspace(0,2*np.pi,100)
x=np.cos(t); y=np.sin(t)
plt.plot(x,y,'r'); plt.axis('equal'); plt.show()
```

De même, pour tracer la courbe définie par  $t \mapsto \left( \frac{2t + t^2}{3 + 3t}, \frac{2t^2 - 1}{4t} \right)$ , on écrit les instructions suivantes.

```
t=np.linspace(-10,10,200)
x=(2*t+t**2)/(3+3*t); y=(2*t**2-1)/(4*t)
plt.plot(x,y,'r'); plt.axis('equal');
plt.axis([-5,5,-5,5]); plt.show()
```



## 1.4 Algèbre linéaire

### 1.4.1 Le module `numpy`

Pour effectuer des calculs sur les polynômes, on importe le module `numpy` : on le fait en général en écrivant `import numpy as np`. Une fois `numpy` importé, il est