

Algorithmique et Programmation avancée en Python



- **Année Académique:**

2022/2023

- **Niveau:**

CIA3 – ISE L3 : Cycle , Première Année

Filière : Ingénieur Statisticien Economiste Cycle long

Institut Sous-régional de Statistique et d'Economie Appliquée

- **Enseignant:**

Dr Wansouwé Wanbitching



Algorithmique et Programmation avancée en Python

- Intitulé : **Algorithmique et Programmation avancée en Python**
 - Code :
 - Cours magistral : 10 heures
 - Travaux dirigés / pratiques : 10 heures
- Éléments constitutifs de l'Unité d'Enseignement
 - *Cours magistral* : soyez attentifs !
 - *Travaux Dirigés* : Exercices
 - *Travaux Pratiques* : le plus intensif possible
 - *Travail Personnel encadré* : Soyez créatifs et appliqués

Organisation de l'UE



- **Types d'évaluation**
 - *TPE : Travail à faire à la maison*
 - *CC : 3 heures*
 - *Examen : 3 heures*
 - *Rattrapage : 3 heures*
- **Organisation des évaluations**
 - Exercices et Travaux Pratiques
 - Questions de cours
 - Parfois des QCM
 - Recherches personnelles

Pourquoi ce cours ?



- Le cours vise à donner aux futurs statisticiens les concepts avancés de l'algorithmique et de la programmation en Python, langage devenu incontournable pour la manipulation des données.

Plan



Chapitre	Intitulé
1	Notions de complexité algorithmique
2	Rappels de quelques concepts fondamentaux de Python
3	Introduction aux modules de webscraping
4	Gestion des Fichiers

Bibliographie



- Thomas Cormen, Charles Leiserson, Ronald Rivest : **Introduction to algorithms**, MIT press 1990
- Sara Baase, Allen van Gelder: **Computer algorithms: introduction to design and Analysis**, Addison Wesley, 2000
- David Brunsell, John Turner: **Understanding Algorithms and Data Structures**, Mc Graw Hill, 1996
- Alfred Aho, Jeffrey Ullman : **Concepts fondamentaux de l'informatique**, Dunod, 1993
- Swinnen, G. (2012). **Apprendre à programmer avec Python 3**, 3e Edition, Eyrolles



Chapitre 2 : Rappels des fondamentaux en Python

Objectifs



- Réviser les bases du langage Python : opérateurs, structures de données, etc.
- Se familiariser avec les bibliothèques de calcul scientifique Numpy, Pandas, Matplotlib et Seaborn en Python.
- Utiliser des fonctions de ces bibliothèques pour la manipulation et la visualisation des données.



Python

- Python est un langage de programmation interprété, multi-paradigme et multiplateforme.
- Il est créé par le néerlandais **Guido van Rossum** en **1989**
- Il s'adapte à tout type d'utilisation grâce à des **bibliothèques spécialisées**.
- En plus, il est particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses comme la collecte de données disponibles sur Internet.



Python 3 vs Python 2

- La version en vigueur est Python 3 car Python 2 n'est plus supporté depuis le 1.1.2019
- Python est le langage le plus populaire à l'ère actuelle pour le calcul scientifique et la science des données, grâce à des bibliothèques telles que numpy, scipy, matplotlib. Il est suivi par R.
- Guide de style pour la programmation en Python PEP8 (<https://www.python.org/dev/peps/pep-0008/>)

Python et les types de données



- Python utilise un *typage dynamique* des variables, c'est-à-dire que le type de donnée est “décidé” par l'ordinateur “à la volée.”
- Python possède un certain nombre de types de données de base, notamment des entiers (**int**), des nombres à virgule flottante (**float**), des booléens (**boolean**) et des chaînes de caractères (**string**).

Opérateur logiques



```
t = True  
f = False  
  
print(type(t))      # Prints "<class 'bool'> "  
print(t and f)     # Logical AND; prints "False"  
print(t or f)      # Logical OR; prints "True"  
print(not t)        # Logical NOT; prints "False"  
print(t != f)       # Logical XOR; prints "True"
```

Chaînes de caractères



```
mot1 = 'hello'      # String literals can use single quotes  
mot2 = "world"      # or double quotes; it does not matter.
```

```
print(mot1) # Prints "hello"  
print(len(mot1)) # String length; prints "5"
```

```
'h' in mot1 # Prints True; 'h' is part of mot1  
print(mot1[0])      # Prints 'h'; first character of mot1
```

```
mot3 = mot1 + ' ' + mot2 # String concatenation  
print(mot3)            # prints "hello world"
```

```
print(mot1 * 3) # prints 'hellohellohello'  
hw12 = '%s %s %d' % (mot1, mot2, 12) # style string formatting  
print(hw12) # prints "hello world 12"
```

Chaînes de caractères



```
s = "hello"  
• print(s.replace('h','c'))  
# Prints 'cello'  
  
• print(s.capitalize())  
# Capitalize a string; prints "Hello"  
  
• print(s.upper())  
# Convert a string to uppercase; prints "HELLO"  
  
• print(s.center(7))  
# Center a string, padding with spaces; prints " hello "  
  
• print(' world '.strip())  
# Strip leading and trailing whitespace; prints "world"
```

Chaînes de caractères



```
s = "hello"  
• print(s.replace('h','c'))  
# Prints 'cello'  
  
• print(s.capitalize())  
# Capitalize a string; prints "Hello"  
  
• print(s.upper())  
# Convert a string to uppercase; prints "HELLO"  
  
• print(s.center(7))  
# Center a string, padding with spaces; prints " hello "  
  
• print(' world '.strip())  
# Strip leading and trailing whitespace; prints "world"
```

Accès aux éléments d'une liste



```
nums = list(range(5)) # range is a built-in function

print(nums)          # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])    # Get a slice from index 2 to 4
                    #(exclusive); prints "[2, 3]"
print(nums[2:])     # Get a slice from index 2 to the
                    # end; prints "[2, 3, 4]"
print(nums[:2])     # Get a slice from the start to
                    # index 2 (exclusive); prints "[0, 1]"
print(nums[:])      # Get a slice of the whole list;
                    # prints "[0, 1, 2, 3, 4]"
print(nums[:-1])   # Slice indices can be negative;
                    # prints "[0, 1, 2, 3]"

nums[2:4] = [8, 9] # Assign a new sublist to a slice
print(nums)         # Prints "[0, 1, 8, 9, 4]"

nums[::-1]          # inverse l'ordre de la liste
```



Liste

- Une liste est similaire à un tableau en Python, mais elle est redimensionnable et peut contenir des éléments de différents types.

```
xs = [3, 1, 2]                      # Create a list

print(xs, xs[2])                    # Prints "[3, 1, 2] 2"
print(xs[-1]) # Negative indices count from the end of
the list; prints "2"

xs[2] = 'foo' # Lists can contain elements of different
types
print(xs)                  # Prints "[3, 1, 'foo']"
xs.append('bar') # Add a new element to the end of the
list
print(xs)                  # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop() # Remove and return the last element of
the list
print(x, xs)                # Prints "bar [3, 1, 'foo']"
```

Chaînes de caractères



- Python 3 dispose d'une nouvelle manière de traiter des “strings” très performant et facile à utiliser: les **f-strings**
- La **f** dans f-strings est pour **fast**

```
mot1='hello'  
mot2='world'  
  
hw12 = f'{mot1} {mot2} 12' # Prints hello world 12  
print(f'Le carré de 8 est {8 * 8}') # Prints Le carré de 8 est 64  
  
name='Pierre'  
age=42  
print(f'L\'âge de {name} est {age} ans') # Prints L'âge de Pierre est 42 ans
```



- Une liste est similaire à un tableau en Python, mais elle est redimensionnable et peut contenir des éléments de différents types.

```
xs = [3, 1, 2]                      # Create a list

print(xs, xs[2])                    # Prints "[3, 1, 2] 2"
print(xs[-1]) # Negative indices count from the end of
the list; prints "2"

xs[2] = 'foo' # Lists can contain elements of different
types
print(xs)                  # Prints "[3, 1, 'foo']"
xs.append('bar') # Add a new element to the end of the
list
print(xs)                  # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop() # Remove and return the last element of
the list
print(x, xs)                # Prints "bar [3, 1, 'foo']"
```

Accès aux éléments d'une liste



```
nums = list(range(5)) # range is a built-in function

print(nums)          # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])    # Get a slice from index 2 to 4
                    #(exclusive); prints "[2, 3]"
print(nums[2:])     # Get a slice from index 2 to the
                    # end; prints "[2, 3, 4]"
print(nums[:2])     # Get a slice from the start to
                    # index 2 (exclusive); prints "[0, 1]"
print(nums[:])      # Get a slice of the whole list;
                    # prints "[0, 1, 2, 3, 4]"
print(nums[:-1])   # Slice indices can be negative;
                    # prints "[0, 1, 2, 3]"

nums[2:4] = [8, 9] # Assign a new sublist to a slice
print(nums)         # Prints "[0, 1, 8, 9, 4]"

nums[::-1]          # inverse l'ordre de la liste
```

Iteration sur les éléments d'une liste



```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)

# Prints "cat", "dog", "monkey", each on its
own line.
```

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print(f'#{idx}: {animal}')

# Prints "#1: cat", "#2: dog", "#3: monkey",
each on its own line
```

Liste en compréhension



Syntaxe

```
new_list = [function(item) for item in list  
           if condition(item)]
```

Utilisation astucieuse des listes en Python:

```
nums = [0, 1, 2, 3, 4]  
squares = [x ** 2 for x in nums]  
print(squares)                  # Prints [0, 1, 4, 9, 16]  
  
nums = [0, 1, 2, 3, 4]  
even_squares = [x ** 2 for x in nums if x % 2 == 0]  
print(even_squares)            # Prints "[0, 4, 16]"
```



```
for x in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:  
  
    print(x)                      # Prints 0 à 9  
  
for x in range(10):  
    print(x)                      # Prints 0 à 9  
  
for x in range(7,10):  
    print(x)                      # Prints 7 8 9  
  
liste = [0,3,5,8,3,3,5,7]  
for x in liste:  
    print(x)                      # Prints 0 3 5 8 3 3 5 7
```

Conditions



```
x = 3

if x % 2:
    print("x est divisible par 2")
elif x % 3 == 0:
    print("x est divisible par 3")
else :
    print("x n'est pas divisible ni par 2 ni par 3")
```

```
x = 101
while x != 1:
    x = x - 2
    print(x)
```

Fonctions et classes



Exemple de fonction

```
def sign(x):  
    if x > 0:  
        return 'positive'  
    elif x < 0:  
        return 'negative'  
    else: return 'zero'  
  
for x in [-1, 0, 1]:  
    print(sign(x))  
#Prints "negative", "zero", "positive"
```

TPE : Python est un langage orienté objet. Qu'est ce que cela signifie?



Fonctions lambda

Syntaxe : **lambda** arguments: expression

NB: Les fonctions lambda ne peuvent contenir qu'une seule expression.

Exemples :

1. `def sum_classic (a, b) :`

`sum_lambda = lambda a, b : a+b`

`return a+b`

2. `some_numbers_list= [1, 2, 3, 87, 40, 49, 303, 34, 32, 98]`

`filtered_list = list(filter(lambda x: x > 10, some_numbers_list))`

`def my_custum_filter(x):`

`return x >10`

`some_numbers_list= [1,2,3,87,40,49,303,34,32,98]`

`filtered_list = list(filter(my_custum_filter, some_numbers_list))`

- Numpy est la bibliothèque centrale de calcul scientifique de Python. Elle permet travailler de manière très efficace avec des matrices (ou tableaux) de multiples dimensions.
- Calcul matriciel optimisé
- Disponibilité d'une bibliothèque riche en fonctions et opérations pour traiter des matrices (voir numpy.org)

Array Numpy



- Un **Array numpy** est un tableau de valeurs, indexé par un tuple d'entiers non négatifs.
- Le nombre de dimensions est le **rank** du tableau ;
- le **shape** d'un tableau est un tuple d'entiers donnant la taille du tableau le long de chaque dimension.

```
import numpy as np
a = np.array([1, 2, 3]).           # Create a rank 1 array
print(type(a))                   # Prints "<class 'numpy.ndarray'> "
print(a.shape)                   # Prints "(3,)"
print(a[0], a[1], a[2])          # Prints "1 2 3"
a[0] = 5                          # Change an element of the array
print(a)                          # Prints "[5, 2, 3]

b = np.array([[1, 2, 3], [4, 5, 6]])      # Create a rank 2 array
print(b.shape)                    # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])       # Prints "1 2 4"
```

Création des Arrays



```
import numpy as np

a = np.zeros((2, 2))          # Create an array of all zeros
print(a)                      # Prints "[[ 0.  0.]
                             #           [ 0.  0.]]"

b = np.ones((1, 2))           # Create an array of all ones
print(b)                      # Prints "[[ 1.  1.]]"

c = np.full((2, 2), 7)         # Create a constant array
print(c)                      # Prints "[[ 7.  7.]
                             #           [ 7.  7.]]"

d = np.eye(2)                 # Create a 2x2 identity matrix
print(d)                      # Prints "[[ 1.  0.]
                             #           [ 0.  1.]]"

e = np.random.random((2, 2))   # Create an array filled with random
values print(e)               # Might print "[[ 0.91940167  0.08143941]
                             #           [ 0.68744134  0.87236687]]"
```

Création des Arrays



```
import numpy as np

a = np.arange(10) # Create an array containing [0 1 2 3 4 5 6 7 8 9]

b = a.reshape(2, 5) # Adapte le shape d'une dimension à deux
# array([[0, 1, 2, 3, 4],
#        [5, 6, 7, 8, 9]])

# Create the array [-1., -0.8, -0.6, -0.4, -0.2, 0., 0.2, 0.4, 0.6,
# 0.8, 1.]

a = np.linspace(-1, 1, 11)
a = np.arange(-1., 1.01, 0.2)

# Un numpy array peut contenir des données de différents types
# Pour cela il faut le déclarer de type object

a = np.array([['A', 32, 2.5], ['B', 16, 1.5], ['C', 2, 5]], dtype=object)
```

Accès aux données



```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

print(a.shape)                      # Prints (3,4) -> 3 rows, 4 columns

# Print first two columns
print(a[0:3, 0:2])
print(a[:3, :2])
print(a[:, :2])

# Print last column
print(a[:, -1])                    # Prints [4 8 12]

# Print last row
print(a[-1, :])                   # Prints [9, 10, 11, 12]
print(a[2])                        # Prints third row [9, 10, 11, 12]
print(a[1, 3])                     # Prints 8, the element at second row and fourth column
```

```
import numpy as np
```

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
rows = [0, 2]
```

```
cols = [1, 3]
```

```
print(a[rows, cols])
```

```
# Prints elements a[0,1] and a[2,3], thus
```

```
# a[0, 1] -> first row, second column -> 2
```

```
# and
```

```
# a[2, 3] -> third row, fourth column -> 12
```

Accès aux données des Arrays



```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

print(a<8)

# Prints
#[[ True True True True]
# [ True True True False]
# [False False False False]]

print(a[a < 8])                                # Prints [1 2 3 4 5 6 7]

# Encore un exemple...
A = np.array([3, 4, 6, 10, 24, 89, 45, 43, 46, 99, 100])

ndiv3 = A[A%3!=0]
print(ndiv3)                                    # Affiche les nombres de A non divisibles par 3
                                                # [4,10,89,43,46,100]

div5 = A[A%5==0]
print(div5)                                     # Affiche les nombres de A divisibles par 5
                                                # [10 45 100]
```

Utilisation des Masques



Comment masquer ou filtrer des données ?

```
import numpy as np

a = np.array([[1, 4, 3, 2], [9, 6, 8, 7], [4, 3, 7, 1]])

mask = a > 5 # sélectionne les valeurs plus grandes que 5
print(mask)

# prints[[False, False, False, False],
#         [ True,  True,  True,  True],
#         [False, False,  True, False]]

print(np.sum(a[mask])) # additionne seulement les valeurs sélectionnées par le masque
# 9 + 6 + 8 + 7 + 7 = 37

x = np.arange(10)
mask = x > 5

print(mask) # Prints [False, False, False, False, False, False, True, True, True, True]
[x[mask] ** 2] # Prints [36, 49, 64, 81]
```

Calcul avec les éléments d'un Array

■ min, argmin, sum.

```
import numpy as np

a = np.array([[1, 4, 3, 2], [9, 6, 8, 7], [4, 3, 7, 1]])

print(np.sum(a)) # additionne toutes les valeurs; affiche 55

# additionne les colonnes
print(a.sum(axis=0)) # Prints [14 13 18 10]

# additionne les lignes
print(a.sum(axis=1)) # Prints [10 30 15]

# le minimum par ligne
print(a.min(axis=1)) # Prints [1 6 1]

# la position du minimum par ligne
print(a.argmin(axis=1)) # Prints [0 1 3]

# autres opérations: max, round, nonzero, prod (product), mean, std, etc...
```

Calcul avec les éléments d'un Array



Listes en compréhension

```
import numpy as np

x = np.arange(10)
y = [item**2 for item in x]
print(y)

# Prints [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] -> le carré des
valeurs de x

x = np.arange(10)
y = [item**2 for item in x if item > 5]
print(y)

# Prints [36, 49, 64, 81] -> le carré des valeurs de x plus grandes
que 5
```

Calcul avec les éléments d'un Array

■ Opérations matricielles

```
import numpy as np

a = np.array([[1,4], [9,6]])
b = np.array([[1,2], [5,6]])

print(a+b)
# Prints
# [[ 2  6]
# [14 12]]

print(np.dot(a,b))

#Calcule et affiche le produit matriciel

#[[1,4], [[1,2],      -> [[1x1 + 4x5 1x2 + 4x6]      -> [[21 26]
# [9,6]] [5,6]]          [9x1 + 6x5 9x2 + 6x6]]      [39 54]]
```

Calcul avec les éléments d'un Array

■ Concaténation

```
import numpy as np

a = np.array([[1,4],
              [9,6]])

b = np.array([[1,2],
              [5,6]])

np.concatenate((a,b),axis=0)                      # axis=0 est la valeur par défaut

# Generates
# array([[1, 4],
#        [9, 6],
#        [1, 2],
#        [5, 6]])

np.concatenate((a,b),axis=1)

# Generates
# array([[1, 4, 1, 2],
#        [9, 6, 5, 6]])
```

Fichiers Numpy



```
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

np.save('myarray',a)
# Enregistre le tableau a dans un fichier .npy

b = np.load('myarray.npy')
# Lit un tableau à partir d'un fichier .npy
```

Types de données basiques



- Les tableaux **Numpy** contiennent des valeurs d'un seul type (sauf type object),
- Il est donc important d'avoir une connaissance détaillée de ces types et de leurs limites.
- Comme Numpy est construit en C, les types sont familiers aux utilisateurs du C, du Fortran et d'autres langues apparentées.
- Lors de la construction d'un tableau, le type de données est spécifié en utilisant une chaîne de caractères :
Exemple: `np.zeros(10, dtype='int16')`
`# -32768 < valeurs < 32767`
- **Autres types de données:** `bool_`, `uint16` (0..65535), `int32`, `int64`, `float16`, `float32`, `float64`, etc.

- Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques, tels que: plots, bar charts, scatter-plots, etc.
- Pour installer matplotlib :
pip install matplotlib
- Site de référence: **matplotlib.org**

Premier plot



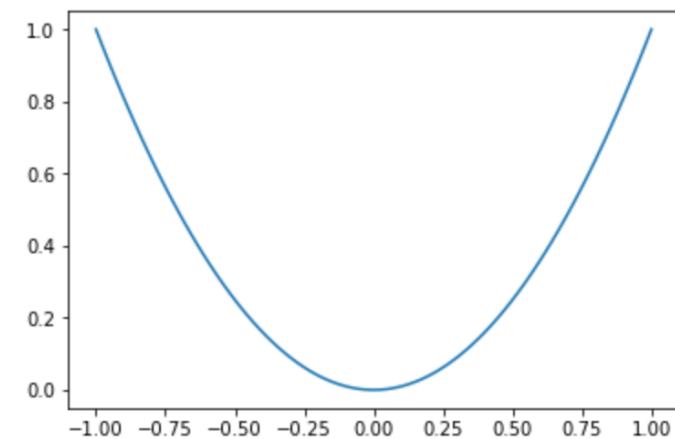
```
import numpy as np

from matplotlib import pyplot as plt
# importer les outils pour tracer des courbes

# affiche les graphiques dans le notebook

%matplotlib inline
# Exemple de fonction quadratique

x = np.linspace(-1.0,1.0,num=100)
squares = [val ** 2 for val in x]
plt.plot(x,squares)
```



Mise en forme d'un graphique

Titre, nom des axes, limites



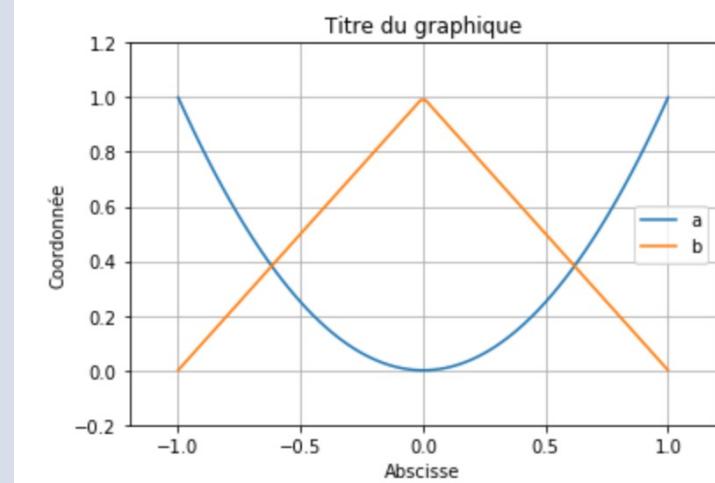
```
import numpy as np
from matplotlib import pyplot as plt

%matplotlib inline
x = np.linspace(-1.0,1.0,num=100)
squares = [val ** 2 for val in x]
linfunc = [1 - abs(val) for val in x]

plt.title('Titre du graphique')
plt.xlabel('Abscisse')
plt.ylabel('Coordonnée')
plt.xlim(-1.2,1.2)
plt.ylim(-0.2,1.2)

plt.plot(x,squares)
plt.plot(x,linfunc)

plt.legend('ab')
plt.grid()
```



Multiples courbes

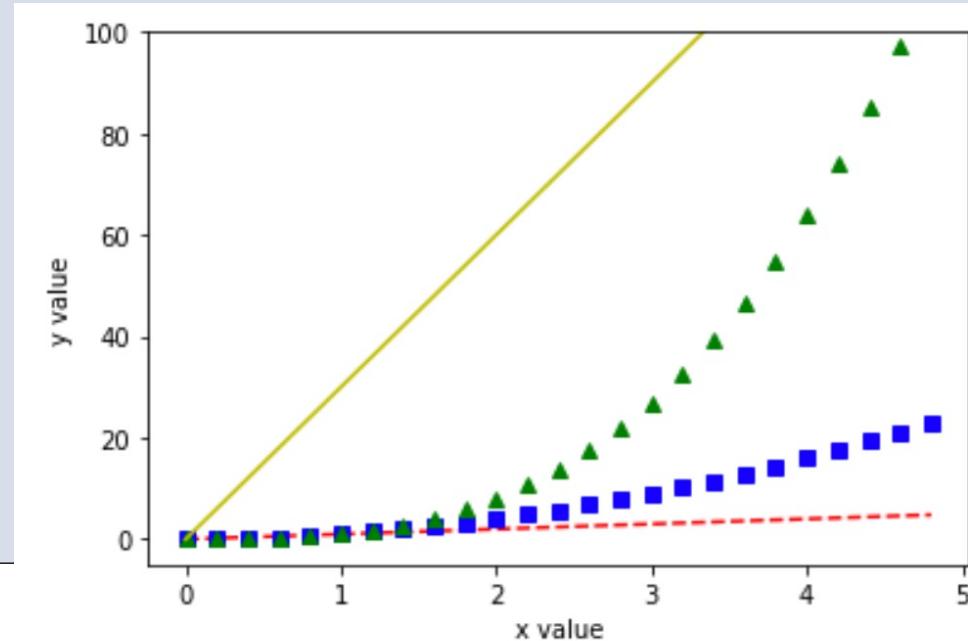
Pour visualiser différents tendances ou des fonctions d'une même variable, p.ex. $f(t)=t$, $g(t)=t^2$, $h(t)=t^3$, $j(t)=30t$, etc.



```
# x axis data
t = np.arange(0., 5., 0.2)
# red dashes, blue squares, green triangles, yellow line

plt.plot(t, t, 'r--')
plt.plot(t, t**2, 'bs')
plt.plot(t, t**3, 'g^')
plt.plot(t, 30*t, 'y')

plt.xlabel('x value')
plt.ylabel('y value')
plt.ylim(-5,100)
```



Sub - plots

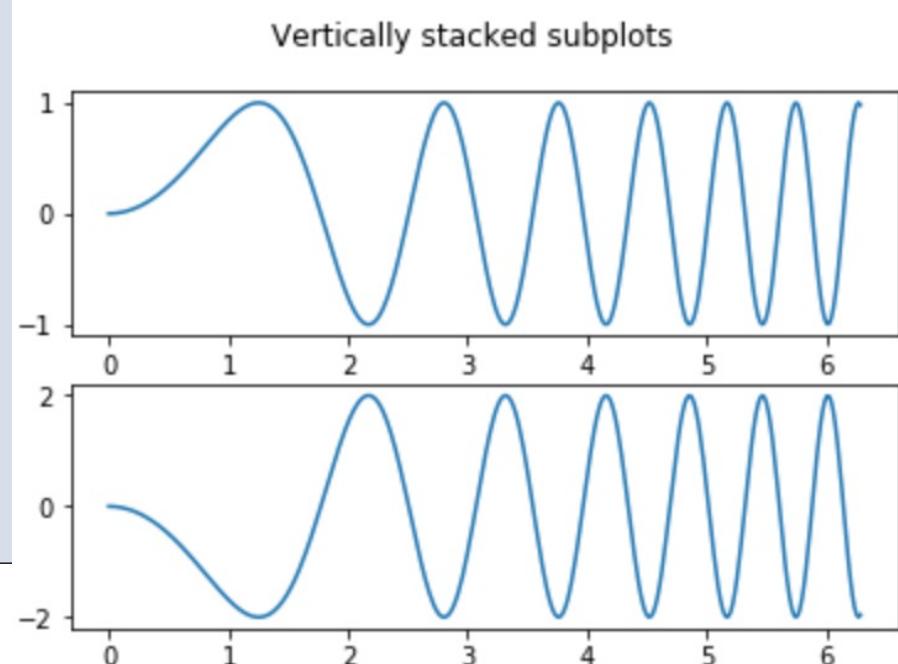
L'utilisation de sub-plots facilite la comparaison et l'analyse des résultats



```
x = np.linspace(0, 2 * np.pi, 360)
y = np.sin(x ** 2)

fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')

axs[0].plot(x, y)
axs[1].plot(x, -2*y)
```



Sub - plots

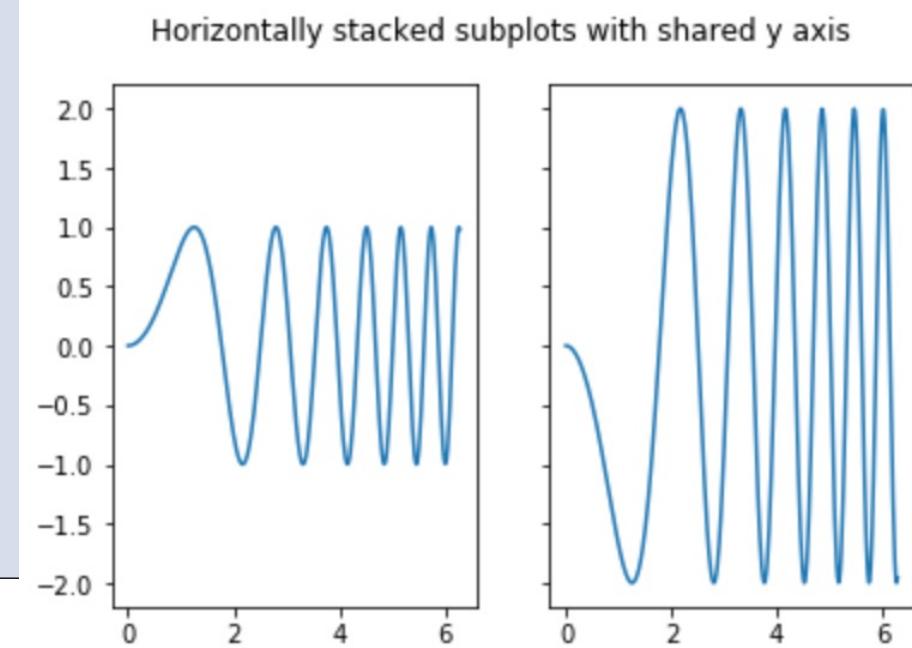
L'utilisation de sub-plots facilite la comparaison et l'analyse des résultats



```
x = np.linspace(0, 2 * np.pi, 360)
y = np.sin(x ** 2)

fig, axs = plt.subplots(1,2, sharey = True)
fig.suptitle('Horizontally stacked subplots')

axs[0].plot(x, y)
axs[1].plot(x, -2*y)
```



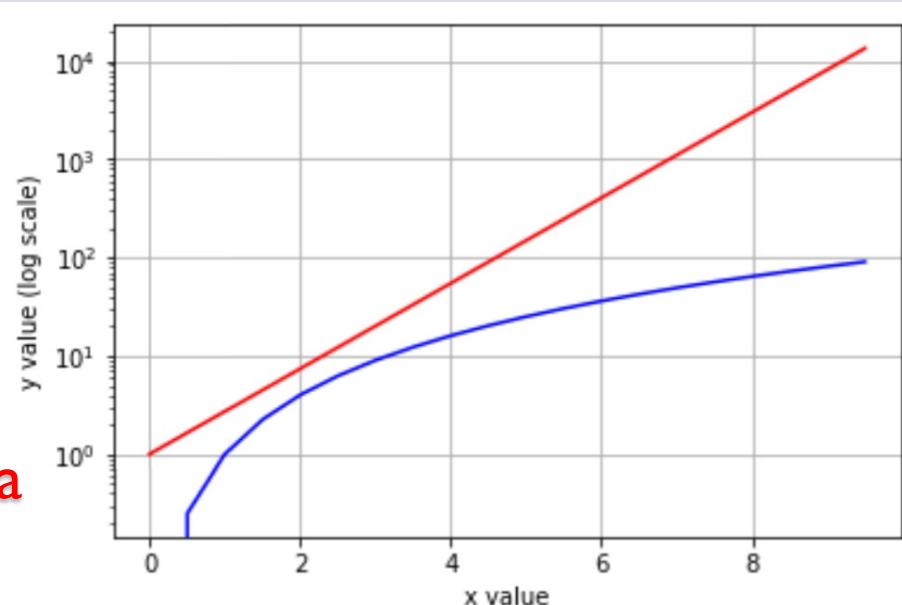
Echelle logarithmique



```
# x axis data
t = np.arange(0., 10., 0.5)
# red dashes, blue squares, green triangles, yellow line

plt.plot(t, t**2, 'b')
plt.plot( t, np.exp(t), 'r')

plt.xlabel('x value')
plt.ylabel('y value (log scale)')
plt.yscale('log')
plt.grid()
```



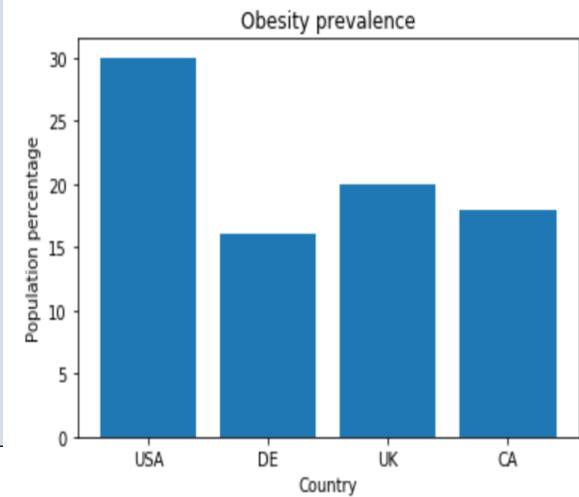
TPE : Laquelle des courbes est la fonction exponentielle ?

Bar - graph

- facilite la comparaison des valeurs associées à divers groupes ou classes, surtout si ces différences ne sont pas trop petites.
- permet aussi de comparer des valeurs à différents moments.

```
pays = ['USA', 'DE', 'UK', 'CA']
obesity = [30, 16, 20, 18]
x = np.arange(len(pays))
plt.bar(x, obesity)
plt.xticks(x, pays)

plt.title('Obesity prevalence')
plt.xlabel('Country')
plt.ylabel('Population percentage')
```



Scatter plot

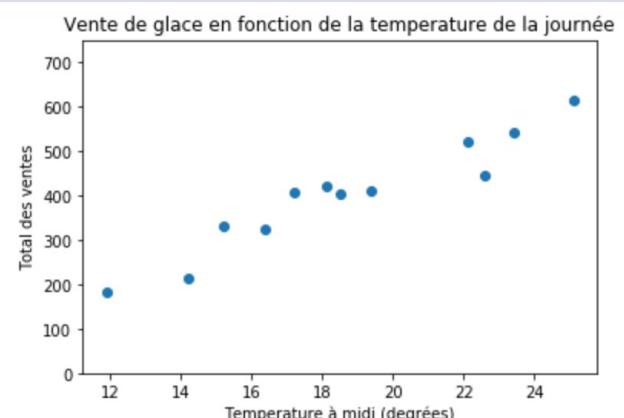


Ou X-Y plot, il permet de déterminer la relation entre deux variables. P.ex., si les deux augmentent en même temps, on peut dire qu'elles sont positivement correlées.

```
temp = [14.2, 16.4, 11.9, 15.2, 18.5, 22.1, 19.4, 25.1, 23.4,  
18.1, 22.6, 17.2]
```

```
vente_glace = [215, 325, 185, 332, 406, 522, 412, 614, 544,  
421, 445, 408]
```

```
plt.scatter(temp,vente_glace)  
plt.ylim(0,750)  
plt.xlabel('Temperature à midi (degrés)')  
plt.ylabel('Total des ventes')  
plt.title('Vente de glace en fonction de la  
'temperature de la journée')
```



Exportation de graphique

- Exporter les résultats pour documenter ou créer un rapport.

```

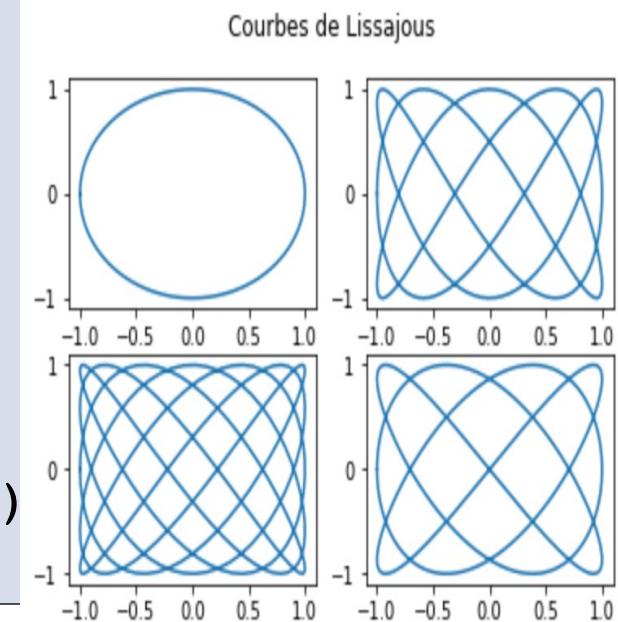
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

a = [1,3,5,3] # plotting the curves for
b = [1,5,7,4] # different values of a/b
delta = np.pi/2
t = np.linspace(-np.pi,np.pi,300)

for i in range(0,4):
    x = np.sin(a[i] * t + delta)
    y = np.sin(b[i] * t)
    plt.subplot(2,2,i+1)
    plt.plot(x,y)

plt.suptitle('Courbes de Lissajous')
plt.savefig('Lissajous.png')

```



- PANDAS (PythoN Data Analysis library) est une bibliothèque open source fournissant des structures de données et des outils d'analyse de données performants et faciles à utiliser pour le langage de programmation Python.
- Installation: **pip install pandas**
- La structure de données de base de *pandas* s'appelle un “**data frame**”.
 - ✓ similaire à un tableau SQL ou à une feuille de calcul excel.
 - ✓ existe aussi en R.

Création de dataframe

Création à partir de dictionnaire



```
import pandas as pd

data =
{'Country': ['Suisse', 'Allemagne', 'France', 'UK'],
'Population': [8654622, 83783942, 65273511, 67886011],
'Budget_education': [15700, 11700, 10500, 11600],
'Member_EU': [False, True, True, False],
'Employment_rate': [79.8, 76.3, 65.1, 75.4] }

df = pd.DataFrame(data)
```

In [51]: df

Out[51]:

	Country	Population	Budget_education	Member_EU	Employment_rate
0	Suisse	8654622	15700	False	79.8
1	Allemagne	83783942	11700	True	76.3
2	France	65273511	10500	True	65.1
3	UK	67886011	11600	False	75.4

Accès aux données d'un dataframe



```
print(df[ 'Country' ])  
df.loc[0]  
  
df.loc[df[ 'Country' ]=='Suisse']  
  
df.loc[df[ 'Member_EU' ]==True]  
  
df.loc[df[ 'Population' ] < 10000000]  
  
df.loc[2, 'Population']
```

```
df.iloc[2,1]
```

In [51]: df

Out[51]:

	Country	Population	Budget_education	Member_EU	Employment_rate
0	Suisse	8654622	15700	False	79.8
1	Allemagne	83783942	11700	True	76.3
2	France	65273511	10500	True	65.1
3	UK	67886011	11600	False	75.4

TPE : Que font les méthodes:
at(), iat(), query() ?

Dataframe et dimensionnalité



```
# Nombre de dimensions du dataframe  
  
df.ndim # affiche 2  
  
df.size # affiche 20  
  
df.shape # affiche (4,5)  
  
df.shape[0] # affiche 4  
  
df.dtypes
```

In [51]: df

Out[51]:

	Country	Population	Budget_education	Member_EU	Employment_rate
0	Suisse	8654622	15700	False	79.8
1	Allemagne	83783942	11700	True	76.3
2	France	65273511	10500	True	65.1
3	UK	67886011	11600	False	75.4

Append et Drop



```
# Pour créer une nouvelle ligne d'un dataframe  
spain = pd.Series(data=['Spain',46754778,8900,True,60.7],index=df.columns)  
  
# Rajouter la nouvelle ligne au dataframe  
df = df.append(spain, ignore_index=True)  
  
# Pour effacer une ligne on donne l'index df =  
df.drop(labels=2) # on efface la France
```

	Country	Population	Budget_education	Member_EU	Employment_rate
0	Suisse	8654622	15700	False	79.8
1	Allemagne	83783942	11700	True	76.3
2	France	65273511	10500	True	65.1
3	UK	67886011	11600	False	75.4
4	Spain	46754778	8900	True	60.7

TPE : Que vaut df.loc[2] après le drop ?

Insert et Pop



```
# Pour rajouter une nouvelle colonne  
df[ 'autre' ] = [ 3.25, 2.5, 3, 9.2 ]
```

	Country	Population	Budget_education	Member_EU	Employment_rate	autre
0	Suisse	8654622	15700	False	79.8	3.25
1	Allemagne	83783942	11700	True	76.3	2.50
2	France	65273511	10500	True	65.1	3.00
3	UK	67886011	11600	False	75.4	9.20

```
df.pop( 'autre' )
```

```
df.insert(loc=2,  
column='autre',value=[ 3.25, 2.5, 3, 9.2 ] )
```

	Country	Population	autre	Budget_education	Member_EU	Employment_rate
0	Suisse	8654622	3.25	15700	False	79.8
1	Allemagne	83783942	2.50	11700	True	76.3
2	France	65273511	3.00	10500	True	65.1
3	UK	67886011	9.20	11600	False	75.4

Calcul avec des données d'un dataframe



```
# Rajoutons des colonnes
```

```
df[ 'Labor_force' ] =  
[ 4922000, 43773000, 29682000, 33964000 ]
```

```
df[ 'Working_percentage' ]=  
df[ 'Labor_force' ]/df[ 'Population' ]*100
```

	Country	Population	Budget_education	Member_EU	Employment_rate	Labor_force	Working_percentage
0	Suisse	8654622	15700	False	79.8	4922000	56.871346
1	Allemagne	83783942	11700	True	76.3	43773000	52.245095
2	France	65273511	10500	True	65.1	29682000	45.473270
3	UK	67886011	11600	False	75.4	33964000	50.030926

Ordonancement de valeurs



Pour les observations par Labor force (du plus petit au plus grand)

	Country	Population	Budget_education	Member_EU	Employment_rate	Labor_force	Working_percentage
0	Suisse	8654622	15700	False	79.8	4922000	56.871346
2	France	65273511	10500	True	65.1	29682000	45.473270
3	UK	67886011	11600	False	75.4	33964000	50.030926
1	Allemagne	83783942	11700	True	76.3	43773000	52.245095

df.sort_values(by='Labor_force')

	Country	Population	Budget_education	Member_EU	Employment_rate	Labor_force	Working_percentage
0	Suisse	8654622	15700	False	79.8	4922000	56.871346
2	France	65273511	10500	True	65.1	29682000	45.473270
3	UK	67886011	11600	False	75.4	33964000	50.030926
1	Allemagne	83783942	11700	True	76.3	43773000	52.245095

Numpy et Pandas



Les fonctions numpy s'appliquent aux Séries pandas.

```
print(np.max(df[ 'Population' ]))  
  
print(np.average(df[ 'Population' ]))  
  
print(np.argmax(df[ 'Population' ]))
```

Caractérisation des données avec **describe()**



Description des données d'iris **df.describe()**

	sepal length	sepal width	petal length	petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Analyse exploratoire des données



Graphiques pour l'analyse exploratoire des données

- Histogramme : `plt.hist(df['petal length'])`
- Box plot : `plt.boxplot(A, showmeans=True)`
- Scatter Matrix : `pd.plotting.scatter_matrix(df, alpha=0.2, figsize =(10,10), color=colors)`

TPE : Comment utiliser les box plots pour comparer des variables (Voir Modèles à base des règles)

Pandas offre des nombreuses interfaces capables de lire des données à partir des fichiers de différents types.

Exemples : Excel, csv json (JavaScript Object Notation, facile à parser), etc.

```
import pandas as pd
column_names = ["sepal length", "sepal width", "petal
length", "petal width", "Type of flower"]
df =
pd.read_csv("https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data",
names=column_names)

df.to_csv('out.csv', index=False)
```