

TP1 -- Informatique CM3 -- Septembre 2014

Python @ Polytech'Lille

Le texte de cette sessions de travaux pratiques est également disponible ici

<http://nbviewer.ipython.org/github/ecalzavarini/python-at-polytech-lille/blob/master/Python-TP1-2014.ipynb>
(<http://nbviewer.ipython.org/github/ecalzavarini/python-at-polytech-lille/blob/master/Python-TP1-2014.ipynb>)

Introduction aux commandes Linux

Avant de commencer avec notre introduction au langage de programmation Python, nous devons familiariser avec l'environnement du système d'exploitation Linux.

La première étape est d'apprendre les commandes les plus courantes. Ces commandes peuvent être tapées sur une console Linux.

Pour savoir comment accéder à une console sur la distribution Linux Debian dont vous disposez, voir: <https://wiki.debian.org/fr/Console> (<https://wiki.debian.org/fr/Console>) ou simplement recherchez l'application "terminal" sur le menu des applications.

La commande pwd

La commande "pwd" affiche le répertoire courant, c'est à dire le répertoire où l'on se trouve au moment où on tape la commande.

La commande ls

La commande "ls" permet de lister ce que contient un répertoire. Il y a plusieurs manières de l'utiliser :

"ls" donne la liste brute de ce qui se trouve dans le répertoire

"ls -l" donne la liste de ce qui se trouve dans le répertoire avec des informations complémentaires (droits de lecture, écriture et exécution, propriétaire, taille, date de création ou de dernière modification).

"ls -a" liste tous (pensez au mot "all") les fichiers du répertoire, y compris les fichiers cachés. Cette option est très utile lorsque l'on se trouve dans son répertoire personnel car il contient les fichiers de configuration de l'utilisateur dont les noms commencent généralement par un point et seule l'option -a permet de détecter leur existence.

La commande mkdir

La commande "mkdir" permet de créer un répertoire.

"mkdir nom_repertoire" crée le répertoire ayant pour nom nom_repertoire.

La commande cd

La commande "cd" permet de se déplacer dans l'arborescence de répertoires. Il y a plusieurs manières de l'utiliser :

"cd nom_repertoire" permet d'aller dans le répertoire nom_repertoire

"cd .." permet de retourner dans le répertoire parent (i.e. Celui qui est au-dessus dans l'arborescence de répertoires).

L'éditeur de texte emacs

Pour créer ou modifier un fichier "emacs -nw nom_fichier.txt"

Pour sauvegarder tapez la combinaison de touches suivante: Ctrl-x Ctrl-s.

Pour quitter emacs: Ctrl-x Ctrl-c.

La commande rm

La commande "rm" permet de supprimer un fichier :

"rm nom_fichier" supprime le fichier nom_fichier si il se trouve dans le répertoire courant.

"rm -r" supprime un répertoire et ses sous répertoires.

La commande top

La commande "top" affiche en continu des informations décrivant l'activité du système. Elle permet surtout de suivre les ressources que les processus utilisent (quantité de mémoire, pourcentage de CPU...). Sous top il est possible d'expédier de manière interactive un signal à un processus, par exemple afin de le stopper, en tapant "k", top demande ensuite l'identifiant (PID) du processus concerné. Pour quitter top, appuyer simplement sur la touche "q".

Archivage de données (tar)

La commande "tar" gère des archives, contenant chacune au moins un répertoire ou fichier.

Vous aurez souvent besoin de "tar xzf nom_du_fichier.tar.gz" , qui décompacte une archive au format .tar.gz ou .tgz. L'extension .tar.gz ou .tgz indique que le fichier est une archive tar et qu'il est compacté. Les arguments (aussi appelés options) employés dans la commande précédente (« xzf ») peuvent être ainsi compris:

x (extraction) déclenche l'extraction de certains fichiers d'une archive (lorsque l'on ne spécifie pas les noms des fichiers que l'on souhaite extraire de l'archive, tar les extrait tous)

z décompacte l'archive grâce à la commande gzip

f traite un fichier-archive dont le nom suit (ici: « nom_du_fichier.tar.gz »)

Si je me trouve dans le répertoire « « /home/delcros/ » » la commande suivante créera une archive du répertoire « /home/delcros/personnel » :

```
"tar cvzf personnel.tgz personnel"
```

l'option "c" permet de créer une archive

L'option "z" compacte l'archive grâce à gzip.

Questions :

1) Ouvrir une console Linux et créer un répertoire TP-PYTHON dans le répertoire courant. Comment vérifier qu'il a bien été créé ?

2) Aller dans le répertoire que vous venez de créer, puis lister ce qu'il contient. Quelle commande vous permet de savoir si vous êtes bien dans le répertoire TP-PYTHON ?

3) Créer un fichier appelé README.txt dans le répertoire TP-PYTHON avec la commande "emacs", taper quelques lignes dedans. Enregistrer le fichier. Vérifier qu'il a bien été créé.

4) Créer un fichier-archive du repertoire TP-PYTHON

5) Supprimer le fichier que vous venez de créer, comment vérifier qu'il a bien été supprimé ?

6) Supprimer le répertoire TP-PYTHON. Vérifier qu'il a bien été supprimé.

7) Lancer l'interprète du langage Python (écrire "python" dans la console de Linux). Supposons maintenant que, pour une raison quelconque, le programme est bloqué. Tuez l'interprète Python en utilisant la commande "top".

Introduction à la programmation numérique en Python

Pour lancer l'interprète du langage Python écrire "python" dans la console de Linux.

Saisir votre série d'instruction Python après l'invite de la ligne de commande ">>>" (à noter que chaque instruction doit être validée avec la touche "return").

Une deuxième possibilité d'utilisation beaucoup plus pratique est de créer un fichier (appelé script), par exemple avec l'éditeur de text emacs : "emacs -nw script.py"

Une fois que les instructions ont été écrites dans ce fichier, le script peut être exécuté avec la commande "python script.py".

Le site le plus utile pour trouver des informations sur python est : <https://docs.python.org/2/> (<https://docs.python.org/2/>). voir notamment dans les sections "Tutorial" et "Language reference".

Pendant ce TP vous aurez à écrire plusieurs scripts (Nous vous suggérons de les nommer script1.py , script2.py ,...)

Les scripts doivent être accompagnés par un par un document descriptif unique (README.txt). Dans ce fichier, vous devrez décrire le mode de fonctionnement des scripts et, si besoin, mettre vos commentaires. Merci d'y écrire aussi vos nomes et prenomns complets.

Tous les fichiers doivent etre mis dans un dossier appelé TP1-nom1-nom2 et ensuite être compressés dans un fichier TP1-nom1-nom2.tgz .

Enfin vous allez envoyer ce fichier par email à l'enseignant : soit Enrico (enrico.calzavarini@polytech-lille.fr) ou Stefano (stefano.berti@polytech-lille.fr)

1) Script 1 : entrer ou afficher du texte et des données

Nous voulons écrire un script qui tout d'abord affiche simplement une message de bienvenue :

```
In [2]: print("Bienvenue au premier TP d'informatique")
```

```
Bienvenue au premier TP d'informatique
```

Vérifiez que le programme ci-dessus fonctionne quand il est écrit dans un script (appelé par exemple script1.py) et interprété par python avec la commande "python script1.py".

Nous souhaitons maintenant que ce programme nous demande aussi combien d'étudiants sont présents dans la salle et qu'il affiche ce numéro sur l'écran.

A noter que la lecture de données peut se faire par interrogation de l'utilisateur. Par exemple :

```
In [18]: p = input("Entrer la valeur de la precision p, p =")
```

Entrer la valeur de la precision p, p =1.0

l'écriture des données enregistrées dans une variable peut se faire tout simplement encore avec la fonction "print"

```
In [19]: print("la valeur de la precision est :" + str(p) )
```

la valeur de la precision est :1.0

ou de façon équivalente :

```
In [20]: print("la valeur de la precision est : %e" %p )
```

la valeur de la precision est : 1.000000e+00

Commentaires en python

Une bonne habitude à prendre est d'écrire des commentaires dans le script. Nous pouvons le faire avec le symbole "#", par exemple :

```
In [21]: print("la valeur de p est : %e" %p ) # ici nous imprimons le résultat final
```

la valeur de p est : 1.000000e+00

Écrivez donc votre premier script!

2) Script2 : effectuer des opérations simples sur les variables

Nous souhaitons écrire un script qui fait la conversion d'unité pour des mesures de pression.

Ce script devra demander à l'utilisateur d'introduire une valeur de pression et une unité de mesure (choisi parmi: Pascal (pa) , Bar (bar) , Atmosphere standard (atm) , Torricelli (torr) et Pounds per square inch (psi)), puis demander à l'utilisateur quelle unité il souhaite pour la conversion et enfin afficher la valeur de pression dans l'unité demandée.

Pour mener à bien ce script les informations suivantes peuvent être utiles :

La table de conversion des unités de pression peut être repérée ici :

http://en.wikipedia.org/wiki/Pressure_measurement#Units
(http://en.wikipedia.org/wiki/Pressure_measurement#Units)

L'affectation de données dans des variables

```
In [2]: a = 5.0  
b , c = 10 , 3  
name = "Mark"
```

Ici "a", "b", "c" et "name" sont des identificateurs (ou simplement des variables).

Python ne nécessite pas de déclarer explicitement les types de variables, différemment qu'en Fortran, C et d'autres langages. Juste affectez une variable et Python comprendra automatiquement, sans préciser de type (réel, entier, caractère, etc...)

Vous pouvez demander à Python de vous dire de quel type il a attribué à vos variables:

```
In [4]: type(a)
```

```
Out[4]: float
```

```
In [5]: type(b)
```

```
Out[5]: int
```

```
In [6]: type(name)
```

```
Out[6]: str
```

Portez une attention particulière à l'attribution de valeurs à virgule flottante à des variables ou vous risquez d'obtenir des valeurs que vous ne vous attendez pas dans vos programmes. Par exemple ,

```
In [3]: print(b/c)
```

```
3
```

Vous voyez , si vous divisez un nombre entier par un nombre entier , Python retourne une réponse arrondie à l'entier le plus proche . Mais si vous voulez une résultat à virgule flottante (type float), l'un des numéros doit être de type float. Le simple ajout d'un point décimal fera l'affaire :

```
In [5]: b , c = 10. , 3.  
print (b/c)
```

```
3.333333333333
```

Utilisation des listes

Une liste est une collections d'objets :

```
In [28]: l = [0, 5, 6, 4]  # Definition et affectation d'une liste  
  
print(l[0])           # Les indices commencent à 0, et non à 1  
  
print(l[3])           # Le dernier indice est le 3ème  
  
print(l[-1])          # On peut compter à partir de la fin, en utilisant des indices négatifs  
  
print(l[0:3])         # On peut sélectionner une tranche (slice) d'éléments dans la liste  
print(l[0:3:2])       # La syntaxe du "slicing" [est start:stop:step]  
  
0  
4  
4  
[0, 5, 6]  
[0, 6]
```

Une liste peut être modifiée:

```
In [29]: l[2] = l[2]*2 + 1

print(l)

[0, 5, 13, 4]
```

Si nous voulons connaître le nombre d'éléments dans une liste :

```
In []: len(l)
```

Nous pouvons créer de la même manière une liste de mots :

```
In [14]: mots = ["arbre" , "avion" , "nuage"]

print(mots[1])

avion
```

Structure alternative "if"

