
TP n° 0

INTRODUCTION À PYTHON

Pour commencer - Utilisation d'un Notebook

Python est un langage de programmation interprété haut niveau. De nombreux packages sont disponibles pour enrichir ses fonctionnalités.
Nous allons utiliser *le Notebook Jupyter* installé sur les machines de l'université.

1. Entrer "Notebook" dans la barre de recherche.
2. Cliquer sur "Notebook Jupyter", ce qui va ouvrir une page web dans le navigateur par défaut.
3. Une fois la page ouverte, cliquer sur *Nouveau –> Python 3*
4. Renommer votre fichier, par exemple « Prise en main ».

Notebook

Utiliser cet outil permet de combiner texte et code. La mise en forme se fait à l'aide de *cellules*.
Chaque cellule s'exécute avec la commande ▶ Exécuter, ou en utilisant "Maj+Entrée".

5. Exécuter la commande `print('Hello')` dans une cellule.

La sauvegarde se fait sous la forme d'un fichier avec l'extension `.ipynb`, on peut aussi exporter sous forme de script Python (extension `.py`), ou de fichier `.pdf` par exemple.
Pour utiliser l'aide, par exemple pour la fonction print) on peut écrire `help(print)` ou `? print` dans une cellule, puis l'exécuter.

6. Exécuter la commande `help(round)`, puis `? range`.

Conseils généraux

— Ne pas hésiter à consulter l'aide (ou l'aide en ligne).
— Bien respecter la syntaxe, la casse (majuscules/minuscules), et les indentations (en particulier pour les boucles et les fonctions).

Exercice 2 - Un peu de Python

Effectuer des opérations :

```
2+5
2345/34578

2**3
5**(1/3)
```

Définition de variables :

```
a = 10
b = a + 5
b

a, b, c = 1, 2, 3
print(a, b, c)

a, b = b, a
print(a, b)

s = "Hello !"
print(s)
```

Les listes : il y aurait beaucoup à dire dessus, voici quelques opérations que l'on sera amené à utiliser plus tard.

```
c = [1, -2, 7, 0, 10]
d = [3, 4]
c + d
2 * c

10 * c[0]

print(c[0], c[1])
print(c[-1], c[-2])

# créer l'intervalle des entiers entre
# 0 et 9
e = range(10)
print(e)

# pour en créer la liste
e = list(e)
print(e)

e = range(5, 11)
print(list(e))

e = [i**2 for i in range(11)]
print(e)
```

Une fonction Python se déclare comme suit :

```
def nom_de_fonction(input1, input2, ...):
   instructions
   return output1, output2, ...
```

où les `input` sont les arguments d'entrée de la fonction et les `output` les argument de sortie. Ni les uns ni les autres ne sont obligatoires.

Attention ! Les indentations sont très importantes en Python. C'est ce qui définit si les instructions font partie de la fonction ou du script. En effet, il n'y a rien qui indique la fin de la fonction à part les indentations !

Par exemple, pour coder la fonction $f : x \mapsto \frac{1}{(1+x)^2} - x$, on écrit les lignes suivantes

```
def f(x):
    return 1/(x+1)**2 - x

print(f(7))
```

Nous allons maintenant voir comment effectuer des tests et des boucles.

- Les tests avec le mot-clé `if`. La syntaxe est la suivante :

```
if une_condition:
  instructions
elif une_autre_condition:
  instructions
else:
  instructions
```

Remarques :

— Les symboles : après le `if`, `elif` et `else` sont obligatoires. Ils marquent le début des blocs.

— Les indentations sont importantes ! Elles définissent si les instructions font partie des blocs ou non. Il n'y a, en effet, pas de mot-clé pour marquer la fin des blocs en Python.

— Les mot-clés `elif` et `else` ne sont pas obligatoires.

Par exemple,

```
def f(x):
    if x < 0:
        return -x
    elif x == 0 or x == 1:
        return 1-x
    else:
        return x**2
```

▶ Tester `f` sur différentes valeurs. Par exemple -5, 1, 3, 1.5.

- Les boucles `for`. La syntaxe est :

```
for variable_boucle in une_variable:
    instructions
```

Remarques :

— la variable `variable_boucle` va, une par une, prendre les valeurs dans la variable `une_variable`. Pour chacune de ces valeurs, les instructions dans le bloc `for` seront exécutées.

— `une_variable` peut être, par exemple, une liste Python ou un tableau (on le verra plus tard).

— Comme pour les `if`, les `:` et les tabulations sont nécessaires.

Par exemple :

```
for i in range(5):
    print(i*i)
```

- Les boucles `while`. La syntaxe est :

```
while condition:
    instructions
```

Les instructions dans le bloc `while` sont exécutées tant que `condition` est vraie. Exemple, pour construire la suite définie pour tout :

```
a, b = 0, 1
while b < 1000:
    a, b = b, a + b
    print(round(b/a, 3), end=", ")
```

Exercice 3 - Le calcul avec Numpy et les tableaux

Le package **Numpy** est **LE package de référence** pour le calcul numérique en Python. Il doit être importé avec la commande suivante

```
import numpy as np
```

Fonctions usuelles

Ces fonctions existent dans plusieurs packages, dont le package Numpy qui est celui que nous utiliserons en permanence.

```
np.abs(-2)
np.sqrt(3)/2
np.exp(2)
np.log(np.e)
np.sin(np.pi/6)
np.sin(np.pi/3)
```

Ces fonctions marchent aussi sur des tableaux en opérant élément par élément.

```
x = np.linspace(0,np.pi,7)
y = np.sin(x)
print(np.round(y,3))
```

Premiers exemples de tableaux

```
# Les listes Python
c = [1, -2, 7, 0, 10]
2 * c

# Avec un tableau Numpy
c = np.array([1, -2, 7, 0, 10])
2 * c
print(2 * c)

c.size
np.size(c)

c[3]
c[0] + c[1] + c[2] + c[3] + c[4]
c[0] = 0
print(c)

d = np.array([[0, 0, 0], [0, 0, 0]])
print(d)

d = np.zeros((2, 3))
print(d)
d.shape
np.shape(d)

a = np.array([[0, 1], [0, 0]])
print(a)
a[1, 1] = 1
a[0, 0] = a[1, 1]
print(a)
```

Création automatique de tableaux structurés

Bien observer le résultat de chaque commande pour comprendre son fonctionnement, en allant éventuellement regarder dans l'aide.

```
print(np.arange(0,10))
print(np.arange(0,10,0.5))

print(np.linspace(0, 10, 10))
print(np.linspace(0, 10, 11))
print(np.linspace(0, 10, 21))
print(np.linspace(0,2*np.pi,7))

x = np.array([2,-1,np.pi,-3,-2,-np.e])
print(x)
print(np.max(x))
print(np.min(x))

x = np.arange(0,20)
print(x)
print(x.size)

print(x[2:17])
print(x[2])
print(x[17])

print(x[:-2])
print(x[3:])
print(x[9:-8])
```

Les dernières commandes montrent qu'il est possible d'extraire un sous-tableau d'un tableau plus grand. Pour cela, il suffit de taper : `tableau[début:fin+1]`.

Pour extraire par exemple une valeur sur deux, on peut

utiliser la commande `tableau[début:fin+1:2]`.
▶ Tester l'extraction dans un sous-tableau de tous les indices pairs du tableau `x`.

Il est également possible de créer un tableau à partir d'une formule et d'une boucle for

```
x = [i**2 for i in range(10)]
print(x)

y = [np.cos(2*k*np.pi/6) for k in range(7)]
print(y)
```

Attention à la copie de tableaux :

```
y=x
y[0]=-1
print(x)

z=y.copy()
z[0]=0
print(y)
```

Exercice 4 - Tracer des graphiques

Il existe beaucoup de packages en Python pour tracer des graphiques. Le plus utilisé est `matplotlib`. Pour charger les fonctions principales de ce package, on utilise la commande suivante :

```
import matplotlib.pyplot as plt
```

