TD n° 0 : Prise en main de Python

Installer Python

Si vous utilisez les ordinateurs de l'université, ignorez cette section et allez tout de suite à "Fondamentaux de Python". Cette section n'est utile que si vous souhaitez utiliser votre **ordinateur personnel** et n'avez encore jamais installé Python.

Il y a de nombreuses manières d'installer et utiliser Python. Vous trouverez de nombreux guides, tutoriels et documentations sur internet si vous souhaitez installer et prendre en main Python à votre manière. Ce document vous propose d'installer Python par le biais de Miniconda.

La page contenant tous les téléchargements se trouve ici. Téléchargez le fichier correspondant à votre système (choisissez bien Miniconda : la version Anaconda est payante), puis suivez les instructions ci-dessous.

Windows : Lancez le fichier, suivez les instructions et c'est tout.

MacOS : Identique à Windows, lancez-le et suivez les instructions.

Linux : Après avoir téléchargé le script d'installation (a priori le x86) :

1. Ouvrez un terminal (dépend de votre distribution, cherchez une application *Terminal* ou *Shell* par exemple).
2. Déplacez-vous vers le dossier contenant le fichier d'installation :
   `cd path/to/the/file`
3. Si nécessaire, autorisez l'exécution du fichier comme ceci :
   `chmod u+x Miniconda3-latest-Linux-x86_64.sh`
4. Lancez-le :
   `./Miniconda3-latest-Linux-x86_64.sh`
5. Acceptez la licence (appuyez sur ESPACE jusqu'à la fin puis tapez `yes`).
6. Laissez le chemin d'installation par défaut et pressez ENTREE (ou changez le chemin d'installation si nécessaire).
7. Répondez `yes` lorsqu'on vous propose de configurer Miniconda en exécutant `conda init`.

Quand l'installation est terminée, vérifiez-la en tapant `conda` dans le terminal (ou dans le Command Prompt sur Windows) : si une liste d'options et de commandes s'affiche, Miniconda est bien installé.

Une fois Miniconda installé, il faudra également installer les trois packages que nous utiliserons en cours : `numpy`, `scipy` et `matplotlib`. Pour installer un package :

1. Ouvrez un terminal/Command Prompt.
2. Écrivez : `conda install nom_du_package1 nom_du_package2 ...`

Par exemple, vous pouvez installer simultanément les trois packages du cours en tapant :

```
conda install numpy scipy matplotlib
```

Il vous faudra également installer un éditeur de code, par exemple `spyder` ou `jupyter` : ils s'installent aussi avec la commande `conda install`, exactement comme les packages.

Fondamentaux de Python

Ce document présente une introduction très succincte à Python, et suppose que vous avez déjà de l'expérience avec les notions de bases. **Il est essentiel d'être à l'aise avec les notions présentées dans cette section** pour les séances de TD. Si vous débutez entièrement ou souhaitez avoir plus de détails, n'hésitez pas à chercher des ressources extérieures : il existe de nombreux documents et tutoriels en ligne, comme celui-ci par exemple.

1. Si vous utilisez votre ordinateur personel, installez Python (ou Python3) comme expliqué ci-dessus si ce n'est pas déjà fait. Ma recommandation personnelle est l'installation par Anaconda, mais ce n'est pas important.
2. Ouvrez un éditeur : sur les ordinateurs du labo, ce sera en général Jupyter. Pour ouvrir Jupyter ou Spyder, vous pouvez normalement les trouver dans votre liste d'applications/de logiciels, ou alors les lancer depuis le terminal en tapant, au choix :

   ```
   jupyter notebook
   spyder
   ```

3. Dans Spyder, repérez les différentes fenêtres : par défaut, vous devriez avoir l'éditeur de code à gauche, la console en bas à droite, et la fenêtre d'aide et d'affichage graphique en haut à droite. Si souhaité vous pouvez personnaliser l'interface à votre guise.

   Pour Jupyter, l'application se lance dans un onglet de votre navigateur internet, et ressemble à un explorateur de dossiers. Déplacez-vous dans le dossier de votre choix, puis cliquez sur File, New pour créer un nouveau notebook, enregistrez-le où vous voulez, et ouvrez-le (il s'ouvrira dans un nouvel onglet). Le notebook est composé de Cells (cellules) de code, lorsque vous en compilez une le résultat s'affichera juste en-dessous (s'il y a une erreur dans le code la console s'affichera à la place).
4. *C'est parti !* Entrez les lignes de code suivantes une par une et comprenez ce qu'elles font. Dans Spyder, vous pouvez exécuter tout le fichier de code en pressant F5 ou CTRL+ENTREE, ou bien presser F9 pour exécuter le code sélectionné (ou la ligne actuelle si rien n'est sélectionné). Sur Jupyter vous pouvez exécuter la cellule actuelle avec MAJ+ENTREE, et vous avez une commande dans la bar d'outil pour exécuter l'intégralité du notebook.

   ```
   print("hello")
   a = 3
   a
   print(a)
   type(a)
   b = 2*a**2+1
   b
   c = [1,2,3,4,5]
   type(c)
   len(c)
   c[0]
   c[2]
   c[:2]
   c[3:]
   c.append(6)
   ```

5. *Boucles for* : Exécutez et comprenez le code suivant. Attention à taper correctement l'indentation (automatique sur la plupart des éditeurs, sinon pressez TAB) et les deux points :

   ```
   for i in [0,1,2,3]:
     print(i)
   ```

Faites de même avec celui-ci :

```
for j in range(4):
  print(j)
for j in range(1,4):
  print(j)
```

6. *Commentaires* : On utilise # pour commenter une ligne de code. Il sera utile d'identifier les raccourci claviers permettant de commenter/dé-commenter une partie sélectionnée du code (CTRL+/ sur Jupyter, CTRL+1 sur Spyder).

```
a = 1 #Ceci est un commentaire
#a = 3 #Cette ligne sera ignorée par l'ordinateur
print(a)
#Prenez l'habitude de rajouter des commentaires utiles dans votre code,
#ils vous serviront autant que vos camarades ou futurs collègues!
```

7. *Tests if* : Comprenez les lignes de code suivantes :

```
a = 8
if a>4:
  print("a est plus grand que 4")
elif a<4:
  print("a est plus petit que 4")
else:
  print("a est égal à 4")
```

8. *Fonctions* : Comprenez le code suivant :

```
def square(x):
  return x**2
print(square(5))

def power(x,a=1):
    return x**a
print(power(5,2))
print(power(5))
```

9. *Indentation* : Python utilise l'indentation pour interpréter correctement le début et la fin d'une boucle for, d'un test if, de la définition d'une fonction... Essayez le code suivant, et rajoutez l'indentation manquante pour le faire fonctionner correctement.

```
#Ce code écrit tous les nombres de 0 à 9 sauf 4, puis "Fini!" à la toute fin.
for i in range(10):
if not i==4:
print(i)
print("Fini!")
```

10. *Help* : Pour comprendre comment utiliser une commande, vous pouvez utiliser la commande intégrée `help` pour avoir des informations et exemples.

```
s = sum(1,2,3,4) # Renvoie une erreur
help(sum) # La console montre que sum doit prendre en argument un seul objet
# itérable, comme une liste ou un range.
s = sum([1,2,3,4])
print(s)
```

Attention, le jour de l'examen vos machines ne seront **pas connectées à internet**. Prenez l'habitude d'utiliser la commande `help` et la documentation de python.

11. *Importer des packages* : Nous allons utiliser commandes venant de modules/packages, qui doivent être importées préalablement. Si cela ne marche pas, vérifiez que vous avez installé correctement les packages :

```
a = np.array([1,2,3]) #erreur, car la fonction n'a pas été importée.
import numpy as np
a = np.array([1,2,3])
a
a[2]
```

12. *Plot basique* : Essayez le code suivant, et utilisez les commandes `print` et `help` pour comprendre l'utilité des différentes fonctions. Remarquez que numpy n'a plus besoin d'être importé, puisqu'il l'a déjà été au point précédent. Aussi, dans Spyder il sera utile de sélectionner l'onglet "Plots" de la fenêtre en haut à droite.

```
import matplotlib.pyplot as plt
x = np.linspace(0,2*np.pi,100)
y = np.cos(3*x)
plt.plot(x,y)
```

13. *Variables aléatoires* : Essayez le code suivant.

```
import scipy.stats as st
st.uniform.rvs(0,1,size=10)
v=st.poisson.rvs(mu=2., size=500)
np.mean(v)
np.var(v)
```

Notions avancées de Python

On présente ici des notions plus avancés que dans la section précédente. **Il n'est pas nécessaire de les maîtriser** pour faire les TD, mais elles peuvent être utiles pour coder de manière concise et efficace, ou pour écrire des programmes plus avancés.

14. *Instructions multiples sur une ligne* : Essayez le code suivant :

```
a = 2; b = 5; print(a+b)
# À utiliser avec modération, votre code doit rester lisible...
```

15. *Listes avancées* : Essayez les commandes suivantes. Utilisez la commande `print` pour comprendre chaque ligne de code si nécessaire :

```
a = [0]*5
a.append(5); a += [6]
b = [i**2 for i in range(5)]
del b[2]
```

16. *Strings avancés* : Il est possible de convertir une variable en string grâce à `str(...)` :

```
print("Bon"+"jour") # Concaténation de strings
x = np.log(2)
print("la solution de l'équation exp(x)=2 est x=" + str(x))
x2 = "{:.4f}".format(x) # Troncature du développement décimal de x
print("la solution de l'équation exp(x)=2 est environ x=" + str(x2))
```

17. *Erreurs courantes de plot* : Essayez le code suivant en compilant les lignes une à une, puis tout en une seule fois. Pensez à bien importer `numpy` et `matplotlib.pyplot` comme précédemment.

```
x = np.linspace(0,2*np.pi,100)
y = np.cos(3*x)
plt.plot(x,y)
plt.plot(y)
```

Pour séparer correctement des représentations graphiques différentes, on pourra utiliser les commandes `plt.figure()` et `plt.show()` respectivement au début et à la fin de chaque plot.

18. *Mise en forme d'un plot* : Expérimentez avec le code suivant.

```
x = np.linspace(0,2*np.pi,100); y = np.cos(3*x); z = np.sin(3*x)
plt.figure()
plt.plot(x,y, label = 'cosinus')
plt.plot(x,z, label = 'sinus', color = 'r')
plt.title("Exemple de plot")
plt.legend()
plt.show()
# Autres options possibles pour l'attribut color ci-dessus:
# ["b","r","g","y","m","c","k","purple","pink",(0.,0.,1.)]
```

19. *Plot de plusieurs graphiques dans une seule figure* :

```
fig,(ax1,ax2) = plt.subplots(ncols=2)
ax1.plot(x,y)
ax2.plot(x,z)
ax1.set_title('cosinus')
ax2.set_title('sinus')
fig.suptitle('Exemple de plot multiple')
plt.show()
```

20. *Histogramme* :

```
x = st.uniform.rvs(-1,2,size=500)
t = np.linspace(-1,1,num=50)
plt.figure()
plt.hist(x,bins = t, density = True)
plt.plot(t,[.5]*np.size(t), color ='r', linestyle = '--')
plt.show()
```

21. *Listes vs. np.array* : En plus du type `list` présent de base dans Python, certains modules (numpy, scipy.stats) utilisent le type `np.array`, qui n'utilise pas exactement les même commandes. Expérimentez avec le code suivant.

```
a = [1,2,3,4,5,10]
b = np.array(a) # Conversion d'une liste en np.array
np.size(b) # Equivaut à len(a)
np.sum(b) # Equivaut à sum(a)
np.cumsum(b) # Pas d'équivalent pour les listes
# De la même manière, on peut appliquer des fonctions déjà implémentées
# simultanément à toutes les entrées d'un np.array
b**2 + 1
np.sqrt(b)
print(b>2); print(2*(b>2)) # Idem avec les tests
```

22. *Matrices* : Le type `np.array` permet également de faire du calcul matriciel simple.

```
A = np.zeros(shape=(2,3)) # Créer une matrice nulle
A[0,1] # Extraire une valeur
A[:,1] # Extraire une colonne entière
A[0,1] = 5 # Modifier une valeur
v = np.array([1,2,3])
A[1,:] = v # Modifier une ligne entière
x = np.arange(6)
B = np.reshape(x,(3,2)) # Créer une matrice à partir d'un np.array
C1 = np.transpose(A) + B; C2 = 2*A+1
D = np.transpose(A)*B # Produit terme à terme
E = np.matmul(A,B) # Produit matriciel usuel
E = A@B # Equivalent à np.matmul(A,B)
I = np.eye(3) # Matrice identité
J = np.ones((2,4)) # Matrice de 1
```

23. *Algèbre linéaire* : `numpy` contient un sous-module `linalg` très utile pour l'algèbre linéaire.

```
print(np.linalg.det(E)) # Déterminant
F = np.linalg.inv(E) # Inverse
print(E@F)
eigenvalues, eigenvectors = np.linalg.eig(E) # Valeurs et vecteurs propres
print("Valeurs propres:")
print(eigenvalues)
print("Vecteurs propres:")
print(eigenvectors) # Attention, les vecteurs propres sont affichés en colonne
v1 = eigenvectors[:,0]
lam1 = eigenvalues[0]
print(lam1*v1); print(E@v1)
```

24. *Syntaxe abrégée* : Il y a de nombreuses façons de coder de manière concise en Python. Essayez les exemples suivant. Utilisez cette syntaxe ou non, c'est une affaire de goût.

```
l1 = [i**2 for i in range(10) if i%2]
l2 = [i**2 if i%2 else -1 for i in range(10)]
l3 = [(i>0)*i for i in l2]
A = np.array([[i+j for j in range(3)] for i in range(3)])
Aeven = A%2==0
LabelsAeven = np.transpose(np.where(A%2==0))
```

Remarque finale : Ce document n'est rien de plus qu'une introduction très brève à Python, elle oublie de nombreuses choses. Il est fortement recommandé d'expérimenter avec les exemples ci-dessus, surtout s'il ne sont pas complètement compris ; mais aussi de chercher des ressources complémentaires en ligne, d'exploiter au maximum la commande `help`, et de continuer d'apprendre à manipuler des objets comme les matrices ou les fonctions de plot graphique.