Regardez l'exemple ci-dessous concernant le calcul des racines réelles d'une équation algébrique du second ordre, pour comprendre le fonctionnement d'une structure "if" :

```
In []: d = b**2-4*a*c

if d > 0 :
    x1 = (-b - sqrt(d))/(2*a)
    x2 = (-b + sqrt(d))/(2*a)
    print("RESULTATS :")
    print("Deux racines distinctes : x1 = " + str(x1) + " et x2 = " + str(x2))
elif d == 0 :
    x1 = -b/(2*a)
    print("RESULTATS :")
    print("Une racine double : x1 = " + str(x1))
else :
    print("RESULTATS :")
    print("Aucune solution!")
```

à noter : indentation nécessaire en langage Python

Pour les structures de contrôle du langage Python, comme "if"(et nous verrons ensuite "for"), il n'y a pas des mots clés "end" pour signaler la fin du bloc de lignes concernées par les structures "if","while","for". Il faut par contre indenter les lignes (c'est à dire créer des décalages à l'aide de la touche tabulation "tab" ou "espace" du clavier) afin de définir une dépendance d'un bloc de lignes par rapport à un autre.

Script 3 : lire des données à partir d'un fichier et les élaborer

Nous disposons des données de pression enregistrées (à la fréquence d'une mesure par seconde) par un tube de Pitot d'un avion en vol à une altitude de 10000 mètres :

<https://github.com/ecalzavarini/python-at-polytech-lille/blob/master/pressure.txt>
(<https://github.com/ecalzavarini/python-at-polytech-lille/blob/master/pressure.txt>)

Vous pouvez télécharger ce fichier avec la commande de Linux " wget ". Ecrivez sur votre console : "wget adresse_internet_du_fichier".

```
In [8]: from IPython.display import Image
Image('http://upload.wikimedia.org/wikipedia/commons/0/00/Pitot_tube_wings-fr.jpg')
```

Out[8]:



Tube de Pitot

On est intéressé à écrire un script qui lit ces données et les transforme dans des mesures de vitesse. La relation entre la pression et la vitesse est donnée par la loi de Bernoulli:

```
In [30]: from IPython.display import Image
Image(filename='vel-press.jpg')
```

Out[30]:

$$V(t) = \sqrt{\frac{2(p(t) - p_s)}{\rho}}$$

Ici

$p(t)$ est la pression mesurée au cours du temps (t) en Pascal

p_s est la pression statique, $p_s = 24 \text{ kPa}$

ρ est la masse volumique de l'air, $\rho = 0.91875 \text{ kg/m}^3$.

Pour la réalisation de ce script les informations suivantes vous seront utiles :

La lecture des données

```
In [33]: # Ouverture d'un fichier
f = open("pressure.txt", "r")
# définition d'une liste vide
data=[]
# lecture de fichier complet et affectation des éléments de la liste
data=f.read().split()
# fermeture du fichier f
f.close()

#contrôle de type
print( type(data[0]) )
#impression de la première donnée
print(data[0])

<type 'str'>
53274.300000
```

```
In [35]: #conversion au type désiré
data[0] = float(data[0])
#contrôle de type
print ( type(data[0]) )
#impression
print data[0]

<type 'float'>
53274.3
```


à noter que lorsque les données sont lus de la manière ci-dessus, ils sont stockés dans une liste de type chaîne de caractères (string). Si nous devons effectuer des opérations mathématiques sur eux, nous devons les convertir en un type numérique (soit "float" ou "int").

Ecriture dans un fichier

```
In [20]: vall = 5.0  # affectation d'une variable réelle

val2 = str(vall) # conversion en chaîne de caractères

# Ouverture d'un nouveau fichier
f = open("output.txt", "w")
# écriture dans le fichier
f.write(val2)
# fermeture du fichier
f.close()
```

à noter que la commande "write" ne peut écrire que des chaînes de caractères (string).

La structure répétitive "for"

Regardez l'exemple ci-dessous concernant tri par sélection de données, pour comprendre le fonctionnement d'une structure boucle pour "for" :

```
In [38]: # Tri par sélection
# Données initiales
M = [1. , 2. , 5. , 40. , 65. , 2.,8. , 98. ,115. ,0. , 3.]

# Détermination de la longueur de la liste
N = len(M)

for i in range(0,N-1):
    for j in range(i+1,N):
        if M[j] < M[i]:
            val_temp = M[i]
            M[i] = M[j]
            M[j] = val_temp
print("Liste triée :")
print(M)

Liste triée :
[0.0, 1.0, 2.0, 2.0, 3.0, 5.0, 8.0, 40.0, 65.0, 98.0, 115.0]
```

à noter que la commande " range(a,b) " produit une liste de nombres entiers progressifs compris entre a et b-1. Par exemple :

```
In [40]: range(0,10)

Out[40]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Utiliser les fonctions mathématiques

En Python afin de pouvoir utiliser les fonctions mathématiques (par exemple, exp, sqrt , sin , cos , tan), vous devez d'abord les importer à partir de la bibliothèque mathématique "math" :

```
In []: from math import *           # importing all functions from the math library
```

Nous vous demandons donc de lire la série des données de pression dans le fichier et de calculer la vitesse de l'avion en utilisant la relation donnée.

On demande ensuite de calculer la valeur moyenne de la vitesse de l'avion (en Km/heure) et de l'écrire (avec print).

Question bonus : Seriez-vous capable de calculer l'accélération de l'avion au cours du temps, et de l'enregistrer dans un fichier?

Script 4 : représentation graphique des données

Nous souhaitons analyser graphiquement les données , par exemple en les traçant ou en faisant leur histogramme .

Pour la réalisation de ce script les informations suivantes vous seront utiles :

Bibliothèques de Python

Python dispose de nombreuses bibliothèques qui fournissent des fonctionnalités avancées comme la possibilité d'effectuer des opérations sur des matrices, des fonctions graphiques, et bien plus encore. Nous pouvons donc importer ces bibliothèques de fonctions pour étendre les capacités de Python dans nos programmes.

Nous allons commencer par l'importation de quelques bibliothèques pour nous aider.

Premièrement, nous l'avons déjà vu, nous utilisons Math, la bibliothèque mathématique standard qui comprend les fonctions trigonométriques et transcendantes, ainsi que d'autres fonctions spéciales.

Pour voir la liste complète des fonctions de la bibliothèque Math:
<https://docs.python.org/2/library/math.html#module-math>
(<https://docs.python.org/2/library/math.html#module-math>)

Notre deuxième bibliothèque préférée est NumPy (Numerical Python), fournissant un grand nombre d'opérations matricielles utiles (cette bibliothèque est similaire à MATLAB). Nous allons l'utiliser beaucoup!

La documentation de numpy se trouve ici :
<http://docs.scipy.org/doc/numpy/reference/>
(<http://docs.scipy.org/doc/numpy/reference/>)

La troisième bibliothèque dont nous avons besoin est Matplotlib, une bibliothèque de graphique 2D que nous allons utiliser pour tracer nos résultats.

La documentation de référence est disponible ici :
<http://matplotlib.org/contents.html> (<http://matplotlib.org/contents.html>)

Le code suivant sera au sommet de la plupart de vos programmes :

```
In [4]: from math import *           # importing all functions from the math library
import numpy as np                 # we import the array library, and call it 'np'
import matplotlib.pyplot as plt    # import plotting library, and call it 'plt'
```

Nous importons donc toutes les fonctions de la bibliothèque "math" et nous importons aussi une bibliothèque nommée "numpy". Enfin, nous importons un module appelé "pyplot" d'une grande bibliothèque appelée "matplotlib".

Les deux dernières lignes ci-dessus ont créé des raccourcis pour les bibliothèques tel que "np" et "plt" respectivement. La raison de la création des raccourcis, c'est que nous avons besoin de taper les noms des bibliothèques assez souvent dans le code. Pour utiliser une fonction appartenant à l'une de ces bibliothèques, nous devons dire à Python où la chercher. Pour cela, chaque nom de fonction est écrit après le nom de la bibliothèque, avec un point entre les deux. Par exemple :

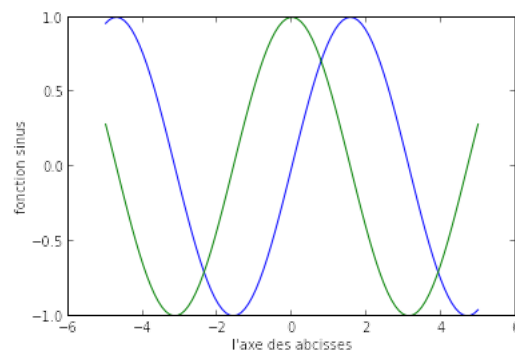
```
In [2]: np.zeros(5)
```

```
Out[2]: array([ 0.,  0.,  0.,  0.,  0.])
```

Tracer des courbes

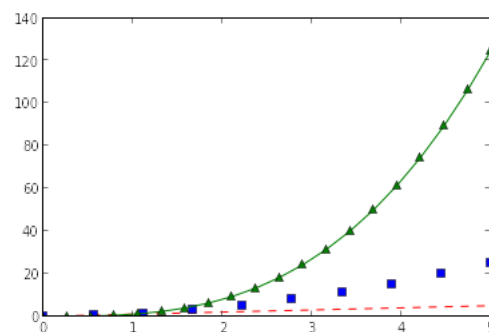
```
In [10]: %matplotlib inline
#ignorez la ligne ci-dessus. c'est juste une commande de notre éditeur de texte
```

```
In [11]: x=np.linspace(-5,5,100) # nous définissons une liste (array) avec Numpy
plt.plot(x,np.sin(x)) # on utilise la fonction sinus de Numpy
plt.plot(x,np.cos(x))
plt.ylabel('fonction sinus')
plt.xlabel("l'axe des abscisses")
plt.show()
```



Changer le style des courbes

```
In [9]: t1=np.linspace(0,5,10)
t2=np.linspace(0,5,20)
plt.plot(t1, t1, 'r--', t1, t1**2, 'bs', t2, t2**3, 'g^-')
plt.show()
```

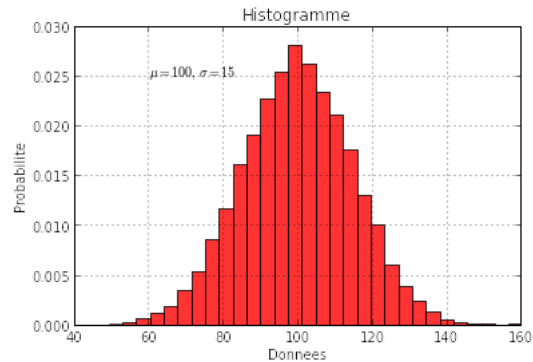


Un histogramme et un affichage de texte sur le graphique

```
In [8]: mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000) #generation de 10000 nombres aleatoires

bins = 30

# histogramme des donnees
plt.hist(x, bins, normed=1, facecolor='r', alpha=0.8)
plt.xlabel('Donnees')
plt.ylabel('Probabilite')
plt.title('Histogramme')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
```



Questions :

D'abord, nous demandons d'écrire un script qui trace un graphique de la pression (en kPa) en fonction du temps (en heures). N'oubliez pas d'indiquer les étiquettes sur les axes.

Sur le même graphique tracer aussi trois lignes (en couleurs différentes) l'une représentant la valeur moyenne (μ) de la pression et les deux autres la valeurs moyennes plus / moins l'écart type (σ).

```
In [32]: from IPython.display import Image
Image(filename='mean-sigma.jpg')
```

Out[32]:

$$\mu = \frac{1}{N} \sum_{i=1, N} x_i$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1, N} (x_i - \mu)^2}$$

Deuxièmement, nous demandons également de construire un histogramme, mais cette fois de la vitesse (en Km/h).

Ignorez la cellule ci-dessous. Elle charge notre style d'affichage pour ce document.

```
In [8]: from IPython.core.display import HTML
def css_styling():
    styles = open('custom.css', 'r').read()
    return HTML(styles)
css_styling()
```

Out[8]: