TP0 Prise en main de Python

February 5, 2024

1 TP0: Prise en main de Python et Jupyter Notebook.

1.1 Petite introduction pour commencer

L'outil principal de ces TPs est **Jupyter Notebook**. **Jupyter Notebook** est un produit open source proposé par le projet Jupyter qui permet la création et le partage de documents constitués d'une liste ordonnée de cellules d'entrées et de sorties. Les cellules peuvent contenir du code, du texte au format Markdown, des formules mathématiques ou des contenus médias.

Jupyter Notebook est développé à l'origine pour les applications de data science écrites en Python (la distribution Anaconda de Python est par exemple livrée avec), mais aussi utilisé pour les applications en R et en Julia. Ce produit propose une facilité pour

- **la visualisation de données.** Jupyter permet de créer ces visualisations, de les partager et d'autoriser les changements interactifs.
- **le partage et les interactions en direct avec le code.** Les services cloud tels que GitHub ou GitLab permettent de partager du code, mais ils ne sont pas interactifs. Avec un cahier Jupyter, on peut voir le code, l'exécuter, le corriger et afficher les résultats en directe. Le code n'est pas statique dans Jupyter.

Pour utiliser Jupyter Notebook il faut d'abord installer, dans son système ou dans le Cloud, l'application client et serveur de l'environnement du code. La seule condition est qu'il soit également installé une version valide de Python. La distribution Anaconda (conseillée pour l'installation) contient aussi bien Jupyter Notebook et Python. Une fois l'installation réalisée, on peut lancer le serveur locale Notebook par la ligne de commande: **jupyter notebook** (en terminal) ou depuis Anaconda-Navigator.

À noter qu'en salle machine l'installation est deja realisée et vous pouvez ouvrir Jupyter Notebook à partir du menu principal.

Les fichiers jupyter notebook sont sauvegardées automatiquement localement sur votre machine, mais il ne serait pas de trop des les sauvegarder de temps en temps (Cntl+S). Ils sont stockées sur la machine localement, vous pouvez ensuite les copier ou vous les envoyer par email si besoin, ainsi que les telecharger sur moodle pour le **rendu de chaque TP**.

1.2 Objectifs de TP 0

1. Connaître les commandes et les bibliotheques de base de Python.
2. Se familiariser avec la syntaxe du langage Python
3. Savoir utiliser la condition if

4. Apprendre à utiliser la boucle for
5. Savoir definir les fonctions en Python
6. Apprendre à tracer les courbes

1.3 Pour vraiment commencer : Importation des bibliotheques de Python

Les bibliothèques **Numpy** et **Matplotlib** sont les plus souvent utilisées en Python. La bibliothèque **Numpy** utilise un large éventail de fonctions mathématiques qui le rendent particulièrement utile pour des calculs scientifiques et problèmes de mathématiques appliquées. Le module **Matplotlib** permet de tracer des fonctions et d'afficher leurs courbes dans des graphiques.

Le programme ci-dessous permet à importer ces bibliothèques en fin de pouvoir les utiliser

[ ]:
```
# importation des fonctions mathématiques, on peut maintenant utiliser les␣
↪fonction sin, exp, la valeur de pi, etc.
from math import *

# on importe la bibliothèque numpy qui permet de travailler avec des tableaux
import numpy as np

# on importe la module matplotlib.pyplot qui permet tracer des fonctions
import matplotlib.pyplot as plt
```

Donc pour chaque TP on commencera toujours par:

```
import numpy as np
from math import *
import matplotlib.pyplot as plt
```

et vous rajouterez d'autres bibliothèques si besoin.

1.3.1 Boucles for et Condition if. Definition d'une fonction.

L'exemple ci-dessous permet de calculer la sommes $1+2+\cdots+10$ en utilisant la boucle **for**.

On definit ensuite une fonction **y=somme(n)** qui calcule la sommes $1+2+\cdots+n$ et on compare avec le resultat de la question 1 pour $n=10$.

[ ]:
```
# on commence par donner les instructions permettant d'imprimer les commentaires
print("somme en utilisant boucle for\n")

s = 0
for i in range(1,11):
    s = s+i
print(s)


print("somme avec création de fonction Python\n")
def somme(n):
```

```
    s = 0
    for i in range(1,n+1):
        s = s+i
    return s

print("1+...+10 = ", somme(10))
```

On peut imprimer le résultat de comparison en utilisant la condition **if**

[ ]:
```
if s == somme(10):
    print('Nous avons obtenu les mêmes résultats!')
```

En utilisant l'exemple,

1. Calculer le produit $1 \times 2 \times \cdots \times 10$ en utilisant la boucle **for**.
2. Écrire une fonction function simple **factorielle(n)** qui calcule $n!$. Comparer avec le resultat de la question 1 pour $n = 10$. La syntaxe Python pour la définition d'une fonction est la suivante

```
def factorielle(n):
    return
```

On rappel que par convention on a $0! = 1$, en utilisant la condition **if** rajouter cette definition à votre fonction **factorielle(n)**.

Remarque. Vous pouvez utiliser la command **print** pour separer vos reponses aux questions, comme il est proposé ci-dessus.

[ ]:
```

```

1.3.2 Tableau dans la bibliothèque Numpy np.arange, np.linspace, np.array

Les commandes np.array, np.arange, np.linspace de la bibliothèque **Numpy** permettent à créer des tableaux. Voici des exemples de leur utilisation:

[ ]:
```
T = np.array([-7,1,2,5]) # permet de créer un tableau numpy à partir d'une␣
↪liste Python
print(T)
```

[ ]:
```
A = np.arange(1,11,1) # tableau de chiffres de 1 à 10 (11 est non inclus) par␣
↪incréments de 1
print(A)
```

[ ]:
```
L = np.linspace(-pi,pi,9) # tableau des nombres équidistants de -pi à pi␣
↪(compris) avec 9 éléments
print(L)
```

Les fonctions de la bibliothèque **numpy** permettent de travailler directement avec les tableaux, par exemple, on peut directement calculer la somme de tableau A sans utiliser la boucle **for** (et on

obtient le même résultat qu'avant)

[ ]:
```
np.sum(A)
```

Si vous souhaitez appliquer une fonction, par exemple **sin**, à chaque élément d'un tableau, vous pouvez bien sûr utiliser une boucle **for**, mais vous pouvez également utiliser **np.sin()**

Si vous essayez d'exécuter

[ ]:
```
sin(L)
```

Vous aller obtenir un message d'erreur!

```
TypeError: only size-1 arrays can be converted to Python scalars
```

La fonction **sin()** est définie dans la bibliothèque **math** et ne peut être utilisée qu'avec des scalaires. En revanche, la fonction **np.sin()** peut être utilisée pour calculer les sinus de chaque élément du tableau :

[ ]:
```
np.sin(L)
```

1.3.3 Vecteurs et Matrices: range(), np.arange, np.linspace, np.array

Exemple: Voici une boucle qui imprime les entiers de 0 à 13

```
for i in range(14):
    print(i)
```

Notons que le compteur i va de 0 à 13 par pas de 1 : il y a donc bien 14 répétitions.

Selon les besoins, on peut aussi d'abord declarer un tableau des entiers de 0 à 13 et ensuite imprimer ces elements:

```
A = np.arange(0,14,1)
for Ai in A:
    print(Ai, end =" ")
```

Voici une boucle qui imprime les entiers de -7 à 3 de deux façon

[ ]:
```
# sans la construction d'un tableau
for i in range(-7,4):
    print(i, end =" ") # Utilisation de end =" " permet d'imprimer en ligne
print("\n")


# avec la construction d'un tableau
A = np.arange(-7,4,1)
for Ai in A:
    print(Ai, end =" ")
```

Maintenant on va s'intéresser aux opérations sur les matrices et vecteurs. Soit $u = \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix}$, $v =$

$$\begin{pmatrix} 6 \\ 1 \\ 2 \end{pmatrix}, M = \begin{pmatrix} 6 & 3 & 3 & 5 \\ 1 & 2 & 1 & 7 \\ 2 & 0 & 4 & 5 \end{pmatrix}$$

À l'aide de commandes Python (afficher le résultat à l'aide de commande **print**)

A. Calculer et afficher $3u - 6v$.

B. Calculer et afficher la somme des éléments de u. **np.sum(u)**

C. Calculer et afficher le produit scalaire des vecteurs u et v definit par

$$< u, v >= \sum_{i=1}^{n} u_i v_i, \text{ ici } n = 3$$

On utilisera la commande **np.dot()**

D. Construire un vecteur u_1 partant de -8 et allant jusque -5 par pas de 0.25 (**np.arange()** ou **np.linspace()**). Quelle est sa longueur? (**len()**)

E. Construire un vecteur v_1 décroissant d'entiers allant de 5 à -7 (**np.arange()** ou **np.linspace()**). Quelle est la longueur de ce vecteur? Peut-on calculer $u_1 + v_1$?

[ ]:
```
# On definit les variables en utilisant np.array()
u = np.array([2,5,8])
v = np.array([6,1,2])
M = np.array([[6, 3, 3, 5], [1, 2, 1, 7], [2, 0, 4, 5]])
```

1.3.4 Fonctions et Graphiques, bibliotéque matplotlib: plt.plot(), plt.title(), plt.show()

Les instruction suivants permettant à definir une fonction f, calculer ses valeurs et tracer son graphe.

[ ]:
```
def f(x):
    return np.exp(2*x)

print('f(2) =', f(2))
print('f([1,3,4]) =', f(np.array([1, 3, 4])))

x = np.linspace(0,1,50)
plt.plot(x, f(x),'r')
```

Tracer les graphes des focntions g et h definies par

1. Soit $g : x \longmapsto \sqrt{3x^2 + 1} - \ln(x)$.
2. Soit $h : x \longmapsto x^2 + 1$ et $h : x \longmapsto 3\sin(x)$.

sur le même plot et sur deux plots differentes (differentes utilisation de la commande **plt.show()**)

[ ]: