



Gestion de Projet Big Data & Développement d'applications Big Data

EDAH Kodjo
Consultant Systèmes d'Information, Big-Data

Objectifs

- Comprendre la notion et les spécificités du Big Data
- Connaître les technologies de l'écosystème Hadoop
- Connaître le langage python et utiliser les librairies de machine learning
- Savoir utiliser les outils de visualisation des données (Dataviz)



Partie 3 : Python

Les bases de python

- ☐ Présentation de Python
- ☐ Les types et les opérations de base
- ☐ Les structures de contrôle
- ☐ Les fonctions
- ☐ Les fichiers
- ☐ Les classes
- ☐ Les exceptions
- ☐ Les modules



Présentation de Python

- Développé en 1989 par Guido van Rossum
- Open source
- Portable (Windows, linux Mac OS)
- Orienté objet
- Dynamique
- Extensible
- Support pour l'intégration d'autre langage



Présentation de Python

- Langage de programmation généraliste, interprété et open source
- Version 2.7 (2012) et 3.8 (2019)
- Multitudes de frameworks et packages:
 - Web: Django, flask
 - Calcul scientifique: Numpy, Scipy
 - Analyse de données et machine learning: pandas, scikit-learn
- Langage pour application Data/Analytics par excellence



Outils

❑ Distribution Anaconda

- 100+ Packages inclus
- Gestion d'environnement virtuel (reproduction)



❑ IPython (Shell) et Projet Jupyter

- Exécution interactive
- Visualisation intégrée
- Support de Markdown
- Sauvegarde de l'état à travers un Kernel
- Exportation en HTML



IP[y]:
IPython



Affichage : la fonction print

❑ La fonction print()



```
print("Hello world!")
```



```
Hello world!
```

❑ Écriture formatée



```
x = 32
```

```
nom = "John"
```

```
print("{} a {} ans".format(nom, x))
```



```
John a 32 ans
```



Variables

- ❑ Commence avec A-Z a-z ou _
- ❑ Autres caractères lettres, chiffres, ou _
- ❑ Sensible à la casse
- ❑ Pas de longueur précise (raisonnablement)
- ❑ Pas les mots réservés au langage
- ❑ Assignations :
 $\text{<nom_variable> = <expression>}$
- ❑ Convention de nommage : **snake_case**

```
▶ age = 10  
print(age)
```

```
📄 10
```

```
▶ age, nom_etudiant, prenom_etudiant = 10, "Hubert", "Charles"  
print(age, nom_etudiant, prenom_etudiant)
```

```
📄 10 Hubert Charles
```



Structure de données

- ❑ Toutes les valeurs en Python sont des objets
- ❑ Nombres : int, long, float, complex
- ❑ Booléen : bool
- ❑ String



```
age = 10
string_var = "Mon texte"
print(type(age))
print(type(string_var))
```



```
<class 'int'>
<class 'str'>
```



Les opérations

- ❑ Arithmétique : +, -, *, /, %, **, //
- ❑ Comparaison : ==, !=, >, <, >=, <=
- ❑ Logique : and, or, not
- ❑ Assignations: +=, -=, /=, *=, %=, **=, //=
- ❑ Bitwise: &, |, ^, ~, >>, <<



```
age = 10  
ma_chaine= "Iris school"  
print(age >= 5 + 5 )  
print(len(ma_chaine) > 25 / 2 )
```



```
True  
False
```



Structure de contrôle - if else

```
if <expr>:  
    <statement(s)>  
elif <expr>:  
    <statement(s)>  
elif <expr>:  
    <statement(s)>  
    ...  
else:  
    <statement(s)>
```

```
nom_etudiant = 'John'  
if (nom_etudiant == 'Fred'):  
    print('Hello Fred')  
elif (nom_etudiant == 'Xander'):  
    print('Hello Xander')  
else :  
    print('Qui es-tu ?')
```

❑ Attention à indentation



Structure de contrôle boucle - for

- C'est une itération sur une séquence d'éléments: iterator
- on peut utiliser break pour interrompre
- on peut utiliser continue pour ignorer la suite de la boucle

```
▶ nom_etudiant = "Fredy"  
for lettre in nom_etudiant:  
    print(lettre)
```

```
↳ F  
   r  
   e  
   d  
   y
```

```
▶ nom_etudiant = "Fredy"  
for lettre in nom_etudiant:  
    print(lettre)  
    if(lettre == "e"):  
        break
```

```
↳ F  
   r  
   e
```

```
▶ nom_etudiant = "Fredy"  
for lettre in nom_etudiant:  
    if(lettre == "e"):  
        continue  
    print(lettre)
```

```
↳ F  
   r  
   d  
   y
```



Structures de contrôle - while

```
while <expr> :  
    <statement(s)>
```



```
i = 0  
while i < 3:  
    print("i =" , i)  
    i = i + 1
```



```
i = 0  
i = 1  
i = 2
```



Listes

- ❑ Une liste est une structure de données qui contient une série de valeurs
- ❑ Python autorise la construction de liste contenant des valeurs de types différents
Accès par indice
- ❑ Concaténation simple avec “+”
- ❑ `append(x)`, `remove(x)`, `insert(i,x)`
- ❑ Les fonctions `range()` et `list()`

```
▶ list(range(10))
```

```
↳ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
▶ liste_present=["Charles","Carine"]  
liste_absent=["Kodjo","Victor", "Charles"]  
liste_total=liste_present + liste_absent  
print(liste_total)  
print(liste_total[3])
```

```
↳ ['Charles', 'Carine', 'Kodjo', 'Victor', 'Charles']  
Victor
```

```
liste_poly=["Charles","Carine",12,True]  
print(liste_poly[3])  
print(liste_poly[5])
```

True

IndexError Traceback (most recent call last)

```
<ipython-input-30-2d233b6f98a0> in <module>()  
    1 liste_poly=["Charles","Carine",12,True]  
    2 print(liste_poly[3])  
----> 3 print(liste_poly[5])
```

IndexError: list index out of range

Les tuples

- ❑ Un tuple est une structure de données qui contient une série de valeurs et qui reste immuable après la création
- ❑ Les tuples peuvent contenir différents types de valeurs
- ❑ Comme les listes on accède aux éléments avec l'index

```
# La liste vide se créé avec []  
# Le tuple vide se créé avec ()  
emptyTuple = ()
```

```
# Tuple avec des éléments  
etudiant_present = ("John",  
                    "Sarah",  
                    {"nom" : "Ahmed", "present" : True},  
                    ["M1-Iris", "L3-Iris"])  
  
for elt in etudiant_present :  
    print(elt)
```



Déballage de séquence (Sequence unpacking)

```
#Déballage de séquence
facture = ("Lait", 2, 100, "Leclerc");
produit, quantite, prix, magasin = facture

print("* Le produit {} est acheté à {}".format(facture[0], facture[3]))
print("*** Le produit {} est acheté à {}".format(produit, magasin))
```

```
* Le produit Lait est acheté à Leclerc
*** Le produit Lait est acheté à Leclerc
```



Dictionnaires

- Ensemble non ordonné de paires clés,valeurs
- Les objets sont accessibles via leur clé (pas de notion d'indice)
- Le type de clé permis : String, Nombre
- tuple (non modifiable)
- Les clés sont hachées => lecture et recherche en $O(1)$

```
dict_1 = dict()  
dict_1["nom"]="Iris"  
dict_1["age"]=16  
dict_1
```

```
↳ {'age': 16, 'nom': 'Iris'}
```

```
dict_2 = {"nom" : "Iris", "age" : 16}  
  
dict_2
```

```
↳ {'age': 16, 'nom': 'Iris'}
```



Fonctions

```
def nom_fonction( arguments ):  
    """  
    La description de ma fonction  
    """  
    expression  
    ....  
  
    return [expression]
```

```
▶ def afficheur(argument1,argument2):  
    """  
    This is the second line of the docstring.  
    """  
    print(argument1)  
    return argument2  
  
print(afficheur("Nom","John"))
```

☞ Nom
John



Bon sens

- ❑ 4 espaces pour l'indentation (pas de tab)
- ❑ Ne pas mixer les tabs et les espaces
- ❑ Longueur maximale d'une ligne fixée à 79 caractères
- ❑ Sauter plus qu'une ligne pour séparer les fonctions de haut niveau et une seule ligne pour séparer les méthodes dans une classe
- ❑ Quand c'est possible, ajouter des commentaires sur une seule ligne
- ❑ Utiliser des espaces entre une expression et les déclarations



Quelques règles



```
import this
"""The Zen of Python, by Tim Peters. (poster by Joachim Jablon)"""

1 Beautiful is better than ugly.
2 Explicit is better than implicit.
3 Simple is better than complex.
4 Complex is better than complicated.
5 Flat is better than nested.
6 Sparse is better than dense.
7 Readability counts.
8 Special cases aren't special enough to break the rules.
9 Although practicality beats purity.
10 raise PythonicError("Errors should never pass silently.")
11 # Unless explicitly silenced.
12 In the face of ambiguity, refuse the temptation to guess.
13 There should be one-- and preferably only one --obvious way to do it.
14 # Although that way may not be obvious at first unless you're Dutch.
15 Now is better than ... never.
16 Although never is often better than right now.
17 If the implementation is hard to explain, it's a bad idea.
18 If the implementation is easy to explain, it may be a good idea.
19 Namespaces are one honking great idea -- let's do more of those!
```

POO : programmation orientée objet

❑ Python est un langage permettant la programmation orientée objet

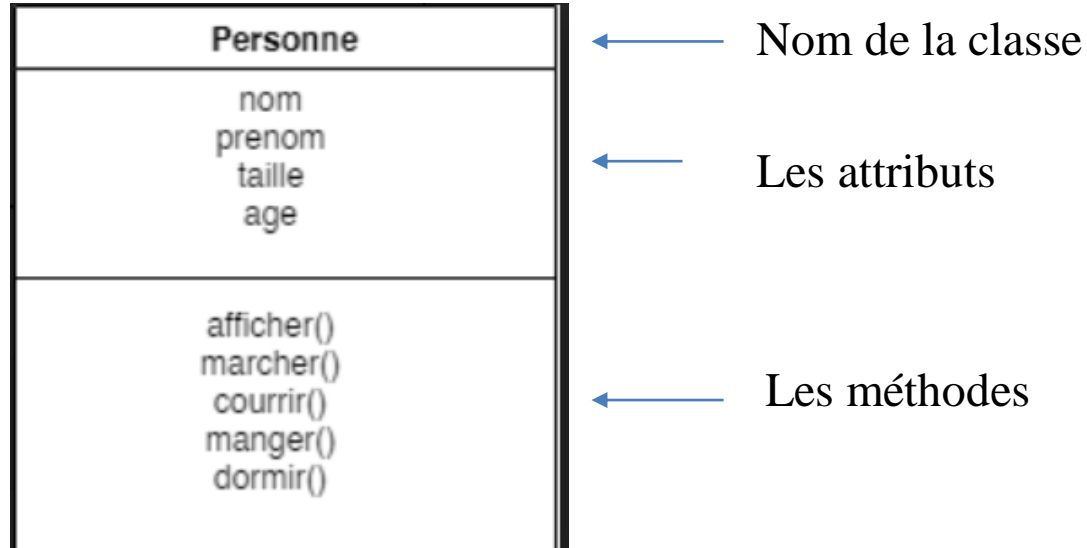
❑ Rappel

- ❑ Encapsulation
- ❑ Héritage
- ❑ Polymorphisme
- ❑ Composition
- ❑ Classe, Object, variable/méthode de classe, attributs, méthodes



POO : programmation orientée object

❑ Classe



POO : programmation orientée object

❑ Classe

```
class Personne:
    "Définition d'une personne"

    # Constructor
    def __init__(self, nom, prenom, age = 18, taille=170):
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.taille = taille

    # Afficher une personne
    def afficher(self):
        print("Nom : {} , Prenom {}, age = {}, taille {}".format(self.nom, self.prenom, self.age, self.taille))
```



POO : programmation orientée object

- ❑ Objet : instance de classe

```
etudiant = Personne("John", "Doe", taille=175)  
etudiant.afficher()
```

```
Nom : John , Prenom Doe, age = 18, taille 175
```



POO : programmation orientée object

❑ Héritage

```
class Etudiant(Personne):  
  
    def __init__(self,nom, prenom, univ, age = 18, taille=170):  
        Personne.__init__(self, nom, prenom, age, taille )  
        self.univ = univ  
  
    def afficher(self):  
        super().afficher()  
        print("L'universite de l 'etudiant est {}".format(self.univ))
```



POO : programmation orientée object

❑ Héritage

```
etudiant = Etudiant("Jobn", "Doe", "Iris")
```

```
etudiant.afficher()
```

```
print(isinstance(etudiant, Etudiant))
```

```
print(isinstance(etudiant, Personne))
```

```
print(issubclass(Personne, Etudiant))
```

```
print(issubclass(Etudiant, Personne))
```

Nom : Jobn , Prenom Doe, age = 18, taille 170

L'universite de l 'etudiant est Iris

True

True

False

True



POO : programmation orientée object

❑ Attributs de classe

```
class Etudiant():
    listUniv = []
    def __init__(self,nom):
        self.nom = nom
        self.etudList = []

    def addUniv(self, univ):
        Etudiant.listUniv.append(univ)
        self.etudList.append(univ)
```

```
etudiantLicence = Etudiant("John")
etudiantLicence.addUniv("Sorbonne")
```

```
print(etudiantLicence.etudList)
print(Etudiant.listUniv)
```

```
etudiantMaster = Etudiant("Sarah")
etudiantMaster.addUniv("Iris")
```

```
print(etudiantMaster.etudList)
print(Etudiant.listUniv)
```

```
['Sorbonne']
['Sorbonne']
['Iris']
['Sorbonne', 'Iris']
```



POO : programmation orientée object

❑ Getter et Setter

❑ Utilisation

- ❑ Ajouter une logique de validation autour de l'obtention et de la définition d'une valeur.
- ❑ Eviter l'accès direct à un champ de classe, c'est-à-dire que les variables privées ne peuvent pas être consultées directement ou modifiées par un utilisateur externe.

```
class Personne:
    "Définition d'une personne"

    # Constructor
    def __init__(self):
        self._nom = ""

    # Normal get_nom method
    def get_nom(self):
        print("Appel de la fonction get_nom")
        return self._nom

    # Normal set_nom method
    def set_nom(self, nom):
        print("Appel de la fonction set_nom ")
        self._nom = nom
```

```
personne = Personne()
```

```
personne.set_nom("John Doe")
personne.get_nom()
```

```
Appel de la fonction set_nom
Appel de la fonction get_nom
'John Doe'
```



POO : programmation orientée object

❑ Getter et Setter

❑ property() function

```
class Personne:
    # Constructor
    def __init__(self):
        self._nom = ""

    # get_nom method
    def get_nom(self):
        print("Appel de la fonction get_nom ")
        return self._nom

    # set_nom method
    def set_nom(self, nom):
        print("Appel de la fonction set_nom ")
        self._nom = nom

    # delete _nom
    def del_nom(self):
        del self._nom

    nom = property(get_nom, set_nom, del_nom)
```

```
personne = Personne()
personne.set_nom("John Doe")
print(personne.nom)
```

Appel de la fonction set_nom
Appel de la fonction get_nom
John Doe



POO : programmation orientée object

- ❑ Getter et Setter
- ❑ @property decorators

```
class Personne:
    # Constructor
    def __init__(self):
        self._nom = ""

    # get_nom method
    @property
    def nom(self):
        print("Appel de la fonction get_nom ")
        return self._nom

    # set_nom method
    @nom.setter
    def nom(self, nom):
        print("Appel de la fonction set_nom ")
        self._nom = nom

    # delete _nom
    @nom.deleter
    def del_nom(self):
        del self._nom
```

```
personne = Personne()
personne.nom = "John Doe"
print(personne.nom)
```

Appeller la fonction set_nom
Appeller la fonction get_nom
John Doe



Les librairies data science



NumPy

- ❑ Numerical Python
- ❑ Stocker et effectuer des opérations sur les données sous forme de tableaux
- ❑ Une librairie principale de l'écosystème Python pour la data science
- ❑ Les tableaux peuvent contenir les données que d'un seul type (à la différence des listes)
- ❑ <https://numpy.org/doc/stable/user/quickstart.html>



```
import numpy as np
```



❑ Créer un tableau : détection automatique des types



```
# Création d'un tableau
# Détection auto des entiers
arr_entier = np.array([5, 2, 50])

arr_float = np.array([5.2, 2, 0.2])

print(arr_entier.dtype)
print(arr_float.dtype)
```

```
int64
float64
```

❑ Créer un tableau : précision du type



```
# Précision du type
arr_precise_type = np.array([5, 2, 50], dtype='double')

print(arr_precise_type.dtype)
```

```
float64
```



❑ Les fonctions usuelles

```
# Créer un tableau rempli de zéros entiers
print("Un tableau rempli de zéros entiers")
zeros = np.zeros(5)
print(zeros)

# une matrice rempli 2x5 composée de 1
print("Une matrice rempli 2x4 composée de 1")
ones = np.ones((2, 4))
print(ones)

# Remplir un tableau multi dimensionnel
print("Remplir un tableau multi dimensionnel ")
full = np.full((2, 4), 6, dtype='float32')
print(full)
```

```
# Un tableau rempli d'une séquence linéaire
print("Tableau de séquence linéaire")
arange_array = np.arange(1, 10, 3)
print(arange_array)

# Créer un tableau contenant des valeurs aléatoires
print("Tableau conteannt des valeurs aléatoires")
alea = np.random.random((2, 4))
print(alea)
```



❑ Les fonctions usuelles

```
# Créer un tableau rempli de zéros entiers
print("Un tableau rempli de zéros entiers")
zeros = np.zeros(5)
print(zeros)

# une matrice rempli 2x5 composée de 1
print("Une matrice rempli 2x4 composée de 1")
ones = np.ones((2, 4))
print(ones)

# Remplir un tableau multi dimensionnel
print("Remplir un tableau multi dimensionnel ")
full = np.full((2, 4), 6, dtype='float32')
print(full)
```

```
# Un tableau rempli d'une séquence linéaire
print("Tableau de séquence linéaire")
arange_array = np.arange(1, 10, 3)
print(arange_array)
```

```
# Créer un tableau contenant des valeurs aléatoires
print("Tableau conteannt des valeurs aléatoires")
alea = np.random.random((2, 4))
print(alea)
```

❑ *ndim, size, dtype*



❑ Récupération et manipulation

```
my_arr = np.array([[1,2,3],[5,8,9],[10, 2, 0]])

print(my_arr)

# Récuper la première ligne
print(my_arr[0])

# Récuper l'élément à l'index 0, 2
print(my_arr[0][2])

# Modifier la valeur de se l'élément à l'index 2,2
my_arr[2][2] = 2000

# Afficher le dernier élément
print(my_arr[-1])
```

❑ slicing

```
my_multi_arr = np.array([[1,2,3],
                          [5,8,9],
                          [10,2,0]])

#Récuper la deuxième colonne
cols = my_multi_arr[:,1]
print(cols)
#ou
my_multi_arr[:,1:2]

#Que remarquez vous ?

[2 8 2]
array([[2],
       [8],
       [2]])
```



❑ Pourquoi préférer les tableaux NumPy ?

```
def multiply_by(values):  
    i = 0  
    while i < len(values) :  
        values[i] = 2*values[i]  
        i = i + 1  
    return values  
  
values = np.random.randint(1, 10, size=5)  
print("Valeurs initiales")  
print(values)  
print("Normal python array multiplication")  
print(multiply_by(values.copy()))  
print("Numpy multiplication ")  
print(2* values.copy())
```

```
# Effectuer les mêmes actions sur des tableaux plus grand  
# en mesurant le temps d'exécution  
#perf_arr = np.ones(1000000)  
perf_arr = np.random.randint(1, 10, size=1000000)  
%timeit multiply_by(perf_arr.copy())  
  
%timeit (1 * perf_arr.copy())
```

1 loop, best of 5: 672 ms per loop
1000 loops, best of 5: 1.46 ms per loop

- ❑ L'accès à la lecture et à l'écriture d'éléments plus rapide
- ❑ Faible occupation de la mémoire