Nuage de points et points reliés
Pour représenter avec Python un nuage de points $\{(x_k, y_k) : 1 \leq k \leq N\}$, on définit un vecteur contenant les abscisses $(x_1, \ldots, x_N)$ et un vecteur contenant les ordonnées $(y_1, \ldots, y_N)$ des points dans le même ordre. On ajoute ensuite le marqueur souhaité pour matérialiser les points (une croix `+`) dans la couleur souhaitée (`r` pour 'red').

```
x = [np.cos(2*i*np.pi/5) for i in range(6)]
y = [np.sin(2*i*np.pi/5) for i in range(6)]
plt.plot(x, y, 'r+')
```

Pour relier les points, il suffit d'ajouter le type de trait par lequel on souhaite que les points soient reliés (`-` pour un trait continu).

```
x = [np.cos(2*i*np.pi/5) for i in range(6)]
y = [np.sin(2*i*np.pi/5) for i in range(6)]
plt.plot(x, y, 'r+-')
```

D'autres options permettent de spécifier la couleur de la courbe (`m` magenta), la représentation des coordonnées (`*` étoile) et le type du trait (`--` discontinu).

```
x = [np.cos(2*i*np.pi/5) for i in range(6)]
y = [np.sin(2*i*np.pi/5) for i in range(6)]
plt.plot(x, y, 'm*--')
```

Représentation des termes d'une suite
Pour visualiser les N premiers termes d'une suite $(u_n)_{n\in\mathbb{N}}$, on peut dessiner le nuage de points $\{(n, u_n) : 0 \leq n \leq N\}$ (réliés ou non). Pour cela, on commence par définir le vecteur des abscisses $(0, \ldots, N)$ puis le vecteur des ordonnées $(u_0, \ldots, u_N)$. On peut finalement tracer la suite en précisant que les marqueurs sont par exemple des ronds (`o`) bleus (`b`).
Dans l'exemple qui suit, la suite $(u_n)_{n\in\mathbb{N}}$ est définie par $u_n = n^2 + 1$ pour tout $n \in \mathbb{N}$.

```
n = np.arange(25)

def u(n):
  return n**2+1

plt.plot(n,u(n), 'bo')
```

Tracé de la courbe représentative d'une fonction
Pour tracer le graphe d'une fonction $f : D_f \to \mathbb{R}$, on choisit d'abord un segment $[a, b]$ inclus dans D_f, où vont être prises les abscisses. On ne peut pas demander à Python de calculer une infinité de valeurs, on va donc créer un vecteur d'abscisses contenant "beaucoup" de points dans $[a, b]$, $(x_0, \ldots, x_N)$ avec $a = x_0 < x_1 < \ldots < x_N = b$ (par exemple une centaine). Ensuite, on va créer le vecteur des ordonnées correspondantes $(f(x_0), \ldots, f(x_N))$. Enfin, on va tracer les points reliés $\{(x_k, f(x_k)) : 0 \leq k \leq N\}$. Dans ce cas, on ne souhaite pas voir les points matérialisés, on indique donc juste le fait que l'on souhaite un trait continu reliant les points $(x_k, f(x_k))$.

```
x = np.linspace(0, 2*np.pi, 100)
plt.plot(x, np.sin(x), '-')
```

▶ Changer dans le code ci-dessus la valeur 100 par 10. Qu'observe-t-on ?
▶ Par défaut, Python ne met pas de marqueur et relie les points. Essayer la commande `plt.plot(x, np.sin(x))`.

Pour tracer plusieurs graphes sur une même figure, on donne les commandes de tracé à la suite.

```
x = np.linspace(0, 2*np.pi, 100)
plt.plot(x, np.cos(x))
plt.plot(x, np.sin(x))
```

Titres et légendes
Toute figure doit être accompagnée d'une légende, et d'un titre pour chacun des axes et pour la figure.

```
plt.plot([0, 1], [1, 0], 'b', label='Segment')
plt.plot(0.1, 0.9, 'r+', label='Point')
plt.xlabel('x')
plt.ylabel('y')
plt.title('joli graphe')
plt.legend()
```