Faculté des sciences

Université Moulay Ismail Meknes

Cours Analyse de données Master Géologie

Séance N°1

Le langage Python

2019/2020

Mohamed Berrada

ENSAM de Meknès

Table des matières

1. Introduction

Python est un langage de programmation polyvalent et multiplateforme. Sa première version est sortie en 1991. C'est le langage favori d'un très grand nombre de développeurs comparé à d'autres langages comme Java, PHP, C++, etc. Python est libre, c'est-à-dire que l'on peut modifier, copier, diffuser en toute liberté. En plus, il est facile à apprendre à cause de sa syntaxe qui présente une certaine clarté, ce qui en facilite la lecture et la compréhension, même si on n'est pas un expert en Python.

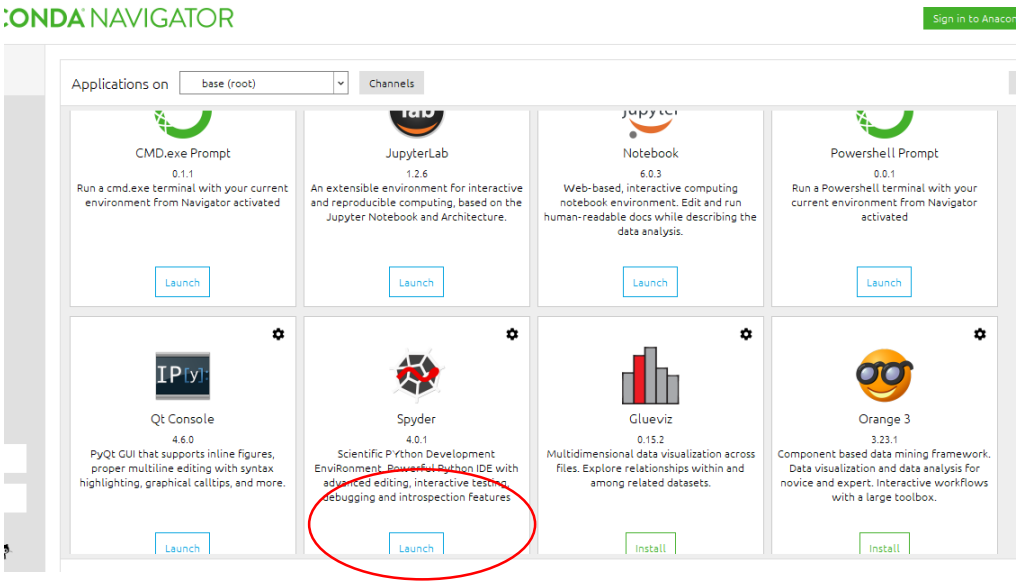
Le langage python est utilisé dans presque tous les domaines de l'informatique : développement Web, cloud computing, science des données, Big Data, etc.

En python, il existe ce qu'on appelle des bibliothèques ou packages qui contiennent des scripts prêts à l'emploi. Il y en a des bibliothèques standard installées avec python et d'autres spécifiques à des projets particuliers qui peuvent être installées par le développeur à partir du dépôt PyPi.

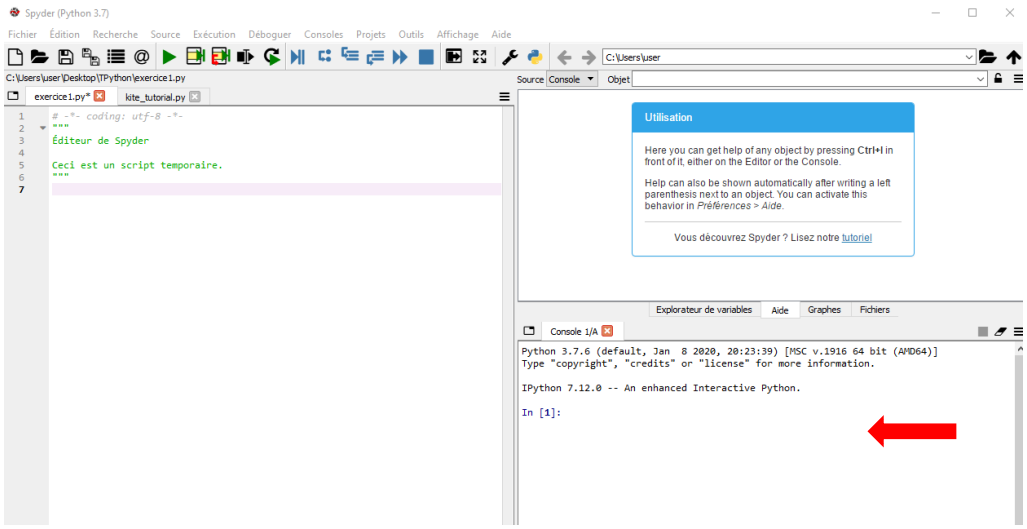
2. Installation

Pour nos travaux pratiques nous allons installer la plateforme anaconda depuis le site officiel « anaconda.com » et suivre les étapes. Pour écrire votre code vous pouvez utiliser les éditeurs de textes Spyder ou Jupyter qui s'installe automatiquement avec anaconda.

Après installation et en lançant anaconda, vous obtenez cette fenêtre :



Vous pouvez ensuite lancer Spyder pour écrire votre code Python :



Les lignes de codes que nous allons voir à travers les exemples de ce cours peuvent être lancer dans la partie droite en bas de la fenêtre Spyder. On peut aussi les taper dans un fichier (partie de gauche) et cliquer sur le petit triangle vert en haut de la fenêtre.

3. Calcul

Python est un langage interprété. Il met à la disponibilité du programmeur tous les opérateurs de calcul (+ ; - ; / ; * ; **).

```
>>> 1+2 # addition

3

>>> 1-10 # soustraction

-9

>>> 2*10 # produit

20

>>> 100/4 # division

25

>>> 10%4 # reste de la division

2

>>> 2**3 # puissance

8

>>> 10 // 3

3
```

```
>>> age = 30 # affectation de la valeur 30 à la variable age
>>> age
30
>>> age = age + 10 # utilisation de la variable age dans une opération
>>> age
40
```

Nous pouvons appliquer sur les chaines de caractères les mêmes opérateurs utilisés pour les nombres à part la division.

```
>>> age = "J'ai 30 ans" # type chaine de caractère
>>> age
"J'ai 30 ans"
>>> mon_age = age + " et je suis encore jeune !" #concaténer deux chaines de caractères
>>> mon_age
"J'ai 30 ans et je suis encore jeune !"
>>> ton_age = "jeune" # multiplier la chaine de caractères
>>> ton_age * 3
'jeunejeunejeune'
```

4. Variables

Une variable est une boite virtuelle qui sert à stocker temporairement une valeur afin de le réutiliser ultérieurement dans un code. Une variable est interprétée par la machine comme une adresse vers la mémoire vive où sont stockées les informations relatives. L'affectation se fait simplement par le signe « = ».

Pour connaitre le type d'une variable, nous pouvons utiliser la fonction « type() ».

```
>>> v = 15
>>> type(v)
<type 'int'>
>>> v = "Olivier"
>>> type(v)
<type 'str'>
```

```
>>> v = 3.2

>>> type(v)

<type 'float'>
```

Exercice :

Essayez de prédire le résultat de chacune des instructions suivantes, puis vérifiez-le dans l'interpréteur Python :

- (1+2)**3
- "Da" * 4
- "Da" + 3
- ("Pa"+"La") * 2
- ("Da"*4) / 2
- 5 / 2
- 5 // 2
- 5 % 2

5. Les listes

Une liste en python est une variable où nous pouvons stocker plusieurs valeurs ou plusieurs variables. Pour créer une liste nous utilisons les crochet « [ ] ». Dedans, nous écrivons les valeurs (ou les noms de variables) séparés par des virgules. Les listes peuvent contenir des valeurs de même type ou de types différents.

```
>>> liste = [] # liste vide

>>> liste = [1,2,3] # liste contenant trois valeurs de même type

>>> liste

[1, 2, 3]

>>> liste = [1,2,'ahmed'] # liste contenant trois valeurs de types différents

>>> liste

[1,2,'ahmed']
```

Les éléments de la liste sont indexés du 0 pour le premier élément jusqu'au n-1 pour le $n^{ème}$ élément. Nous pouvons, ainsi, appliquer plusieurs fonctions pour manipuler les listes, tel que, l'ajout d'une valeur, le remplacement d'une valeur ou la suppression d'une valeur.

```
>>> liste = ["a","d","m"]

>>> liste[0] # obtenir le premier élément de la liste

'a'

>>> liste[2] # obtenir le troisième élément de la liste

'm'
```

```
>>> liste[2] = "z" # changer la troisième valeur par "z"

>>> liste

['a', 'd', 'z']

---------------------------------------

>>> liste = [] # création d'une liste vide

>>> liste

[]

>>> liste.append(1) # ajouter la valeur 1

>>> liste

[1]

>>> liste.append("ok") # ajouter la valeur "ok"

>>> liste

[1, 'ok']

-----------------------------------------

>>> liste = ["a", "b", "c"]
>>> del liste[1] # suppression du deuxième élément de la liste par son indice
>>> liste
['a', 'c']

>>> liste = ["a", "b", "c"]
>>> liste.remove("a") # suppression du deuxième élément de la liste avec sa valeur
>>> liste
['b', 'c']

>>> liste = [1,2,3,5,10]
>>> len(liste)
5

>>> liste = ["a", "b", "c"]
>>> liste.reverse()
>>> liste
['c', 'b', 'a']

>>> liste = ["a","a","a","b","c","c"]
>>> liste.count("a")
3

>>> liste = ["a","a","a","b","c","c"]
>>> liste.index("b")
3
```

6. Les tuples

Un tuple est une liste qui ne peut pas être modifiée. Pour créer un tuple nous utilisons les parenthèses « ( ) ».

```
>>> mon_tuple = () # un tuple vide

>>> mon_tuple = (1, "ok", "olivier") # un tuple avec trois éléments
```

L'indexation des éléments dans un tuple se fait de la même façon que pour les listes.

7. Les dictionnaires

Un dictionnaire est une liste où les éléments sont indexés par des clés alphanumériques. Pour déclarer un dictionnaire nous utilisons les accolades « { } » ou par les fonction dict().

```
>>> a = {} # créer un dictionnaire vide appelé a

>>> a = dict()
```

Pour ajouter un élément au dictionnaire, il faut définir sa clé et sa valeur selon la syntaxe suivante : nom_dic["clé"] = "valeur"

Exemples :

```
>>> a["nom"] = "Wayne" # ajouter la clé-valeur ("nom" ; "Wayne")

>>> a["prenom"] = "Bruce" # ajouter la clé-valeur ("prenom" ; "Bruce")

>>> a # afficher le dictionnaire a

{'nom': 'Wayne', 'prenom': 'Bruce'}
```

Pour récupérer une valeur nous utilisons sa clé comme index.

Exemple :

```
>>> data = {"name": "Wayne", "age": 45} # dictionnaire contenant deux valeurs

>>> data.get("name") # fonction get() pour récupérer la valeur liée à la clé "name"

'Wayne'
```

8. Les fonctions

Sur python, nous pouvons stocker des instructions dans des fonctions que nous pouvons appeler par leurs noms.

Une fonction se définie par à l'aide du mot-clé « def », son nom, les paramètres et les instructions. La syntaxe générale est la suivante :

def nom_fonction(paramètre1, paramètre2,…) :

    block d'instructions

Exemple :

```
>>> def fonction1(a, b): # définir une fonction avec deux paramètres

...     return a + b     # l'instruction que va faire la fonction (addition de a et b)

...

>>> fonction1(1, 2)     # application de la fonction pour a = 1 et b = 2

3
```

9. Les conditions

Utiliser les conditions est une chose importante dans la programmation. Le principe est de définir des instructions qui s'exécutent quand une certaine (ou plusieurs) condition est vérifiée. La syntaxe générale des conditions en python utilise trois mot clés, « if » pour définir une condition, « elif » pour définir une deuxième condition et « else » pour définir le cas qui sort de toutes les conditions définies précédemment.

Exemples :

```
>>> a = 10         # initiation de la variable a par la valeur 10
>>> if a > 5:     # condition si a est supérieur à 5
...     a = a + 1 # instruction à exécuter si la condition est vérifiée
...
>>> a
11

>>> a = 20          # initiation de la variable a par la valeur 20
>>> if a > 5:      # condition si a est supérieur à 5
...     a = a + 1  # instruction à exécuter si la condition est vérifiée
... else:          # SI NON
...     a = a - 1  # instruction à exécuter si la condition n'est pas vérifiée
...
>>> a
21

>>> a = 5
>>> if a > 5:             # première condition
...     a = a + 1
... elif a == 5:          # deuxième condition
...     a = a + 1000
... else:                 # SI NON
...     a = a - 1
...
>>> a
1005
```

Nous pouvons utiliser plusieurs opérateurs de comparaison dans les conditions :

Comparaison	Symbole
égal à	= =
différent de	!=
strictement supérieur à	>
supérieur ou égal à	>=
strictement inférieur à	<

inférieur ou égal à	<=

Nous pouvons aussi écrire des conditions composées en utilisant les opérateurs logiques `and` et `or`.

```
>>> a = 20                  # initiation de la variable a par la valeur 20
>>> if a > 5 and a != 0:    # condition si a est supérieur à 5
...     a = 1/a             # instruction à exécuter si la condition est vérifiée
... else:                   # SI NON
...     a = a + 1           # instruction à exécuter si la condition n'est pas vérifiée
...
```

10. Les boucles

Les boucles sont utilisées en programmation pour pouvoir répéter des instructions plusieurs fois selon le besoin. En python, il y a deux types de boucles : « for » et « while ».

Exemple de la boucle while :

Par ce code, nous demandons à python d'afficher la phrase `"Je ne dois pas poser une question sans lever la main"` tant que i est inférieur à 10. Le « i » doit être initialisé avant la déclaration de la boucle. Dans notre exemple, la valeur initiale de i est 0 et la valeur définie dans la boucle 10.

```
>>> i = 0
>>> while i < 10:
...     print("Je ne dois pas poser une question sans lever la main")
...     i = i +1
...
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
Je ne dois pas poser une question sans lever la main
```

Exemple de la boucle for :

Dans ce code, nous demandons à python de calculer le carré de a pour chaque valeur que va prendre cette dernière de l'intervalle définie par la fonction range(). La fonction range(n) donne des valeurs de 0 à n-1. Dans notre cas, range(10) donne des valeurs de 0 à 9.

```
>>> for a in range(10):
...     print(a**2)
...
0
1
4
9
16
25
36
49
64
81
```

Exercice

Soit la liste['vache','souris','levure','bacterie']. Affichez l'ensemble des éléments de cette liste (unélément par ligne) de trois manières différentes (deux avecforet une avecwhile).

11. Les arrays de la bibliothèque Numpy

Une alternatif intéressante des Listes sous Python est les arrays de la bibliothèque Numpy. Ils sont plus rapide et plus simple et permettent au développeur de faire des calculs vectoriels (Somme, multiplication, produit scalaire,…).

La fonction arange() permet de construire un array à une dimension de manière simple.

```
>>> np.arange (10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

On peut spécifier en argument une borne de début, une borne de fin et un pas :

```
>>> np.arange (10, 0,  -1)
array ([10,   9,   8,   7,   6,   5,   4,   3,   2,   1])
```

Array à 2-dimension (Liste de liste)

```
>>> w = np.array ([[1 ,2] ,[3 ,4] ,[5 ,6]])
```

Array à 3-dimensions (liste de liste de liste)

```
>>> x = np.array ([[[1 ,2] ,[2 ,3]] ,[[4 ,5] ,[5 ,6]]])
```

Pour connaitre la dimension et la taille selon chaque dimension

```
>>> w = np.array ([[1 ,2] ,[3 ,4] ,[5 ,6]])
>>> w.ndim
2
>>> w.shape
(3,2)
```

La méthode.reshape() modifie les dimensions d'un array:

```
>>> a = np.arange(0, 6)
>>> a.shape
(6,)
>>> b = a.reshape ((2, 3))
>>> b.shape
(2, 3)
```

Matrice qui contient des zéros partout ou des 1 :

```
>>> np.zeros ((2, 3)) # matrice 2x3 0 partout
>>> np.ones((3, 3))   # matrice 3x3 1 partout
```

Transposé

```
np.transpose(a)  # transposée de la matrice a
```

Produit de matrice

```
np.dot(a, a)   # produit matricielle
a * a          # produit élément par élément
```

Inversion et valeurs propres d'une matrice

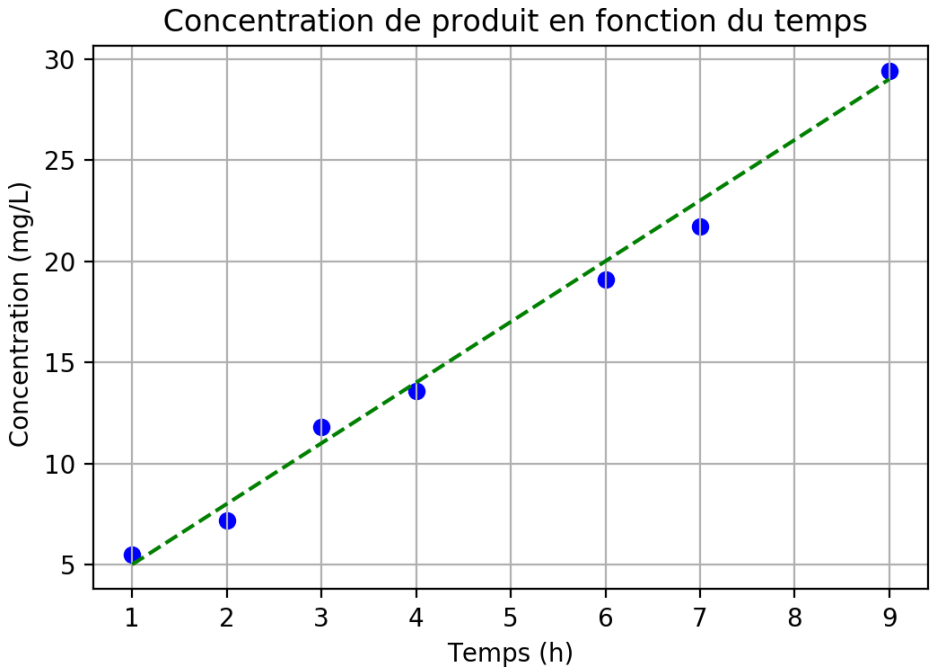
```
np.linalg.inv(a)  # inverse d'une matrice
np.linalg.eig(a)  # valeurs et vecteurs propres d'une matrice
```

12. La bibliothèque Matplotlib

Le package matplotlib permet de créer des graphiques. Voici un exemple

```
# Creer un graphique representant concentration en fonction du temps
import numpy as np
import matplotlib.pyplot as plt  # importer pyplot
temps = [1, 2, 3, 4, 6, 7, 9]
concentration = [5.5, 7.2, 11.8, 13.6, 19.1, 21.7, 29.4]
plt.scatter(temps , concentration , marker ="o", color ="blue")
plt.xlabel ("Temps (h)")
plt.ylabel (" Concentration (mg/L)")
plt.title(" Concentration de produit en fonction du temps")

#Plot de la courbe y=2+3x et la sauvegarder dans un fichier
x = np.linspace(min(temps), max(temps), 50)
y = 2 + 3 * x
plt.plot(x, y, color='green', ls="--")
plt.grid()
plt.savefig('concentration_vs_temps.png', bbox_inches='tight', dpi =200)
plt.show()
```



13. La bibliothèque pandas

Le package pandas a été conçu pour la manipulation et l'analyse de données. Il est particulièrement puissant pour manipuler des données structurées sous forme de tableau. Pandas introduit le type dataframe. C'est un tableau à deux dimensions avec des étiquettes pour nommer les lignes et les colonnes.

Pandas permet en particulier de lire des fichiers excel et CSV :

```
import numpy as np
import pandas as pd

# reading excel file with pandas
df = pd.read_excel (r'./Exemple1.xls') # df est de type dataframe
df

df = pd.read_excel (r'./Exemple1.xls',index_col=0) #1ere col contient les noms d ech
df

#get specific columns
Zn = pd.DataFrame(df, columns=['a','c']) # récupérer les 2 colonnes 'a' et 'c'

# reading CSV file with pandas
df = pd.read_csv (r'./Exemple2.csv',index_col=0)
shape = df.shape
```

Exercice

Dans le code précédent, afficher et donner la différence entre les deux commandes suivantes :

```
df = pd.read_excel (r'./Exemple1.xls') # df est de type dataframe
df = pd.read_excel (r'./Exemple1.xls',index_col=0) #1ere col contient les noms d ech
```