Matplotlib

- ❑ Tracer et visualiser des données sous formes de graphiques
- ❑ Export possible en de nombreux formats matriciels (PNG, JPEG...) et vectoriels (PDF, SVG...)
- ❑ Documentation en ligne en quantité, nombreux exemples disponibles sur internet
- ❑ Forte communauté très active
- ❑ Bibliothèque haut niveau : idéale pour le calcul interactif

```
#Importer les librairies
import numpy as np
import matplotlib.pyplot as plt
```



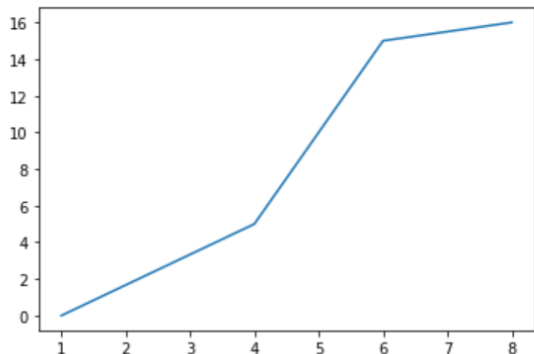
Matplotlib

❑ Exemple : plot

```
# Un exemple de graphique
```

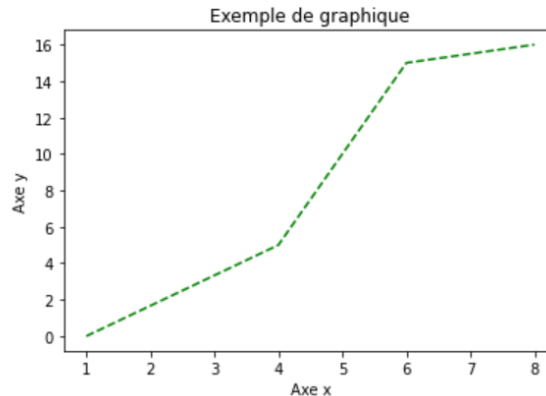
```
x = np.array([1, 4, 5, 6, 8])  
y = np.array([0, 5, 10, 15, 16])  
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x7fd9bd654c50>]
```



```
# la fonction plot  
# https://matplotlib.org/2.1.1/api/\_as\_gen/matplotlib.pyplot.plot.html  
x = np.array([1, 4, 5, 6, 8])  
y = np.array([0, 5, 10, 15, 16])  
plt.xlabel("Axe x")  
plt.ylabel("Axe y")  
plt.title("Exemple de graphique")  
plt.plot(x, y, 'g--')
```

```
[<matplotlib.lines.Line2D at 0x7fd9bd23cf10>]
```



❑ A découvrir : https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

Matplotlib

❑ Exemple : sub plot

```
# sub plots

x = np.array([1, 4, 5, 6, 8])
y = np.array([0, 5, 10, 15, 16])

fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('Exemple des sous graphes')

ax1.plot(x, y, 'r+')

ax2.plot(x, y, 'g-')
ax2.set_xlabel('Axe x')
ax2.set_ylabel('Axe y')

plt.show()
```

❑ https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplot.html



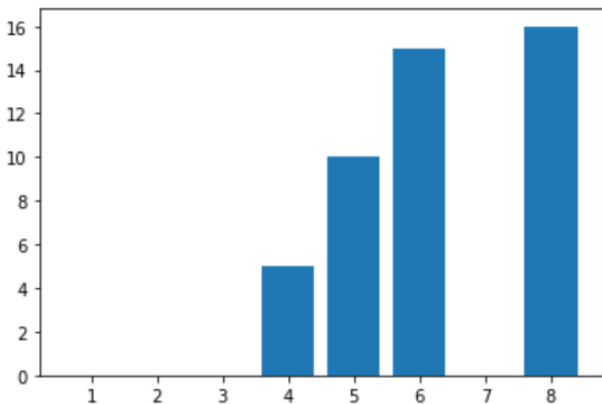
Matplotlib

❑ Exemple : bar diagram (histogramme)

```
# bar
```

```
x = np.array([1, 4, 5, 6, 8])  
y = np.array([0, 5, 10, 15, 16])  
plt.bar(x, y)
```

<BarContainer object of 5 artists>



❑ https://matplotlib.org/stable/gallery/lines_bars_and_markers/bar_label_demo.html



Matplotlib

❑ Exemple : Scatter plot (nuage de point)

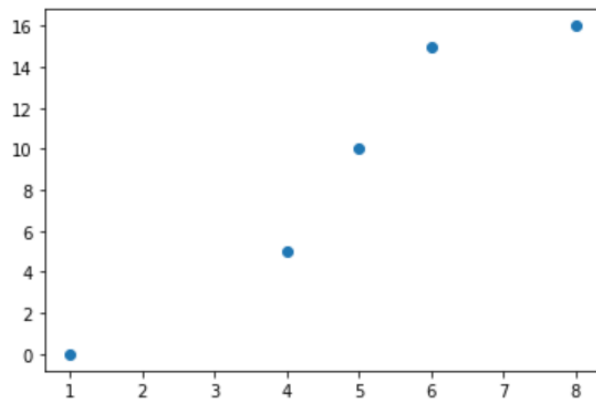
```
# scatter
```

```
x = np.array([1, 4, 5, 6, 8])
```

```
y = np.array([0, 5, 10, 15, 16])
```

```
plt.scatter(x, y)
```

```
<matplotlib.collections.PathCollection at 0x7fd9c4479350>
```



Matplotlib et seaborn

- ❑ Seaborn est une bibliothèque permettant de créer des graphiques statistiques en Python. Il s'appuie sur matplotlib et s'intègre étroitement aux structures de données pandas.

- ❑ <https://seaborn.pydata.org/>



Pandas

- ❑ Pandas est une librairie python qui permet de manipuler facilement des données à analyser (des tableaux de données)

```
s = pandas.Series([1, 2, 5, 7]) : série numérique entière.  
s = pandas.Series([1.3, 2, 5.3, 7]) : série numérique flottante.  
s = pandas.Series([1, 2, 5, 7], dtype = float) : série numérique flottante
```

- ❑ On peut utiliser comme type les types numpy, par exemple :

```
s = pandas.Series([1, 2, 5, 7], dtype = numpy.int8)
```

- ❑ Pandas offre plus de possibilité et de facilité de manipulation des tableaux par rapport à numpy



Pandas : dataFrames

- ❑ Un dataframe se comporte comme un dictionnaire dont les clés sont les noms des colonnes et les valeurs sont des séries.

- ❑ En ligne : les notes d'un élève

- ❑ En colonne les matières

- ❑ Pandas : utilisation des index et columns



```
import numpy as np
import pandas as pd

note_etudiants = [[10,18,15],[4,5,19], [14, 16, 18]]

pd.DataFrame(note_etudiants)
```

0 1 2

0 10 18 15

1 4 5 19

2 14 16 18



Pandas : dataFrames

- ❑ Un dataframe se comporte comme un dictionnaire dont les clés sont les noms des colonnes et les valeurs sont des séries.

- ❑ En ligne : les notes d'un élève

- ❑ En colonne les matières


- ❑ Pandas : utilisation des index et columns

```
import numpy as np
import pandas as pd

note_etudiants = [[10,18,15],[4,5,19], [14, 16, 18]]

note_df = pd.DataFrame(note_etudiants, index=["Aymen", "Marie", "Ana"],
                        columns=["Big Data", "Reseaux", "Gestion projet"]
                        )

note_df
```



	Big Data	Reseaux	Gestion projet
Aymen	10	18	15
Marie	4	5	19
Ana	14	16	18

Pandas : dataFrames

❑ Récupérer les éléments d'une colonne

```
# Afficher les notes big Data

big_data_series = note_df["Big Data"]

big_data_series

Aymen      10
Marie       4
Ana        14
Name: Big Data, dtype: int64
```

❑ Boucler sur les lignes

```
#boucler sur les lignes
for index_ligne, ligne in note_df.iterrows():
    print("Les notes de l'étudiant {}".format(index_ligne))
    print(ligne.values)

Les notes de l'étudiant Aymen
[10 18 15]
Les notes de l'étudiant Marie
[ 4  5 19]
Les notes de l'étudiant Ana
[14 16 18]
```



Pandas : dataFrames

❑ Récupérer les éléments d'une ligne

```
# récupérer les éléments d'une ligne
# Exemple Ana

# Avec loc(), indexation par label
note_df.loc["Ana"]
# Avec iloc(), indexation positionnelle
note_df.iloc[2]
```

```
Big Data      14
Reseaux       16
Gestion projet 18
Name: Ana, dtype: int64
```

❑ Filtrer

```
# Les étudiants n'ayant pas la moyenne en big data

bonne_note_big_data = note_df[note_df["Big Data"] >= 10]
bonne_note_big_data
```

	Big Data	Reseaux	Gestion projet
Aymen	10	18	15
Ana	14	16	18



	Big Data	Reseaux	Gestion projet
Aymen	10	18	15
Marie	4	5	19
Ana	14	16	18



Pandas : dataFrames

❑ Ajouter une nouvelle matière

```
#Ajouter une nouvelle matière  
  
note_df["Anglais"] = [15, 11, 10]  
  
note_df
```

	Big Data	Reseaux	Gestion projet	Anglais
Aymen	10	18	15	15
Marie	4	5	19	11
Ana	14	16	18	10

❑ Filtrer

```
# Les étudiants n'ayant pas la moyenne en big data  
  
bonne_note_big_data = note_df[note_df["Big Data"] >= 10]  
  
bonne_note_big_data
```

	Big Data	Reseaux	Gestion projet
Aymen	10	18	15
Ana	14	16	18

	Big Data	Reseaux	Gestion projet
Aymen	10	18	15
Marie	4	5	19
Ana	14	16	18



Pandas : dataFrames

❑ Ajouter de nouveaux étudiants

```
nouveaux_etudiants = pd.DataFrame( [[2, 5, 8, 10],  
                                     [14,8,12, 100],  
                                     [-12, 15, 17, 21]],  
                                   index=["Mathias", "Christelle", "Georges"],  
                                   columns=["Big Data", "Reseaux", "Gestion projet", "Anglais"]  
                                   )  
note_df = note_df.append(nouveaux_etudiants)  
  
note_df
```

	Big Data	Reseaux	Gestion projet	Anglais
Aymen	10	18	15	15
Marie	4	5	19	11
Ana	14	16	18	10
Mathias	2	5	8	10
Christelle	14	8	12	100
Georges	-12	15	17	21

❑ Fonctions usuelles

```
#Fonctions usuelles sur dataframe  
note_df.describe()
```

	Big Data	Reseaux	Gestion projet	Anglais
count	24.000000	24.000000	24.000000	24.000000
mean	2.333333	9.791667	12.958333	39.708333
std	10.610359	4.671646	3.950555	39.777074
min	-12.000000	5.000000	8.000000	10.000000
25%	-12.000000	5.000000	8.000000	10.000000
50%	2.000000	8.000000	12.000000	21.000000
75%	14.000000	15.000000	17.000000	100.000000
max	14.000000	18.000000	19.000000	100.000000

Pandas : dataFrames

- ❑ Fonctions usuelles

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>



Pandas : dataFrames

- ❑ On peut facilement lire et écrire ces dataframes à partir ou vers un fichier.

```
df = pandas.read_csv('chemin_de_mon_fichier.csv')
```



```
df.to_csv('new_file.csv', sep = '\t')
```



Pandas

- ❑ On peut facilement tracer des graphes à partir de ces DataFrames grâce à matplotlib.

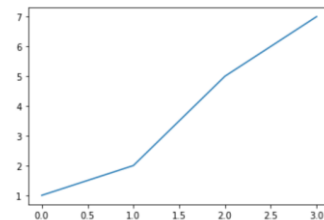
```
DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False, sharex=None, sharey=False, layout=None,
figsize=None, use_index=True, title=None, grid=None, legend=True, style=None, logx=False, logy=False,
loglog=False, xticks=None, yticks=None, xlim=None, ylim=None, rot=None, fontsize=None, colormap=None,
table=False, yerr=None, xerr=None, secondary_y=False, sort_columns=False, **kwds) ¶ \[source\]
```

Make plots of DataFrame using matplotlib / pylab.

- ❑ Pour utiliser pandas : import pandas

```
In [16]: s = pd.Series([1, 2, 5, 7])
s.plot()
```

```
Out[16]: <AxesSubplot:>
```



Questions ?

Merci

