
TP : 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. **Pour ces TP, il n'y a pas d'installation à effectuer (que ce soit sur les machines des salles de TP ou sur votre propre machine)** [1].

Nous allons utiliser la plateforme *Jupyter* du département de mécanique de Lyon 1 et le *notebook* associé.

`https://jupyter.mecanique.univ-lyon1.fr/`

1. Rendez vous sur cette plateforme.
2. Connectez vous avec vos login et mot de passe de l'université.
3. Cliquez 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 espaces (en particulier pour les boucles et les fonctions).

Pour la suite de ce TP vous utiliserez le fichier fourni via le lien ci-dessous. Pour cela :

7. *Fichier –> Fermer et arrêter* ce premier *Notebook* « Prise en main ».
8. Télécharger le fichier `http://math.univ-lyon1.fr/~blossier/Analyse3/TP0.ipynb`.
9. *Téléverser* sur la plateforme *Jupyter* le fichier téléchargé puis ouvrir le *Notebook* correspondant.

Pour tous les exercices, écrire les commandes dans des les parties adéquates du *Notebook* fourni et observer les réponses afin d'identifier l'utilisation de chacune des commandes.

Pour les questions mathématiques de l'exercice 3, rédiger vos réponses sur une feuille indépendante en reportant les numéros des questions.

Exercice 1 - Un peu de Python

Effectuer des opérations :

1. Si vous souhaitez installer Python sur votre propre machine, une implémentation peut être téléchargée avec Miniconda3 par exemple `https://conda.io/miniconda.html`. Une fois Python installé, il y a plusieurs possibilités :
   — utilisation d'un *Notebook*,
   — utilisation d'un *IDE* (environnement de développement), par exemple *Pyzo* ou *Spyder*.

```
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 :

```
def f(x):
    x=x+1
    x=x*x
    return x
f(7)
```

Les fonctions mathématiques usuelles sont accessibles dans le package `math` :

```
sqrt(2)
import math
math.sqrt(2), math.pi, math.log(5)
```

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(n):
    if n <0:
        return(-n)
        print('Nombre strictement négatif')
    elif n == 0 or n == 1:
        n=1-n
        print('Zéro ou Un')
    else:
        n=math.factorial(n)
        print('Strictement plus grand que Un')
    return(n)
```

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

À votre tour ! Coder la fonction

$$g : \begin{array}{l} \mathbb{R} \to \mathbb{R} \\ x \mapsto \begin{cases} x \sin x & \text{si } x < 0 \\ \ln x & \text{si } x > 0 \\ 0 & \text{si } x = 0 \end{cases} \end{array}$$

— 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 (suite de Fibonacci) :

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

Exercice 2 - 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 `import numpy`.

Premiers exemples de tableaux :

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

# Avec un tableau Numpy
import numpy as np
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)
```

D'autres opérations sur les tableaux numpy. Bien observer le résultat de chaque commande pour comprendre son fonctionnement.

```
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]`.

Attention à la copie de tableaux :

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

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

Fonctions usuelles

Ces fonctions existent à la fois dans le package math et dans le package Numpy. Comme on utilisera essentiellement des tableaux Numpy, il est préférable d'utiliser les versions Numpy.

```
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))
```

Exercice 3 - Approximations du maximum d'une fonction de classe C^1 sur un segment $[a,b]$

Définir une fonction `max_approx` prenant en argument une fonction réelle f, les extrémités a et b et un entier $n \geq 1$, et retournant la valeur

$$\max_{0\leq k\leq n} f\left(a+k\frac{(b-a)}{n}\right).$$

```
def max_approx(f,a,b,n):
    # compléter - utiliser la commande linspace
    return # compléter
```

Test avec la fonction sinus sur $[0,2]$:

```
print([max_approx(np.sin,0,2,10**i) for i in range(2,7)])
```

1) Pourquoi la fonction sinus admet-elle un maximum sur $[0,2]$?

2) Que vaut-il ? Où est-il atteint ?

Fixons un entier $n \geq 1$. Notons p l'unique entier tel que $\pi/2 \in [2p/n, 2(p+1)/n[$.

3) Montrer que

$$\max_{0\leq k\leq n} \sin\left(\frac{2k}{n}\right) = \max\left\{\sin\left(\frac{2p}{n}\right), \sin\left(\frac{2(p+1)}{n}\right)\right\}.$$

4) Expliquer pourquoi $\pi/2$ ne peut être le milieu du segment $[2p/n, 2(p+1)/n]$. On notera x_n l'extrémité la plus proche de $\pi/2$.

5) En utilisant le fait que : $\forall x \in \mathbb{R}$, $\sin\left(\frac{\pi}{2}+x\right) = \sin\left(\frac{\pi}{2}-x\right)$, montrer que $\max_{0\leq k\leq n} \sin\left(\frac{2k}{n}\right)$ est atteint en x_n.

6) Montrer enfin que

$$\left|1 - \max_{0\leq k\leq n} \sin\left(\frac{2k}{n}\right)\right| \leq \frac{1}{n}.$$

Observer ce que retourne la commande suivante :

```
print([(1-max_approx(np.sin,0,2,10**i))*(10**i) for i in range(2,7)])
```

7) Montrer que $\dfrac{\sin(x_n)-1}{x_n - \frac{\pi}{2}} \xrightarrow[n\to\infty]{} 0$.

8) En déduire que $n\left(1 - \max_{0\leq k\leq n} \sin\left(\frac{2k}{n}\right)\right) \xrightarrow[n\to\infty]{} 0$.

Coder la fonction $f : \mathbb{R} \to \mathbb{R}$, $x \mapsto |x(x-1)|$.

9) Qu'observe-t-on en utilisant la fonction `max_approx` sur f entre $[0,1]$ selon la parité de n ?

10) Déterminer théoriquement le maximum de f sur $[0,1]$.

Coder la fonction $g : \mathbb{R} \to \mathbb{R}$, $x \mapsto |(x+1)x(x-1)|$.

11) Déterminer théoriquement le maximum de g sur $[-1,1]$, puis vérifier votre résultat à l'aide de la fonction `max_approx`.

Exercice 4 - Tracer des graphiques

Il existe beaucoup de packages en Python pour tracer des graphiques. Le plus utilisé est `matplotlib`.

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

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

```
plt.plot(np.cos(x), np.sin(x))
plt.show()

plt.plot(x, np.cos(x), x, np.sin(x))
plt.show()

plt.plot(x, np.cos(x))
plt.plot(x, np.sin(x))
plt.show()
```

Les graphiques sont tracés uniquement lorsque la commande `show()` est lancée.

Changer dans le code ci-dessus la valeur 100 par 10. Qu'observe-t-on ?

RÉPONSE :

Pour afficher plusieurs fenêtres :

```
x = np.linspace(0, 2*np.pi, 100)
y = np.linspace(0., 1., 100)

plt.figure(1)
plt.plot(x, np.sin(x))

plt.figure(2)
plt.plot(y, y * np.sin(y))

plt.show()
```

Premières options de `plot.` :

```
x = np.linspace(0, 4 * np.pi, 30)
plt.plot(x, np.sin(x), 'r*--')
plt.show()
```

Le troisième argument est une option permettant de spécifier la couleur de la courbe (`r` red), la représentation des coordonnées (`*` étoile) et le type du trait (`--` discontinu). Voir l'aide `? plt.plot` pour avoir la liste des options possibles.

On peut également mettre une légende, donner des titres aux axes et à la figure.

```
plt.plot([0, 1], [1, 0], 'b')
plt.plot(0.1, 0.9, 'r+')
plt.xlabel('x')
plt.ylabel('y')
plt.title('joli graphe')
plt.legend(['segment', 'point'])
plt.show()
```

La commande `fill_between` :

```
x =np.linspace(0,7,100)
plt.fill_between(x,np.cos(x),0)
plt.show()

x =np.linspace(0,7,10)
plt.fill_between(x,np.cos(x),0)
plt.show()
```

```
plt.fill_between(x,np.cos(x),0,step='pre')
plt.show()
```

Exercice 5 - Représentations graphiques de méthodes d'intégration numérique - FACULTATIF

Tester, puis analyser le code suivant

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

def rep_int(f,a,b,n,methode):
    x = np.linspace(a,b, 100)
    plt.plot(x,f(x),color='red')
    x,h = np.linspace(a,b, n+1, retstep=True)
    if methode=='rectgauche':
        y=f(x)
        titre='à gauche'
    elif methode=='rectdroite':
        y=f(x+h)
        titre='à droite'
    plt.fill_between(x,y,0,step='post',facecolor='green',alpha=0.5)
    plt.vlines(x[:-1],0,y[:-1])
    plt.vlines(x[1:],0,y[:-1])
    plt.hlines(y[:-1],x[:-1],x[1:])
    plt.hlines(0,a-h/2,b+h/2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('méthode des rectangles '+titre)
    plt.legend(['f','aire'])
    plt.show()

rep_int(np.cos,0,3*np.pi/2,5,'rectgauche')

rep_int(np.cos,0,3*np.pi/2,5,'rectdroite')
```

On pourra lors de l'étude des méthodes en question (fiche suivante) :

- compléter la fonction `rep_int` pour représenter la méthode des rectangles au milieu ;
- adapter cette fonction, pour représenter la méthode des trapèzes.