



Python pour l'analyse de données

Initiation

Avril 2025
Martin Puig
martin.puig@umontpellier.fr

C'EST QUOI PYTHON?

BASE DU LANGAGE (PARTIE 1)

Instruction,
Commentaire &
Variables

OBJETS PYTHON

List – Tuple -- Dictionnaire -- Set

1

2

3

STRUCTURE DU COURS

4

BASES DU LANGAGE (PARTIE 2)

Instructions de contrôle
Boucles
Fonctions et exceptions
Module – Package -- Import

5

MANIPULATION DE DONNÉES AVEC PANDAS

Types
Import/export
Data manipulation

TD

6

INTRODUCTION À LA VISUALISATION DE DONNÉES

Principes
Visualisation avec
pandas

TD

01

C'est quoi Python?



Python

Langage de programmation créé en 1989 par Guido van Rossum. Actuellement version 3 (3.12), support 5 ans

- Langage haut niveau
- Langage interprété (non compilé) ≠ C++ ou Fortran
- Orienté objet
- Complet pour **Datascience, Machine Learning, Artificial Intelligence** mais ne détrône pas R pour les **statistiques**.
- Applicable à tout type d'OS (Operating System): Ubuntu, Window, Mac, Android ...



Haut niveau

C'est un langage conçu pour être **facile d'utilisation**

- Typage dynamique
- Pas besoin de gérer la mémoire des variables
- Les packages python facilitent au maximum **la gestion** de nombreux types de données différents
- Très peu d'interaction avec les composantes hardware sous-jacentes.

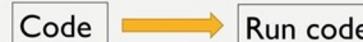
Interprété

COMPILED VS INTERPRETED LANGUAGES

❑ A compiled language is when a person writes the code the compiler separates the file and the end result is an executable file. Basically, the owner keeps the source code.



❑ Interpreted languages are different because the code is not compiled first hand. Instead, a copy is given to another machine and that machine interprets it. An example is **JavaScript** and **PHP** (which is used everywhere online).



Compiled		Interpreted	
PROS	CONS	PROS	CONS
ready to run	not cross platform	cross-platform	interpreter required
often faster	inflexible	simpler to test	often slower
source code is private	extra step	easier to debug	source code is public

- Donc **moins rapide** que les langages compilés.
- Plus simple pour la gestion de projets et les dépendances
- Mais Python intègre des éléments compilés (écrit en C++)

Les avantages de Python

Python est un langage **auto-suffisant**, on peut y **développer toute la pipeline** de travail de A à Z et ce pour de nombreux domaines

- Simplicité du langage (notamment parce qu'il est haut-niveau)
- **Indentation obligatoire** = lisibilité
- Typage dynamique
- Très grande communauté : développeurs de package, blogs, communauté
- Structuration multifichier très facile pour gérer les **gross projets**
- Autosuffisant pour de nombreuses applications
- Modèle orienté objet puissant (mais pas obligatoire)



Applications multiples

Sites web/
applications/API

Gestion/
Administration
de systèmes

Analyse de
données (tous
formats)

Applications
graphiques

Recherche
scientifique
(tous domaines)

Simulations
numériques

DataScience,
Machine
Learning, IA

Et bien d'autres ...



Comment comparer à d'autres langages?

Si python est très généraliste, ce n'est pas toujours le langage le plus adapté :

- Pour **l'optimisation** (Recherche Opérationnelle ou ML), de nombreux spécialistes vont préférer **Julia**
- Pour les **statistiques** (surtout pour les tests), il vaut mieux utiliser **R**.
- Pour les réseaux de neurones très pointus, il vaut mieux utiliser **CUDA** pour être au plus proche du fonctionnement hardware (personne ne fait ca)
- Par contre pour le **Machine Learning, python** est idéal.



Packages Python et environnements

- Python basé sur le développement de **packages** en open-source
- Package = ensemble de **modules** (fichiers python) importables (code réutilisable)
- Un package pour chaque type de données ou application

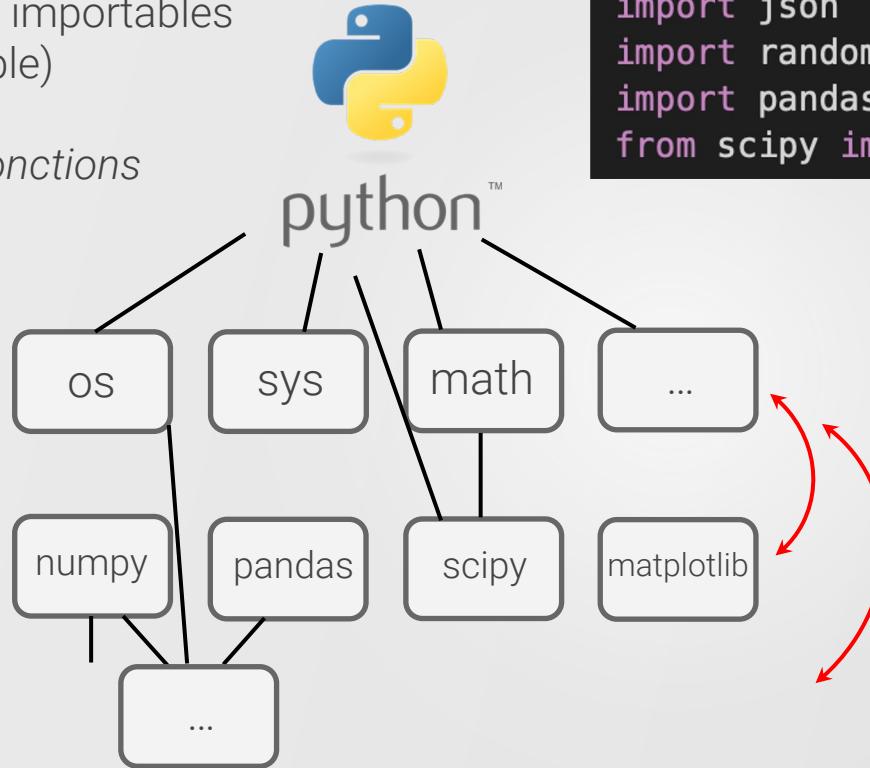
Tout est fait pour faciliter le travail et interagir de façon intuitive avec les objets, donc pas besoin de tout retenir (d'autant que les assistants de code sont de mieux en mieux)

Un langage modulaire orienté autour de packages

package = ensemble de fichiers Python importables (code réutilisable) comprenant des objets et fonctions

Packages natifs

Packages additionnels à installer



Syntaxe

```
import json  
import random  
import pandas as pd  
from scipy import stats
```

importer un package entier

importer un objet/une fonction d'un package

développés en open-source, nombreuses versions qui évoluent constamment

Incompatibilités de versions possibles :
Gestionnaires de packages





NumPy



xarray



seaborn



Principaux packages



SciPy

django



Keras



PyTorch



TensorFlow



Packages Python et environnements

- Les packages **évoluent constamment** et ont de nombreuses versions
- Certains packages sont **natifs** et d'autres sont à **installer**
 - Les packages os, glob, sys, argparse, math, random, ... sont natifs
 - **Il faut installer matplotlib, pandas, etc.**

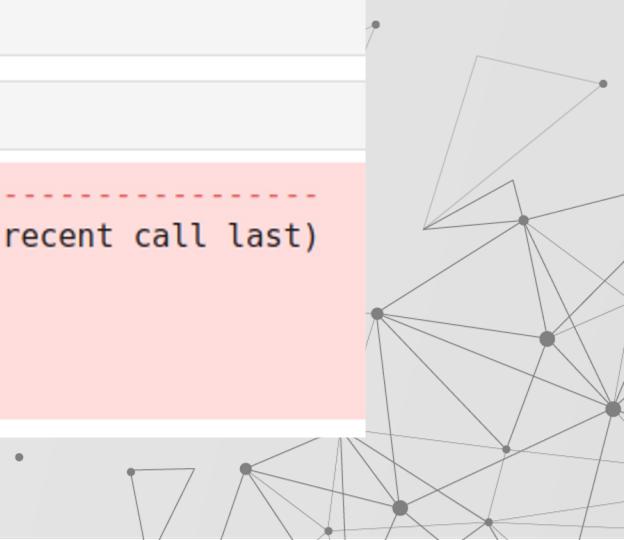
```
[3]: import os
```

```
[4]: import torch
```

```
ModuleNotFoundError  
Cell In[4], line 1  
----> 1 import torch
```

```
Traceback (most recent call last)
```

```
ModuleNotFoundError: No module named 'torch'
```



Packages Python et environnements

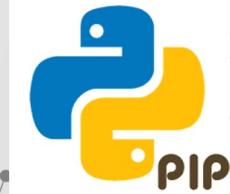
De base, Python n'intègre qu'un petit nombre de packages pour ne pas avoir à gérer trop de problèmes de compatibilité

- Chaque package dépend de nombreux autres packages avec parfois des **incompatibilités de version**
- Il faut donc un **gestionnaire de packages** pour gérer le téléchargement et l'installation du package dans l'**environnement python**

```
(base) thomas@thomas-NP5x-6x-7x-SNx:~$ conda install matplotlib
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done
```



```
(base) thomas@thomas-NP5x-6x-7x-SNx:~$ pip install matplotlib
Collecting matplotlib
  Obtaining dependency information for matplotlib from https://files.pythonhosted.org/packages/ef/1d/bf1d78126c3d106100232d3a18b7f3732e7dc3b71ee38ab735e4064b19cc/matplotlib-3.8.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
```



Packages Python et environnements

Les gestionnaires de packages tels que **pip** ou **conda** permettent d'installer des packages mais aussi de créer des **environnements virtuels** python

- Un environnement = installation de quelques packages dans un même espace
- Les environnements permettent de compartimenter python : **très bonne pratique**
- On peut partager les environnements aux collaborateurs
- **Définitions = différents environnements**

```
# create environment 'myenv'  
$ conda create --name myenv  
  
# list all your environments  
$ conda env list
```

```
# activate environment 'myenv'  
$ conda activate myenv  
  
(myenv) $ conda install numpy  
# Note: all dependencies of numpy are also installed. Accept with  
'yes'.  
  
# list all installed packages in myenv  
(myenv) $ conda list
```



Installer Python

- Python (seul) : <https://www.python.org/downloads/>
- Distributions (Python, interpréteur, packages) :
<https://wiki.python.org/moin/PythonDistributions>
- **Anaconda** : distribution Python multiplateforme très répandue pour la *data science* et le *machine learning*, cf. <https://docs.anaconda.com/anaconda/install/index.html>
- **Miniconda** : installation minimale pour Anaconda (qqs packages vs centaines)
- **conda** : package de gestion d'environnement Python (installation, M&J de packages)
<https://conda.io/projects/conda/en/latest/user-guide/install/index.html#regular-installation>

Développer du code Python



python™

shell

Console Python de base

Utilité limitée sauf pour single-line codes

Éditeur de texte

Interface via le terminal ou logiciel

vim, bloc notes, textEdit, ...

IDE

Environnement de développement intégré

PyCharm, Spyder, Visual Studio Code, Eclipse, Sublime Text, ...

notebooks

Développement interactif via navigateur web

Jupyter notebooks, Google Colab

Code opérationnel

Exploration rapide

Code Python via IDE

La meilleure façon de développer du code opérationnel :

- Le plus **d'extensions** possibles pour faciliter le code (correction, documentation, **assistants** ...)
- Structurer facilement des programmes complexes
- Recommandation : **VisualStudio**

```
File Edit Selection View Go Run Terminal Help
utils > utils_plot.py 9+ x
EXPLORER
PRODUCT_BLACK_CARBON
__init__.py
CONSTANTS.py
utils_algo_flare_detection.py
utils_atmo_noise.py
utils_clean.py
utils_cloud_mask.py
utils_compute_BL_LUT.py
utils_dataset_Unet.py
utils_flare_detection.py
utils_lattice.py
utils_load_models.py
utils_metadata.py
utils_plot.py 9+ x
utils_plume_mask.py
utils_plume_rate.py
utils_plumes.py
utils_query.py
utils_raw_S2.py
utils_read.py
utils_slicing.py
utils_transforms.py
utils_unet.py
utils_validation.py
utils.py
.gitignore
OUTLINE
TIMELINE
utils > plot_prediction_on_batch > name_in_tensorboard
110     i1g, dx = plt.subplots(1, 3, figsize=(10*3, 10))
111
112     ax[0].set_title("Ground truth")
113     im = ax[0].imshow(y_train[i, -1, :, :].detach().cpu().numpy(), cmap=plt.cm.inferno, vmin=0, vmax=1)
114     fig.colorbar(im, ax=ax[0], orientation='horizontal', fraction=.05)
115     ax[0].axis('off')
116
117     ax[1].set_title("Model prediction")
118     im = ax[1].imshow(predictions[i, :, :].detach().cpu().squeeze().numpy(), cmap=plt.cm.inferno, vmin=0, vmax=1)
119     fig.colorbar(im, ax=ax[1], orientation='horizontal', fraction=.05)
120     ax[1].axis('off')
121
122     ax[2].set_title("RGB map")
123     im = ax[2].imshow(np.moveaxis(x_train[i, -1, l_idx_rgb, :, :].detach().cpu().numpy(), 0, -1), cmap='gray')
124     fig.colorbar(im, ax=ax[2], orientation='horizontal', fraction=.05)
125     ax[2].axis('off')
126
127     name_in_tensorboard = f'validation_batch_{epoch}_{epoch_index}'
128
129     plot_to_tensorboard(
130         writer,
131         fig,
132         i,
133         name_in_tensorboard)
134
135     def plot_tile_RGB_BC_prediction(
136         sentinel: np.ndarray,
137         band names: List[str],
138         pred: np.ndarray,
139         lats: np.ndarray,
140         lons: np.ndarray,
141         url_save: str,
142         band names: List[str],
143         pred: np.ndarray,
144         lats: np.ndarray,
145         lons: np.ndarray,
146         url_save: str,
```

Ln 133, Col 24 Spaces 4 UTF-8 LF Python 3.10.12 64-bit 94 Spell

Code Python sur éditeur de texte

On peut aussi développer du code Python sur un éditeur de texte tel que **vim**

- C'est beaucoup moins pratique
- Cela permet néanmoins de développer une **culture IT**
- Permet de gérer les **connexions SSH/serveurs** à distance

```
thomas@thomas-NP5x-6x-7x-5Nx: ~/Desktop/CODE/SAR_many
```

```
import json
import os
import pdb
import time
from typing import Tuple

import numpy as np
from torch.utils.tensorboard import SummaryWriter
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader
import tqdm

from utils.utils_constants import DEVICE
from utils.utils_dataset_dataloader import (
    create_loader,
    make_small_dataset)
from utils.utils_model import load_model

def run_val_loop(
    model: nn.Module,
    loader_val: DataLoader,
    criterion,
    type_model: str = 'resnet18') -> Tuple[float, float]:
    """
    Run model on validation dataset.

    Inputs:
    -- REPLACE --
    
```



Code Python sur Notebook

Très utile pour **explorer des données** et faire des projets qui intègrent des parties équivalentes de texte et de code

- Très rapide pour **visualiser**
- Inclusion de code et de texte
- Manipulation interactive dans un navigateur web (en local)
- Le code est divisé en **cellules** (exécutables) dont on peut facilement intervertir l'ordre
- N'encourage pas les **bonnes pratiques de code** (variables globales, compartimentation du code, etc...)

The screenshot shows a Jupyter Notebook window titled "jupyter Lecture-2B-Single-Atom-Lasing (unsaved changes)". The interface includes a toolbar with file operations like File, Edit, View, Insert, Cell, Kernel, Help, and a CellToolbar button. Below the toolbar are two code cells:

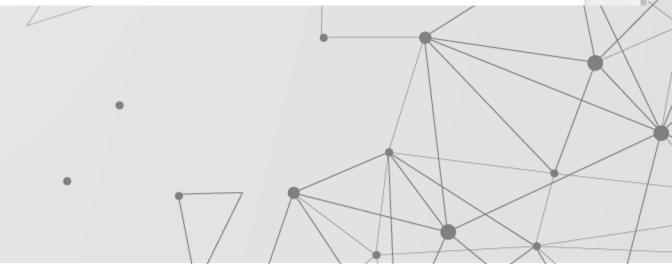
```
In [1]: # setup the matplotlib graphics library and configure it to show
# figures inline in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

```
In [2]: # make qutip available in the rest of the notebook
from qutip import *
from IPython.display import Image
```

Cell [2] contains the text "Introduction and model" followed by a descriptive paragraph about a single atom coupled to a cavity mode. Below this is another code cell:

```
In [3]: Image(filename='images/schematic-lasing-model.png')
```

Cell [3] displays a schematic diagram of a cavity mode. The diagram shows two vertical mirrors at the ends of a horizontal yellow rectangular region representing the cavity. Inside the cavity, there is a wavy line representing an atom. Two arrows point away from the cavity: one upwards labeled $|e\rangle$ and one downwards labeled $|g\rangle$. Between the two arrows is a curved arrow labeled g , representing the coupling strength. To the right of the cavity, an arrow labeled κ points outwards, representing the decay rate.



Code Python sur Notebook

```
conda install -c conda-forge notebook
```

```
[(base) alexandre@mbpdealexandre ~ % jupyter notebook
[I 2024-03-25 08:12:38.007 LabApp] JupyterLab extension loaded from /Users/alexandre/opt/anaconda3/lib/python3.9/site-packages/jupyterlab
[I 2024-03-25 08:12:38.007 LabApp] JupyterLab application directory is /Users/alexandre/opt/anaconda3/share/jupyter/lab
[I 08:12:38.014 NotebookApp] Serving notebooks from local directory: /Users/alexandre
[I 08:12:38.014 NotebookApp] Jupyter Notebook 6.4.5 is running at:
[I 08:12:38.014 NotebookApp] http://localhost:8888/?token=351c4bb73967e9bb68baaf59a5fada6a401ecd0d9501e3ac
[I 08:12:38.014 NotebookApp] or http://127.0.0.1:8888/?token=351c4bb73967e9bb68baaf59a5fada6a401ecd0d9501e3ac
[I 08:12:38.014 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 08:12:38.032 NotebookApp]

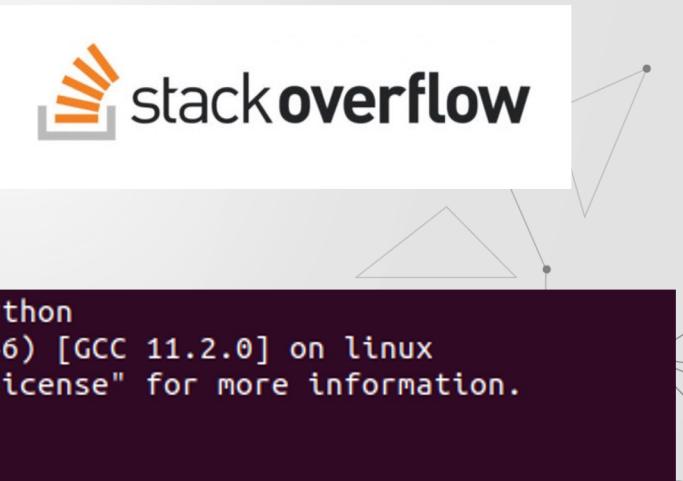
To access the notebook, open this file in a browser:
  file:///Users/alexandre/Library/Jupyter/runtime/nbserver-2635-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=351c4bb73967e9bb68baaf59a5fada6a401ecd0d9501e3ac
  or http://127.0.0.1:8888/?token=351c4bb73967e9bb68baaf59a5fada6a401ecd0d9501e3ac
```



Accéder à la documentation

Nombreuses façons d'obtenir de la documentation = choisir la plus productive en fonction de son choix d'environnement et de la question posée :

- Jupyter notebook : SHIFT + TAB
- Visual Studio : extension, surlignage, etc...
- Page web du package
- Depuis le terminal
- stackoverflow
- ...



Au-delà de la documentation : l'assistant IA

Depuis ChatGPT : nombreux assistants IA de code

- **Utiliser une version intégrée à un IDE** parce que
 - Pour le chat : permet de **contextualiser automatiquement** la question de code
 - Pour la complétion : **auto-complétion IA**
- **Gain de productivité majeur** : gare à la concurrence !
- **Gain sur la courbe d'apprentissage** pour chaque nouveau framework : très bonne combinaison avec l'utilisation de Python qui permet de tout faire : Web, DataScience, ML, Deep Learning, etc.



Pratiquer Python dans ce cours

Lien de partage vers dossier Google Drive :

<https://drive.google.com/drive/folders/1BlcbV1MVSpcbZial1Dx8QXCA92-rbpvc?usp=sharing>

Trois options :

- (1) **Google Colab** : environnement intégré, pas de problème avec les packages ou l'environnement.
- (2) Binder : [https://github.com/Qurosity129/cours_python_initialisation_intermediaire](https://github.com/Quriosity129/cours_python_initialisation_intermediaire)
- (3) Sur votre ordinateur personnel





02

Bases du langage (Partie 1)

Instruction

- Ordre unitaire donné à un programme.
- Séparées dans le programme par un point-virgule ou un retour à la ligne.

```
print(1)
1 + 1
3E5
a, b, c = 1, 2, 3
x = 1; y = 2
```

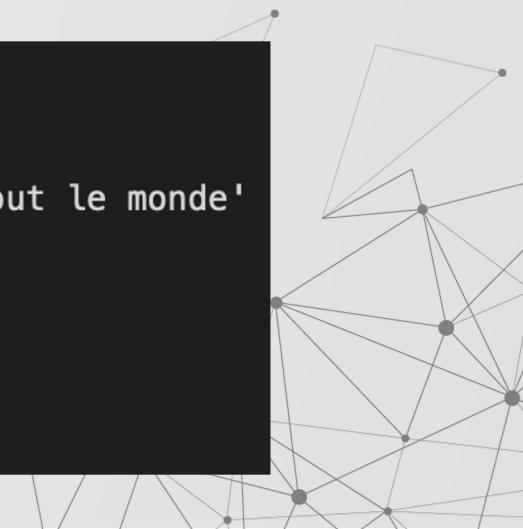


Commentaire

- Dans une ligne, tout ce qui est après un # n'est pas pris en compte par Python, c'est un commentaire.
- Cette règle ne s'applique pas lorsque le # est dans une chaîne de caractères.

```
# Ceci est un commentaire
print('Bonjour') # ceci est un commentaire
print('Bonjour # tout le monde') # Affiche 'Bonjour # tout le monde'

x, y = 1, 2
# Ajouter y à x
x = x + y
```



Variables en Python

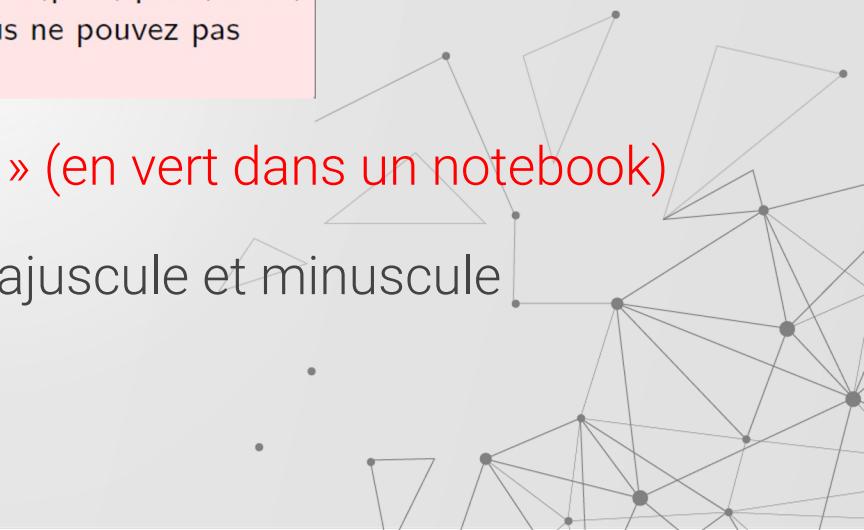
- Espace mémoire dans lequel on peut mettre une valeur.
- On dit alors qu'on affecte une valeur à une variable.

Noms réservés

and, as*, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with* et yield sont les noms que vous ne pouvez pas donner à vos variables. (*Python 3.x)

- Éviter également les mots « connus » (en vert dans un notebook)

« **Case sensible** » : Python distingue majuscule et minuscule



Variables en Python

- Se déclarent avec la commande « = »
- On dit alors qu'on affecte une valeur à une variable.

```
age = 16  
name = 'toto'
```

- Typage dynamique

```
x = 1  
print(type(x))  
y = 1.0  
print(type(y))  
z = "toto"  
print(type(z))
```



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





03

Objets Python

Objet Python

Un objet = un nom +
un contenu +
un/des type(s) (entier / str...) +
une structure de données +
1 longueur/dimension

Terminologie : On appelle souvent l'objet par sa structure (liste,
dictionnaire...)



Principaux types de données

- Entier **int**
- Flottant **float**
- Complexe **complex**
- Booléen **bool**
- String **str**
 - string = "" ou " # on peut combiner les deux
 - Indexation : string[], string[m:n], string[m:], string[:n]
 - Concaténation (+) et replication (*)
 - Longueur : len()

Fonction `type()` # envoie le type ou la structure des données



Types numériques : résumé des opérations

Operation	Result
$x + y$	sum of x and y
$x - y$	difference of x and y
$x * y$	product of x and y
x / y	quotient of x and y
$x // y$	floored quotient of x and y
$x \% y$	remainder of x / y
$-x$	x negated
$+x$	x unchanged
$\text{abs}(x)$	absolute value or magnitude of x
$\text{int}(x)$	x converted to integer
$\text{float}(x)$	x converted to floating point
$\text{complex}(re, im)$	a complex number with real part re , imaginary part im . im defaults to zero.
$c.\text{conjugate}()$	conjugate of the complex number c
$\text{divmod}(x, y)$	the pair $(x // y, x \% y)$
$\text{pow}(x, y)$	x to the power y
$x ** y$	x to the power y



Objet/Structure de données

Collection d'objets
sous Python de base



List
Tuple
Ensemble
Dictionnaire

Packages
supplémentaires



Vecteur
Matrice



Series
DataFrame



Les listes ([List](#))

- **Constructeur**
 - List = []

```
Python >>> a = ['foo', 'bar', 'baz', 'qux']

>>> print(a)
['foo', 'bar', 'baz', 'qux']
>>> a
['foo', 'bar', 'baz', 'qux']
```

- **Propriétés**
 - ordonnées
 - peuvent contenir des éléments de différents types
 - les éléments d'une liste sont accessibles par indexation
 - peuvent être imbriquées
 - **mutables**

```
x = [1, '2', 3.5]
print(type(x[0]))
print(type(x[1]))
print(type(x[2]))
```



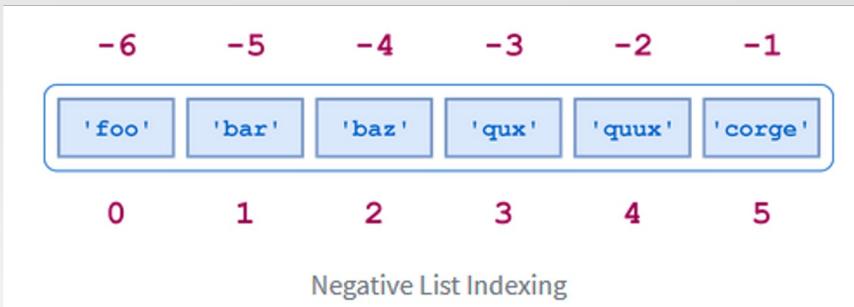
```
<class 'int'>
<class 'str'>
<class 'float'>
```

Les listes (List) : indexation

- Indices positifs ou négatifs
(gauche à droite)

Python indexe à partir de 0 !

- a[m:n]; a[m:]; a[:n];
a[m:n:step]
- a[::-1] => syntaxe d'inversion d'une liste similaire à celle d'une string



```
# Création
x = [1, 4, 9, 16, 25]
print(len(x))

# Indexation
print(x[-1])
print(x[2:5])
print(x[2:])
print(x[:3])
print(x[::-1])
```

5

25

[9, 16, 25]

[9, 16, 25]

[1, 4, 9]

[25, 16, 9, 4, 1]

Les listes (List)

Opérateurs

- o in / not in
- o concatenation (+) et replication (*)
- o len(), min(), max()

```
# Opérateurs
print(4 in x)
print(5 in x)

# Concaténation
y = [1, 4, 9] + [16, 25, 36]
print(y)

# Réplication
print(x*2)

# Built-in functions
print(len(x))
print(min(x), max(x))
```

True
False

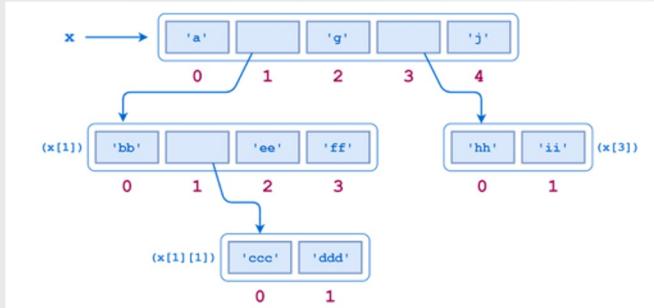
[1, 4, 9, 16, 25, 36]

[1, 4, 9, 16, 25, 1, 4, 9, 16, 25]

5
1 25

Imbrication

- o `x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']`



Les listes (List) : modifications

- Modifier/insérer/supprimer élément(s) via **indexation** ou **valeur**
- Concaténation : `append(<obj>)` /
`+=<iterable>` / `a.extend(<iterable>)`
- Insertion : `a.insert(<index>, <obj>)`
- Suppression : `a.remove(<obj>)`

```
# Lists are mutable
z = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
z[2] = 10
print(z)

# Modifier un élément
z[3] = 21
print(z)

# Ajouter un élément
z.append(10)
print(z)
z.insert(4, 30)
print(z)

# Supprimer un élément
z.remove(10)
print(z)
del z[4]
print(z)
```

```
[0, 1, 10, 3, 4, 5, 6, 7, 8, 9]
```

```
[0, 1, 10, 21, 4, 5, 6, 7, 8, 9]
```

```
[0, 1, 10, 21, 21, 4, 5, 6, 7, 8, 9, 10]
```

```
[0, 1, 10, 21, 30, 4, 5, 6, 7, 8, 9, 10]
```

```
[0, 1, 21, 30, 4, 5, 6, 7, 8, 9, 10]
```

```
[0, 1, 21, 30, 5, 6, 7, 8, 9, 10]
```

La compréhension de liste

- Concept très puissant et très pratique
- Construire et opérer sur des listes de manière concise et élégante

```
squares = []
for x in range(10):
    squares.append(x**2)
squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
squares = [x**2 for x in range(10)]
squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Exercices

- Échanger les premier et dernier éléments d'une liste.
- Étant donné une liste et un élément, compter le nombre d'occurrences de cet élément dans la liste.
- Étant donné une liste et un élément, enlever toutes les occurrences de cet élément dans la liste.
- Afficher tous les entiers pairs contenus dans une liste
- Compter le nombre d'entiers pairs et impairs dans une liste donnée



Les tuples (Tuple)

■ Constructeur

- tuple = ()

■ Propriétés

- sont identiques aux listes sauf **non-mutable**
- L'instruction tuple() gèle une liste et l'instruction list() dégèle un tuple
- Peuvent contenir des valeur mutables

■ Utilité

- Plus rapide que la liste (quand la longueur est significative)
- Protégé en écriture : plus de sécurité

```
Python >>>
>>> t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
>>> t
('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t[0]
'foo'
>>> t[-1]
'corge'
>>> t[1::2]
('bar', 'qux', 'corge')
```



Les ensembles (set)

- Collection **non-ordonnée** sans double

- **Constructeur**

- { } ou set()

```
basket = {'pomme', 'orange', 'pomme', 'poire', 'orange', 'banane'}
print(basket) # doublons enlevés

{'poire', 'banane', 'orange', 'pomme'}
```



```
'ananas' in basket
```



```
False
```



```
set([1,2,2,3,5,5])
```



```
{1, 2, 3, 5}
```

- **Utilité**

- Enlever des doublons dans une liste
 - Énumérer sur ensemble désordonné



Les dictionnaires (dict)

Constructeurs :

- o $d = \{ \}$

```
Python
```

```
d = {  
    <key>: <value>,  
    <key>: <value>,  
    .  
    .  
    .  
    <key>: <value>  
}
```

```
Python
```

```
>>> MLB_team = {  
...     'Colorado' : 'Rockies',  
...     'Boston'    : 'Red Sox',  
...     'Minnesota': 'Twins',  
...     'Milwaukee': 'Brewers',  
...     'Seattle'   : 'Mariners'  
... }
```

```
Python
```

```
d = dict([  
    (<key>, <value>),  
    (<key>, <value>),  
    .  
    .  
    .  
    (<key>, <value>)  
])
```

```
Python
```

```
>>> MLB_team = dict([  
...     ('Colorado', 'Rockies'),  
...     ('Boston', 'Red Sox'),  
...     ('Minnesota', 'Twins'),  
...     ('Milwaukee', 'Brewers'),  
...     ('Seattle', 'Mariners')  
... ])
```

- o Ou $d = \text{dict}([])$

Les dictionnaires (dict)

■ Propriétés

- Non ordonnés (\neq list)
- Peuvent contenir des éléments de différents types
- Les éléments sont accessibles via key
- Valeurs mutables (\neq keys)
- Peuvent être imbriqués

■ Restrictions sur les clés de dictionnaire

- **Unicité**
- **Non mutable** : atomique (integer, float, string, Boolean) ou tuple (pas de liste ou dictionnaire)

■ Pas de restrictions sur les éléments de dictionnaire

```
people_dict = {  
    'alan': {'age': 30, 'height': 1.72},  
    'monica': {'age': 31, 'height': 1.65}  
}  
  
people_dict = {  
    'alan': {'age': 30, 'height': 1.72},  
    'monica': {'age': 31, 'height': 1.65}  
}  
print(people_dict['monica'])  
print(people_dict['monica']['age'])  
people_dict['alan'] = {'age': 32, 'height': 1.72}  
people_dict['alan']['age'] = 32  
  
{'age': 31, 'height': 1.65}  
31
```



Les dictionnaires (**dict**) : opérateurs

- in / not in (**for key**)
- len(), min(), max()
- concatenation (+) et replication (*)
- Vider : d.clear()
- Accéder : d[key]
- Parcourir : d.items(), d.keys(), d.values()
- Modifier : d[key] = , d.update(<obj = dictionary>) ou del d[key]

```
print('alan' in people_dict)
print('age' in people_dict)
print(len(people_dict))
print(min(people_dict))
print(max(people_dict))
```

True
False
2
alan
monica

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.items():
    print(k, v)
```

```
gallahad the pure
robin the brave
```



Exercices

- Compter le nombre de valeurs distinctes dans une liste.
- Afficher les clés d'un dictionnaire et les types de variables correspondantes
- Sommer les valeurs des variables entières dans un dictionnaire





04

Bases du langage (Partie 2)

```
def fibonacci(x):
    a,b = 0,1
    if x < 0:
        return "x doit être un entier positif"
    for i in range(x):
        a, b = b, a+b
    return a
```

Indentation

- Synonyme de décalage (4 espaces ou une tabulation). Le fait d'indenter les lignes permet de définir une dépendance d'une instruction ou un bloc de instructions par rapport à celle d'avant.
- On utilise l'indentation pour instructions de contrôle (**if, while, for**) ainsi que la définition (**def**)
- Ne mélangez pas les espaces et les tabulations.



Instruction for

```
for itérateur in séquence:  
    bloc d'instructions
```

Séquence est une collection de valeurs

```
for s in ('hello', 'world'):  
    print(s)
```

```
hello  
world
```

```
for s in ['hello', 'world']:  
    print(s)
```

```
hello  
world
```

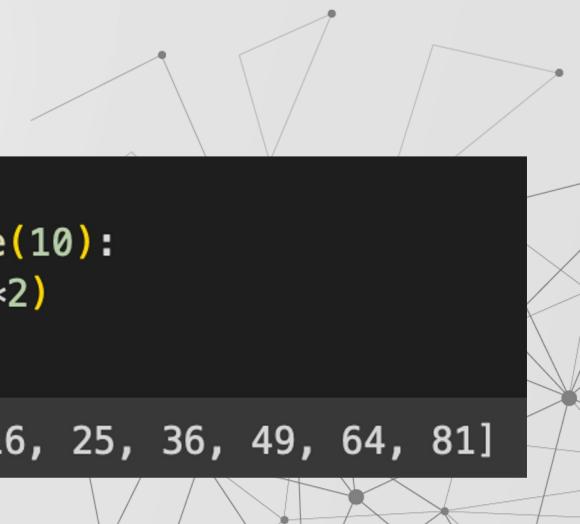
- Boucle for ne s'applique que sur une collection de valeurs (ex. tuples, listes, strings...)
- Itérerer sur une suite de nombres : range()

```
for s in 'hello':  
    print(s)
```

```
h  
e  
l  
l  
o
```

```
x = []  
for y in range(10):  
    x.append(y**2)  
print(x)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



Instruction if

```
if condition:  
    Instruction A  
else:  
    Instruction B  
  
if condition:  
    Instruction A  
elif condition:  
    Instruction B  
else:  
    Instruction C
```

- Nombre quelconque de elif
- Partie else = facultative



Instructions while et break

```
while condition:  
    bloc d'instructions
```

Opération de comparaison
(résultat : booléen)

```
x = 0  
while x < 3:  
    print(x)  
    x = x + 1
```



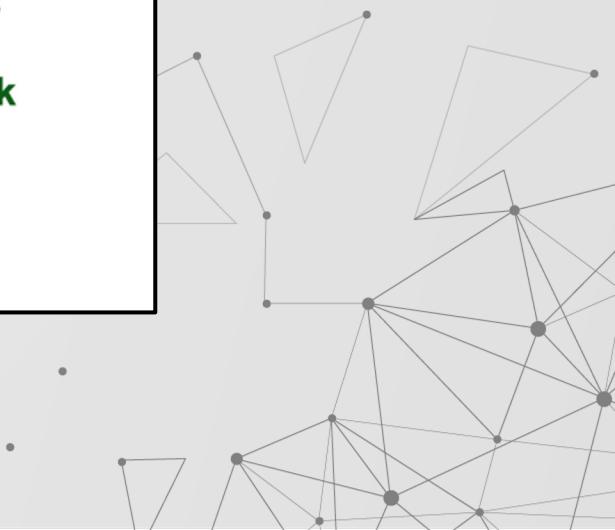
```
0  
1  
2
```

« break » permet de sortir
d'une boucle for ou while

```
>>> x = 0  
>>> while x < 3:  
...     if x == 2:  
...         break  
...     print(x)  
...     x = x + 1
```

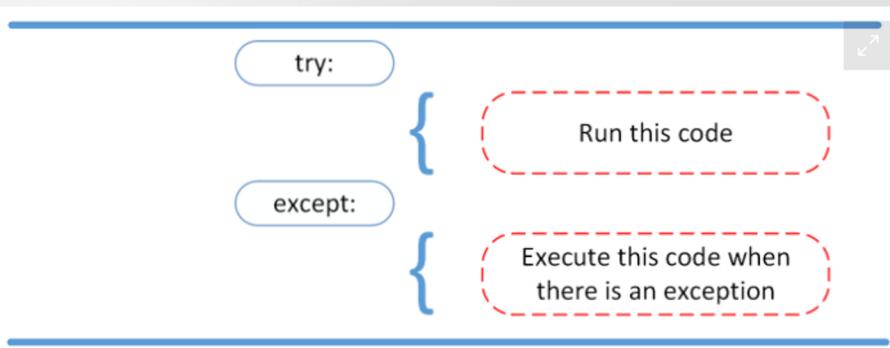


```
0  
1
```



Exceptions

- Déetecter des erreurs lors de l'exécution d'une fonction lorsque la syntaxe est correcte = exceptions
- Block **try** / **except** utilisé pour rattraper et gérer les exceptions
- try : code à exécuter
“normalement: si pas d'erreur
- except : à exécuter si erreur
- Type d'erreur peut être spécifié



Attention à l'indentation !

Interpréteurs renvoient parfois une erreur quand
erreur d'indentation grossière, mais...

```
x = 10
if x == 10:
    x = x * 2
    x += 1
elif x == 20:
    x = x * 3
x += 2
x
```



Exercices

- Initialisez deux entiers : $a = 0$ et $b = 10$. Écrire une boucle affichant et incrémentant la valeur de a tant qu'elle reste inférieure à celle de b .
- Écrire une autre boucle décrémentant la valeur de b et affichant sa valeur si elle est impaire, tant que b reste positif.
- Afficher les entiers de 0 à 15, de trois en trois, en utilisant (i) une boucle `for` et (ii) l'instruction `range()`
- Créer la liste `names = ['Tom', 'Lea', 'Martin', 'Sylvain', 'Philippe']` puis, à partir de cette liste, une nouvelle liste `long_names` qui ne contient que les noms de plus de 3 lettres.

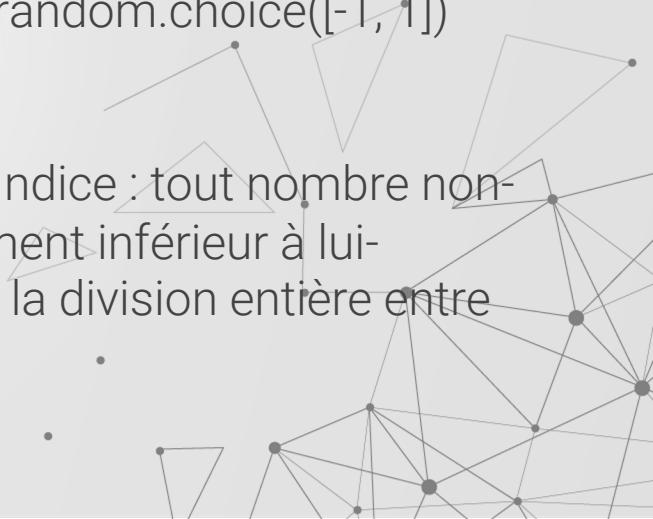
Exercices

- Imaginez une puce, qui se déplace aléatoirement sur une ligne, d'un bon en avant ou en arrière (+/- 1) à chaque pas de temps. Par exemple, si elle est à l'emplacement 2, elle peut se rendre en 1 ou en 3.

Simulez avec une boucle while le parcours de cette puce partant de l'emplacement 0, en s'arrêtant lorsqu'elle atteint l'emplacement -5 ou 5. Afficher combien de sauts sont nécessaires à ce parcours.

Indice : utilisez le package random et l'instruction « `random.choice([-1, 1])` pour générer un nombre aléatoire entre -1 et 1.

- Déterminez les nombres premiers inférieurs à 100. Indice : tout nombre non-premier est divisible par un nombre premier strictement inférieur à lui-même. Utiliser l'opérateur % pour obtenir le reste de la division entière entre deux entiers.



Fonctions en Python

- Une fonction est un objet qui prend en entrée des arguments, effectue des opérations et (éventuellement) renvoie un objet en retour.
- La syntaxe Python pour la définition d'une fonction est la suivante :

```
def my_function(param1, param2):  
    instruction1  
    instruction2  
    instruction3  
    ...  
    result = ...  
    return result
```

- Mots clés = **def** et **return**
- **return** optionnel (renvoie None)



Paramètres ou Arguments

Paramètres (formels)

noms que l'on spécifie dans la signature de la fonction (paramètres)

```
def ma_fonction(param1, param2)
```

Arguments (paramètres effectifs)

valeurs passées à l'appel de la fonction

```
ma_fonction(ar1, ar2)
```

En résumé

On définit des paramètres formels (parameters) et on passe des paramètres effectifs (arguments)



Paramètres par défaut

- Possible de spécifier des valeurs *par défaut* pour les paramètres d'une fonction
- Si renseignés à l'appel de la fonction : prennent la valeur renseignée, sinon valeur par défaut

```
def exponent(x: float, expnt: int = 2) -> float:  
    ...  
    Returns x to the power of expnt. By default returns x**2.  
    ...  
    return x**expnt
```

- Arguments par défaut obligatoirement renseignés **après** les arguments non nommés

Exercices

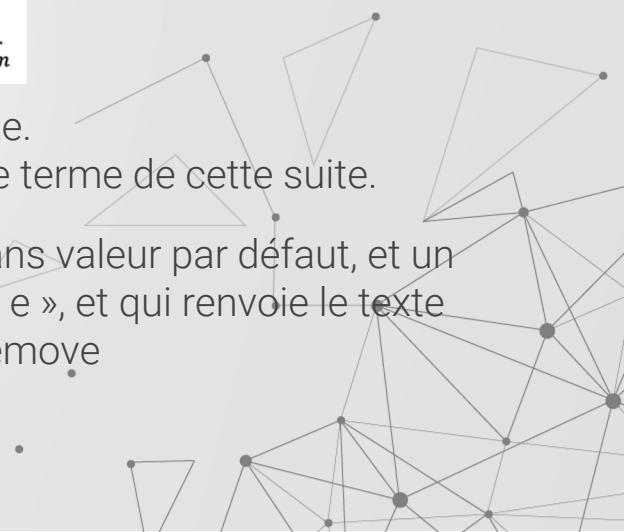
- Écrire une fonction « cube » qui retourne le cube de son argument.
- Écrire une fonction dont les paramètres sont x et n et qui renvoie x à la puissance de n .
- Écrire une fonction qui prend une liste d'entiers L et renvoie le maximum de cette liste
- Écrire une fonction `check_if_palindrome` qui détermine si une chaîne de caractères est un palindrome
- La suite de Fibonacci est définie récursivement par

$$\begin{aligned}f_0 &= 0 \\f_1 &= 1 \\f_{n+2} &= f_{n+1} + f_n\end{aligned}$$

Écrire une fonction `fibo` qui calcule le n -ième terme de cette suite.

Écrire une fonction récursive `fibo_recursive` qui calcule le n -ième terme de cette suite.

- Écrire une fonction « `disappear` » qui prend un argument `text`, sans valeur par défaut, et un argument `char_to_remove`, avec valeur par défaut le caractère « `e` », et qui renvoie le texte du premier argument auquel on a enlevé le caractère `char_to_remove`



Exercices

- Écrire une fonction qui, étant donné une liste d'entiers, enlève les entiers pairs de cette liste et renvoie le résultat
- Écrire une fonction qui, étant donnée une phrase, renvoie les entiers présents dans cette phrase. Indice : une chaîne de caractères peut être divisée en mots avec la méthode `.split()`. Utiliser `.isnumeric()` pour vérifier si une chaîne de caractères représente un entier.



Modules, packages, import

- Lorsque l'on regroupe des **fonctions** dans un fichier on crée un ensemble de fonctions que l'on nomme "**module**".
- Un module peut être importé dans un autre fichier en spécifiant « import module »
- Une classe est un module avec objet (diapo suivant)
- Lorsque l'on cherche à regrouper des modules sous un répertoire, on parle de **package**.
 - **Syntaxes :**
 - **from package import module** → **module.fonction**
 - **from package.module import fonction** → **fonction**
 - **Appel fonction : à partir de l'élément après « import »**
- **Python importe toujours les modules / une partie de module** (si import package directement, il faut préciser quels sont les modules importés dans `__init__.py`, comme pour pandas)



05

Manipulation de données avec Pandas



Package pandas

pandas est un package Python **spécialisé** pour l'analyse des données, spécifiquement :

- Les **bases de données tabulaires** : matrices 2D avec labels
- Les **séries temporelles**

spécialisé = tout est prévu par Python pour facilement **stocker, manipuler** et **visualiser** ce genre de données.

- Implémentée à partir de C – donc très rapide
- Bonne intégration avec autres librairies comme matplotlib et scikit-learn

→ une des librairies **les plus utilisées** pour **la science des données**.



Les types pandas

Il y a deux objets importants dans pandas :

- **pandas.DataFrame** pour représenter n'importe quelle base de données tabulaire.
- **pandas.Series** pour représenter des séquences unidimensionnelles (souvent une colonne ou une ligne d'un DataFrame)

```
[ ]: import pandas as pd
```

Package pandas

Pandas permet de faire un grand nombre d'opérations/transformations sur DataFrame et Series :

- Sélectionner une partie des données (lignes, colonnes)
- Appliquer des filtres
- Reformater les données
- Joindre des séries/dataframes
- Agréger les données
- Visualiser les données
- Calculer des statistiques
- Écrire/lire des fichiers csv/txt/xlsx
- Et bien d'autres !



Importation de données

- Fichier CSV

```
df = pd.read_csv("diabetes.csv")
```

- Fichier texte (préciser le séparateur avec sep=...)

```
df = pd.read_csv("diabetes.txt", sep="\s")
```

- Excel (.xls ou .xlsx ; possible de spécifier la feuille avec sheet_name=...)

```
df = pd.read_excel('diabetes.xlsx')
```



Importation de données

Les options		read_csv()	read_table()	read_excel()
		Par défaut :		
header	= 0 si l'entête contient les noms de colonnes = None si contraire	0	0	0
sep	= séparateur de champs	', '	'\t'	---
names	= ['1', '2', '3'].. ↗ liste contenant le nom des colonnes	(header=0) ['1', '2', '3']..	Idem	Idem
index_col	= étiquette des lignes	None	Idem	Idem
dec	= caractère utilisé pour les décimales	'.'	Idem	Idem
na_values	= valeur des données manquantes	'NaN'	Idem	Idem
nrows	= nb max de lignes à lire	Tout	Idem	Idem
skiprows	= nb de lignes à sauter avant de commencer la lecture	0	Idem	Idem
performance		Rapide	Idem	Lent

Importation de données

Python For Data Science Importing Data Cheat Sheet

Learn Python online at www.DataCamp.com

> Importing Data in Python

Most of the time, you'll use either NumPy or pandas to import your data:

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

> Help

```
>>> np.info(np.ndarray.dtype)
```

```
>>> help(pd.read_csv)
```

> Text Files

Plain Text Files

```
>>> filename = "huck.txt"
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(text) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Using the context manager with

```
>>> with open("huck.txt", "r") as file:
...     print(file.readline(), end='') #Read a single line
...     print(file.readline())
...     print(file.readline())
...     print(file.readline())
```

Table Data: Flat Files

Importing Flat Files with NumPy

```
>>> filename = "huck.txt"
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(text)
>>> file.close() #Close file
>>> print(text)
```

Files with one data type

```
>>> filename = "enrich1.txt"
>>> data = np.loadtxt(filename,
...                   delimiter=',', #String used to separate values
...                   skiprows=2, #Skip the first 2 lines
...                   usecols=(1,2), #Read the 1st and 2nd column
...                   unpack=True) #Type of the resulting array
```

Files with mixed data type

```
>>> filename = "titles.csv"
>>> data = np.genfromtxt(filename,
...                      delimiter=',', #String used to separate values
...                      names=True, #Look for column header
...                      dtype=None)
>>> data_array = np.recfromcsv(filename)
>>> data_array.dtype #The default dtype of the np.recfromcsv() function is None
Importing Flat Files with Pandas
>>> filename = "titles.csv"
>>> data = pd.read_csv(filename)
```

```
>>> data = pd.read_csv(filename, header=0, #Number of rows of file to read
...                     header=None, #Row number to use as col names
...                     sep=',', #Delimiter to use
...                     comment='#', #String to split comments
...                     na_values=['-']) #String to recognize as NA/NAN
```

> Exploring Your Data

NumPy Arrays

```
>>> np.array.dtype #Get type of array elements
>>> data_array.shape #Array dimensions
>>> len(data_array) #Length of array
```

Pandas DataFrames

```
>>> df = pd.DataFrame({'first_name': 'Aamir', 'last_name': 'Khan', 'age': 25, 'city': 'London', 'country': 'UK'}, index=[0])
>>> df[0:2] #Return first 2 rows
>>> df.describe() #Describe DataFrame columns
>>> df.info() #Information on DataFrame
>>> df_array = data_array.convert(DataFrame to a NumPy array)
```

SAS File

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('huck.sas7bdat') as file:
...     df_sas = file.to_dataframe()
```

Stata File

```
>>> data = pd.read_stata("urbanpop.dta")
```

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet1 = data.parse('Sheet1',
...                         skiprows=0,
...                         names=['Country',
...                                'Year', 'Pop(2002)'])
>>> df_sheet1 = data.parse(0,
...                         parse_cols=[0],
...                         skiprows=0,
...                         names=['Country'])
To access the sheet names, use the sheet_names attribute:
>>> data.sheet_names
```

> Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///huck.sqlite')
Use the table_names() method to fetch a list of table names:
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> rs.close()
>>> con.close()
```

```
Using the context manager with
>>> with engine.connect() as con:
...     rs = con.execute("SELECT OrderID FROM Orders")
...     df = pd.DataFrame(rs.fetchmany(size=5))
...     df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

> Pickled Files

```
>>> import pickle
>>> with open('fruit.pkl', 'rb') as file:
...     pickled_data = pickle.load(file)
```

> Matlab Files

```
>>> import scipy.io
>>> filename = 'huck.hdf5'
>>> mat = scipy.io.loadmat(filename)
```

> HDF5 Files

```
>>> import h5py
>>> filename = 'huck.hdf5'
>>> data = h5py.File(filename, 'r')
```

> Exploring Dictionaries

Querying relational databases with pandas

```
>>> print(data.keys()) #Print dictionary keys
For key in data.keys(): #Print dictionary keys
    print(key)
meta
quality
strain
print(data.values()) #Return dictionary values
print(data.items()) #Returns items in list format of (key, value) tuple pairs
```

Accessing Data Items with Keys

```
>>> for key in data['meta'].keys(): #Explore the HDF5
...     print(key)
...     Description
...     DescriptionURL
...     DescriptionDoc
...     Duration
...     Dataset
...     Observatory
...     Type
...     Value
...     #Retrieve the value for a key
...     print(data['meta'][key].value)
```

> Navigating Your FileSystem

Magic Commands

```
!ls #List directory contents of files and directories
cd .. #Change current working directory
pwd #Return the current working directory path
```

OS Library

```
>>> import os
>>> path = "/usr/local"
>>> os.listdir(path) #List contents of the directory in a list
>>> os.getcwd() #Change current working directory
>>> os.rename("test1.txt", "test2.txt")
>>> os.remove("test1.txt") #Delete an existing file
>>> os.mkdir("newdir") #Create a new directory
```

Learn Data Skills Online at www.DataCamp.com

<https://www.datacamp.com/cheat-sheet/importing-data-in-python-cheat-sheet>

Exportation de données

- Fichier CSV

```
df.to_csv("diabetes_out.csv", index=False)
```

- Fichier texte

```
df.to_csv('diabetes_out.txt', header=df.columns, index=None, sep=' ')
```

- Excel

```
df.to_excel("diabetes_out.xlsx", index=False)
```

Pandas : DataFrame

	Name	Age	Salary	Department
0	Tom	20	50000	HR
1	Nick	21	60000	Engineering
2	John	19	55000	Marketing
3	Tom	20	70000	HR

Rows indexes → Column Names

Rows → Columns

Pandas : DataFrame

Bon tutoriel : <https://www.datacamp.com/tutorial/pandas>

Voir aussi : https://pandas.pydata.org/pandas-docs/stable/user_guide/

Création d'un DataFrame

```
# Empty DataFrame
df = pd.DataFrame(data=None, index=None, columns=None, dtype=None)

# Non-empty DataFrame
df = pd.DataFrame(data=np.array([[3, 2, 0, 1], [0, 3, 7, 2]]).T, columns=['apples', 'oranges'])

# À partir d'un dictionnaire
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
df = pd.DataFrame(data)
```

Structure d'un DataFrame

```
type(df) # pandas.core.frame.DataFrame
df.dtypes # type of dataframe columns
df.ndim # Nb de dimensions
df.shape # nb de lignes et colonnes
df.size # nb total de valeur (lignes * colonnes)
```

Afficher et renommer

```
df.head() # Affiche les quelques premières lignes
df.tail() # Affiche les quelques dernières lignes
df.rename(columns={"apples": "a", "oranges": "o"})
```

Pandas : DataFrame

Statistique descriptive

```
# Toutes opérations faites sur chaque colonne
df.sum() # somme
df.count() # nb de valeurs non-NaN/null
df.median() # médiane
df.quantile([0.25, 0.75]) # quantiles
df.mean() # moyenne
df.min() # minimum
df.max() # maximum
df.var() # variance
df.std() # écart-type
df.describe() # statistiques descriptives (cf. 'summary' en R)
```

Sélectionner colonnes/lignes

```
# Sélectionner une colonne
df['apples']
df.apples # de type pandas.core.series.Series

# Ajouter une nouvelle colonne
df['bananas'] = [5, 1, 3, 0]

# Sélectionner une ligne par son indice de ligne
df[df.index == 1]
df.loc[1]

# Plusieurs lignes
df[df.index.isin([0, 2, 3])]
df[df.index.isin(range(2,4))]

# En utilisant iloc (utile si les indices sont complexes)
df.iloc[1]
df.iloc[1:4]
df.iloc[[0, 2, 3]]
```



Pandas : DataFrame

Filtrer avec booléen

```
df[COND boolean]  
# COND boolean : colonne vérifie ou  
non certaine(s) condition(s)  
[] : condition  
= df.loc [COND boolean, :]
```

```
# Sélection avec condition  
print(df[df.apples >= 3])  
print(df.loc[df.apples >= 3])  
  
   apples  oranges  bananas  
0       3        0        5  
   apples  oranges  bananas  
0       3        0        5
```

Négation

```
df[~COND boolean]
```

Combinaison des conditions (utiliser des parenthèses)

```
df[ (COND boolean 1) & (COND boolean 2)]  
df[COND boolean 1 | COND boolean 2]
```

COND dans une liste

```
.isin([ ])
```

COND comparaison des strings

```
str.contains()  
str.startswith()  
str.endswith()
```



Pandas : DataFrame

Traitement des valeurs manquantes

- Vérifier s'il y a des données manquantes
`df.isnull()`
`df.isnull().any()`
`df.isnull().sum()`
- Supprimer les enregistrement avec des valeurs manquantes : `dropna()`
`df.dropna()`
`df.dropna(, inplace = True)`
- Replacer les valeurs manquantes : `fillna()`
`df.fillna(value = , inplace = True)`

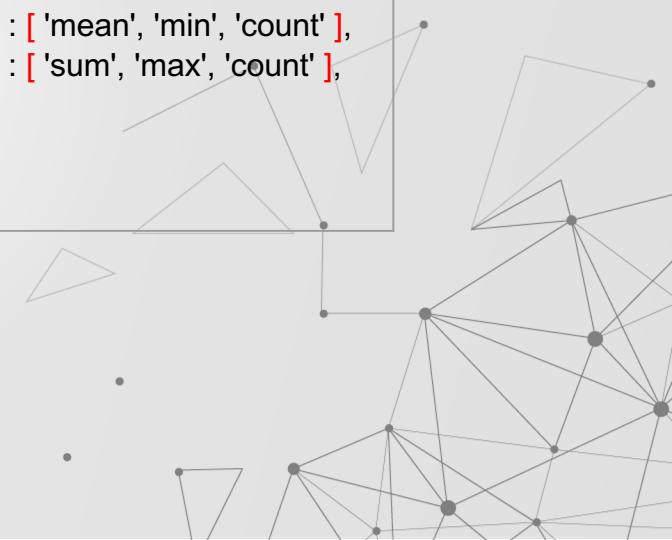


Pandas library : DataFrame

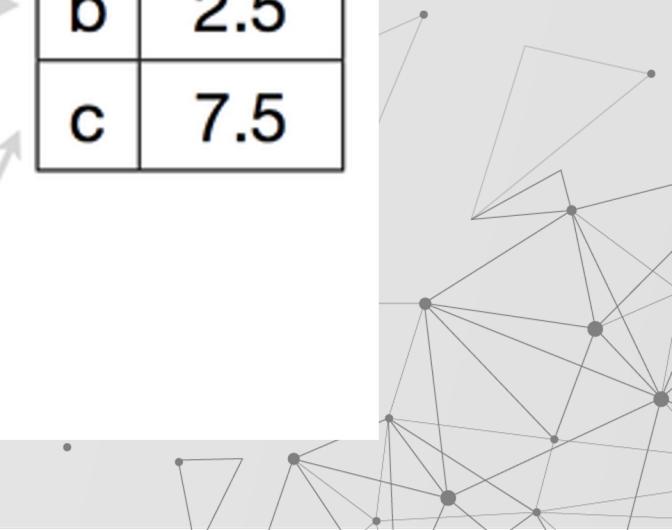
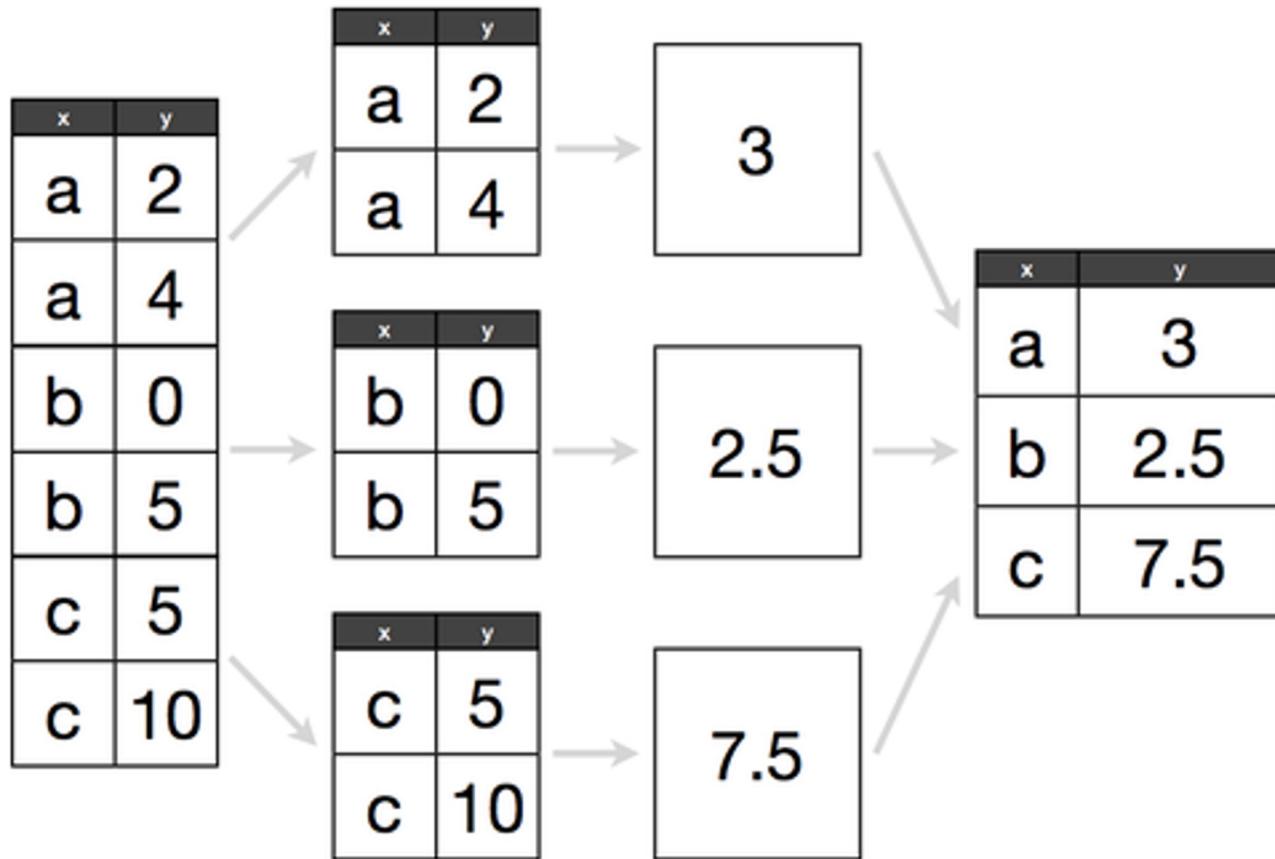
Divers	<p>Apply a function along an axis of the DataFrame apply(func, axis=0) # 0 = colonne; 1 = ligne # attention à l'écriture de function = label "</p> <p>créer une nouvelle colonne assign()</p> <p>créer un compteur par valeur de col1 groupby('col1').cumcount()</p> <p>créer un échantillon aléatoire à partir d'un DF sample(frac =)</p>
Suppression d'une ligne / colonne	<p>df.drop(df[cond bool].index , inplace=True)</p> <p>df.drop('column_name', 1)</p>



DataFrame.groupby()

Agrégat	DataFrame	Regroupement	Colonne(s)	Opération(s) = fonction(s)
df		.groupby()	.Col1 ou ['Col1']	.mean() . agg(['mean', 'min', 'count'])
		.groupby([,])	[['Col1', 'Col2', ...]]	. agg({ 'Col1' : ['mean', 'min', 'count'], 'Col2' : ['sum', 'max', 'count'], ... })
				

□ : indexation
■ : liste





TD pandas



« Cheat sheets »

<https://www.utc.fr/~jlaforet/Suppl/python-cheatsheets.pdf>

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Et nombreux sites web...

<https://www.geeksforgeeks.org/python-pandas-dataframe/>

<https://www.datacamp.com/tutorial/pandas>

<https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>

06

Packages Matplotlib et Seaborn Python pour la data visualisation



Étape 1 : Choisir le type de graphique et poser la base

Étape 2 : Personnaliser => obtenir une figure

(Apparence : couleur, taille...)

Étape 3 : Mise en page => gérer le cadre

(Titre, axes...)



Matplotlib

- Principale bibliothèque graphique en Python sur laquelle sont construites les fonctionnalités de pandas, seaborn etc.
- Générer la plupart des graphes classiques
- Assez complète mais syntaxe et fonctionnement parfois complexes
- Instructions explicites (\neq seaborn, xarray etc.) mais grande flexibilité



Matplotlib : les objets principaux

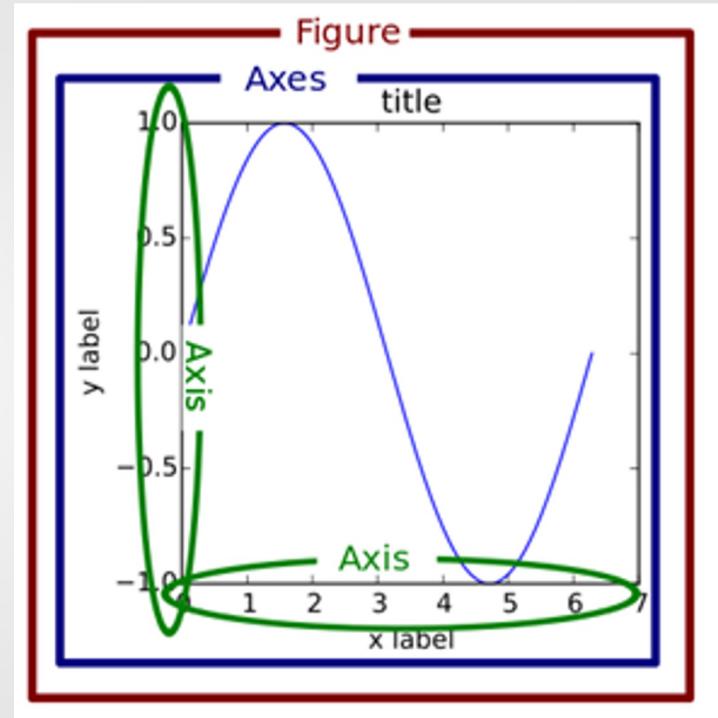
Quelques termes :

* Figure

Objet de niveau supérieur qui contient tout ce qui peut être « graphé ».

* Axes

Surface sur laquelle on peut grapher quelque chose. Il est possible de placer un Axes à n'importe quelle position. Axes est l'objet sur lequel va être dessiné le graphique



Matplotlib : premier exemple

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

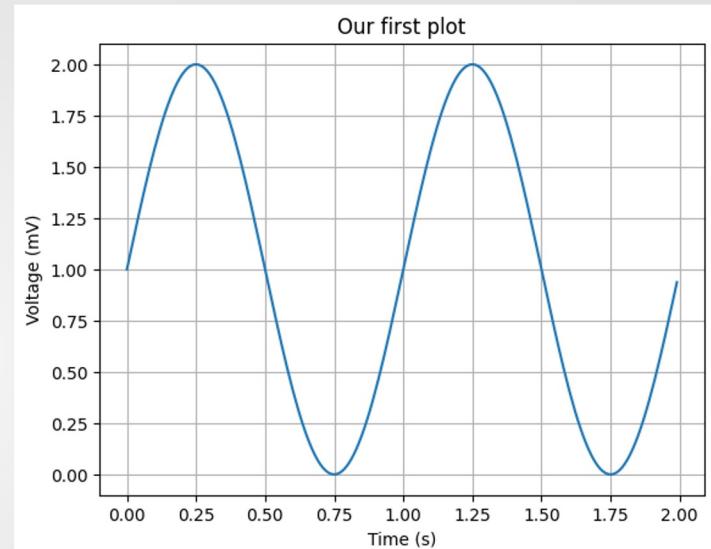
x = np.arange(0.0, 2.0, 0.01)
y = 1 + np.sin(2 * np.pi * x)

plt.plot(x, y)
plt.xlabel('Time (s)')
plt.ylabel('Voltage (mV)')
plt.title('Our first plot')
plt.grid(True)
# plt.savefig("test.png" # si on veut enregistrer la figure
plt.show()
```

Gérer les axes

Gérer le titre

Instructions de
plot



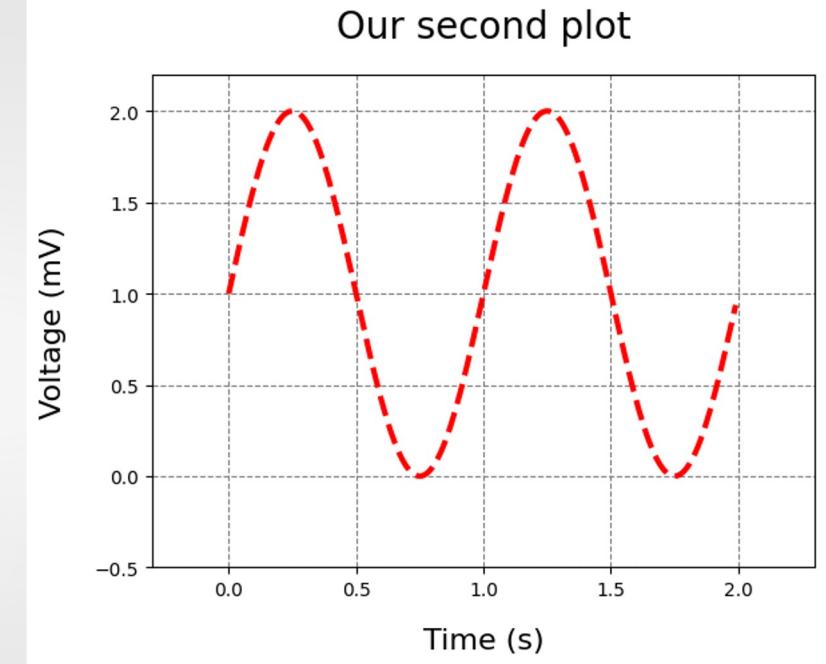
Contrôler les axes de la figure

Apparence de la courbe

```
plt.plot(x, y, linestyle='--', color='red', linewidth=3)
plt.xlabel('Time (s)', fontsize=16, labelpad=15)
plt.ylabel('Voltage (mV)', fontsize=16, labelpad=15)
plt.xlim([-0.3, 2.3])
plt.ylim([-0.5, 2.2])
plt.title('Our second plot', fontsize=20, pad=20)
plt.grid(True, linestyle='--', color='gray')
plt.show()
```

Options pour labels

Étendue des axes

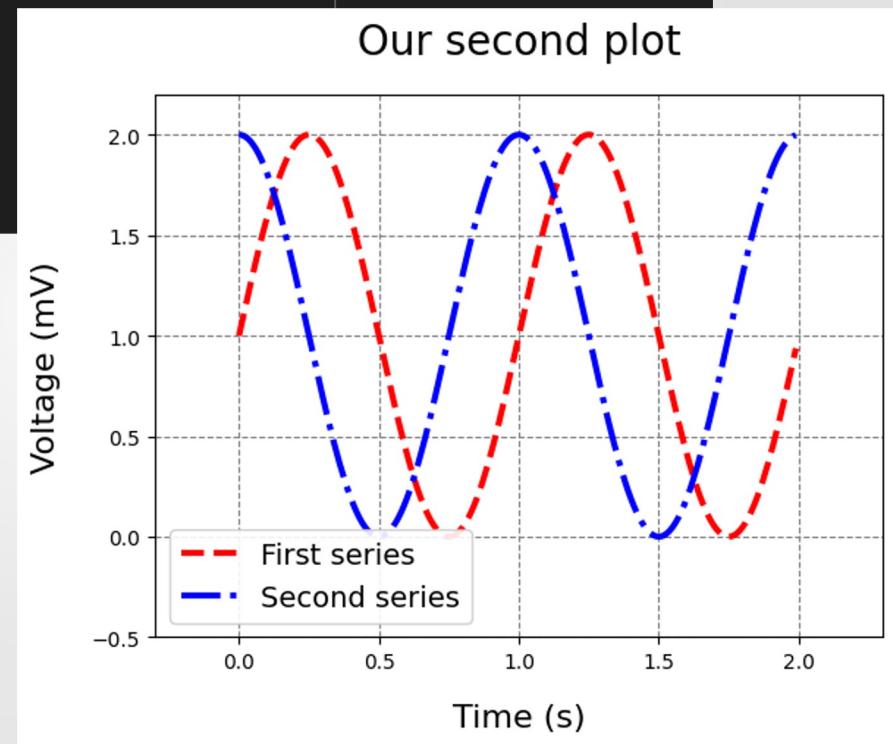


Ajouter une légende

```
plt.plot(x, y, linestyle='--', color='red', linewidth=3, label='First series')
plt.plot(x, 1 + np.cos(2 * np.pi * x), linestyle='-.', color='blue', linewidth=3, label='Second series')
plt.xlabel('Time (s)', fontsize=16, labelpad=15)
plt.ylabel('Voltage (mV)', fontsize=16, labelpad=15)
plt.xlim([-0.3, 2.3])
plt.ylim([-0.5, 2.2])
plt.title('Our second plot', fontsize=20, pad=20)
plt.grid(True, linestyle='--', color='gray')
plt.legend(loc='lower left', fontsize=14, framealpha=0.9)
plt.show()
```

Position
dans
le cadre

Transparenc
e

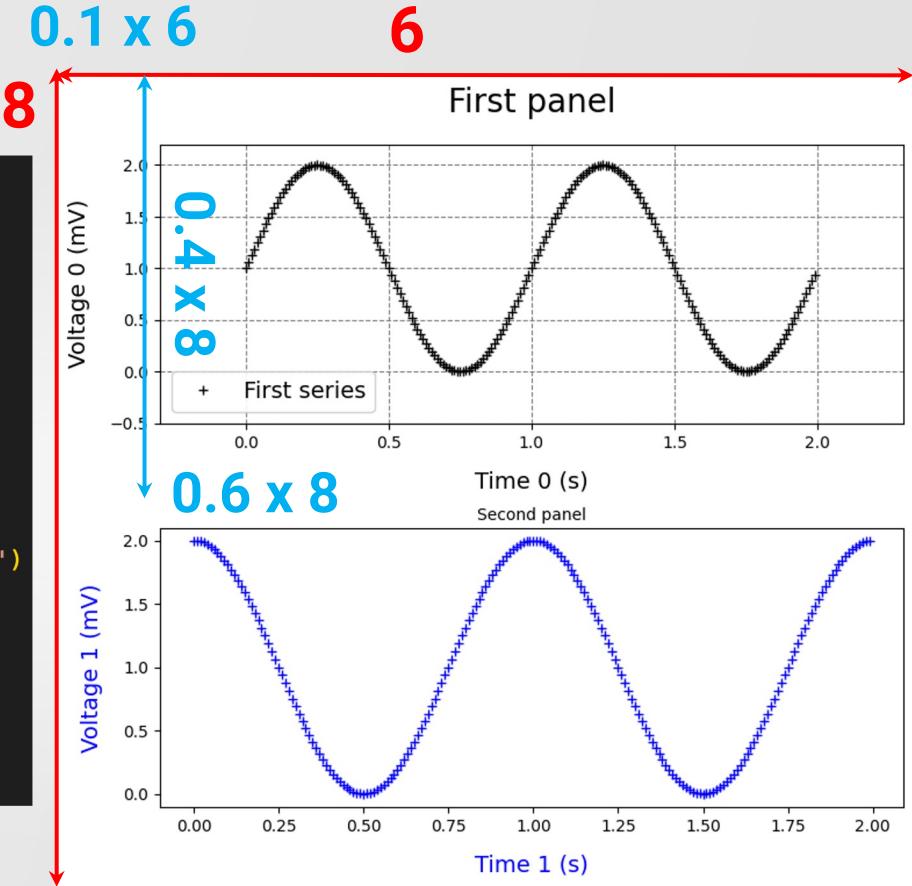


Axes multiples : explicites

Dimensions absolues de la figure

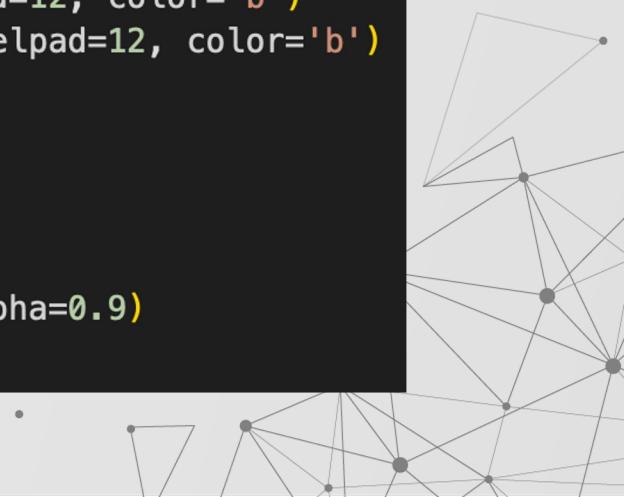
Taille des axes (fraction)

```
# Crée un objet figure
fig = plt.figure(figsize=(8, 6))
# Créons des axes explicites
ax0 = fig.add_axes([0.1, 0.6, 0.8, 0.4])
ax0.plot(x, y, 'k+', label='First series')
ax1 = fig.add_axes([0.1, 0.05, 0.8, 0.4])
ax1.plot(x, 1 + np.cos(2 * np.pi * x), 'b+', label='Second series')
# Les propriétés de chaque axe peuvent être définies
# indépendamment.
ax0.set_xlabel('Time 0 (s)', fontsize=14, labelpad=12)
ax0.set_ylabel('Voltage 0 (mV)', fontsize=14, labelpad=12)
ax1.set_xlabel('Time 1 (s)', fontsize=14, labelpad=12, color='b')
ax1.set_ylabel('Voltage 1 (mV)', fontsize=14, labelpad=12, color='b')
ax0.set_xlim([-0.3, 2.3])
ax0.set_ylim([-0.5, 2.2])
ax0.set_title('First panel', fontsize=20, pad=20)
ax1.set_title('Second panel', fontsize=10, pad=5)
ax0.grid(True, linestyle='--', color='gray')
ax0.legend(loc='lower left', fontsize=14, framealpha=0.9)
plt.show()
```



Axes multiples : subplots()

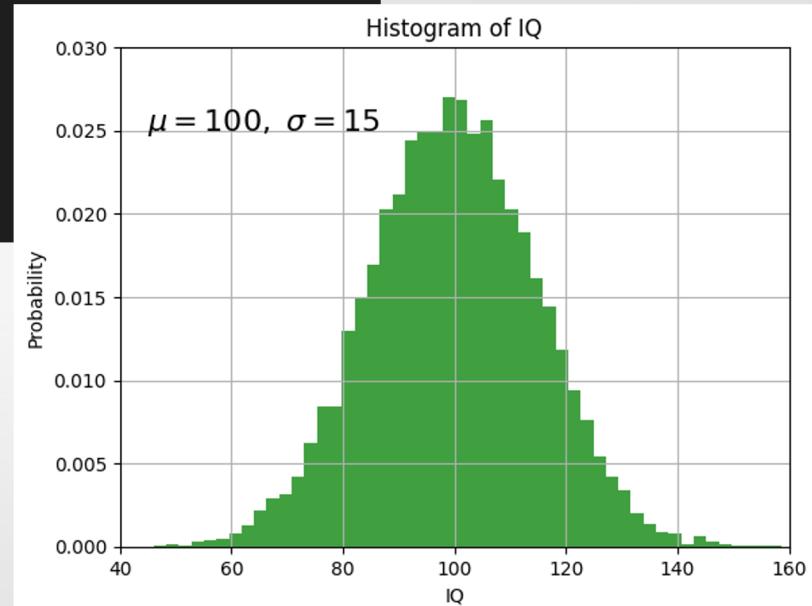
```
# Créer un objet figure et les subplots associés
fig, axes = plt.subplots(2, 1, figsize=(8, 6))
# Nombre de lignes, nombre de colonnes
axes[0].plot(x, y, 'k+', label='First series')
axes[1].plot(x, 1 + np.cos(2 * np.pi * x), 'b+', label='Second series')
axes[0].set_xlabel('Time 0 (s)', fontsize=14, labelpad=12)
axes[0].set_ylabel('Voltage 0 (mV)', fontsize=14, labelpad=12)
axes[1].set_xlabel('Time 1 (s)', fontsize=14, labelpad=12, color='b')
axes[1].set_ylabel('Voltage 1 (mV)', fontsize=14, labelpad=12, color='b')
axes[0].set_xlim([-0.3, 2.3])
axes[0].set_ylim([-0.5, 2.2])
axes[0].set_title('First panel', fontsize=20, pad=20)
axes[1].set_title('Second panel', fontsize=10, pad=5)
axes[0].grid(True, linestyle='--', color='gray')
axes[0].legend(loc='lower left', fontsize=14, framealpha=0.9)
plt.show()
```

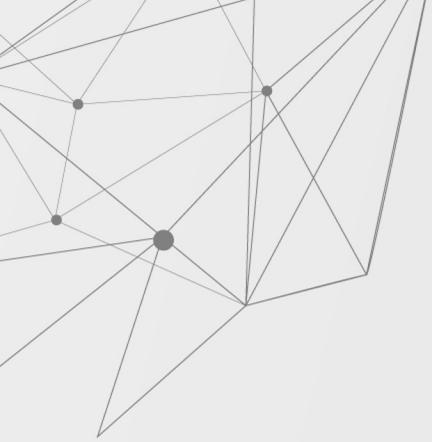


Graphes type : histogrammes

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

n, bins, patches = plt.hist(x, bins=50, facecolor='g', alpha=0.75, density=True)
plt.xlabel('IQ')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(45, .025, r'$\mu=100,\ \sigma=15$', fontsize=16)
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```







TD : visualisation



Annexes

Résumé : structure d'un programme Python type

```
# -*- coding:Utf8 -*-

#####
# Programme Python type #
# auteur : G.Swinnen, Liège, 2009 #
# licence : GPL #
#####

#####
# Importation de fonctions externes :
from math import sqrt

#####
# Définition locale de fonctions :
def occurrences(car, ch):
    "Cette fonction renvoie le \
     nombre de caractères <car> \
     contenus dans la chaîne <ch>"
    nc = 0
    i = 0
    while i < len(ch):
        if ch[i] == car:
            nc = nc + 1
        i = i + 1
    return nc

#####
# Corps principal du programme :
print("Veuillez entrer un nombre :")
nbr = eval(input())
print("Veuillez entrer une phrase :")
phr = input()
print("Entrez le caractère à compter :")
cch = input()
no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print("La racine carrée du cube", end=' ')
print("du nombre fourni vaut", end=' ')
print(rc)

print("La phrase contient", end=' ')
print(no, "caractères", cch)
```

Un programme Python contient en général les blocs suivants, dans l'ordre :

- Quelques instructions d'initialisation (importation de fonctions et/ou de classes, définition éventuelle de variables globales).
- Les définitions locales de fonctions et/ou de classes.
- Le corps principal du programme.

Le programme peut utiliser un nombre quelconque de fonctions, lesquelles sont définies localement ou importées depuis des modules externes. Vous pouvez vous-même définir de tels modules.

La définition d'une fonction comporte souvent une liste de PARAMÈTRES. Ce sont toujours des VARIABLES, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit toujours inclure au moins quatre éléments :

- l'initialisation d'une variable 'compteur';
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent;
- le bloc d'instructions à répéter;
- une instruction d'incrémentation du compteur.

La fonction "renvoie" toujours une valeur bien déterminée au programme appelant. Si l'instruction 'return' n'est pas utilisée, ou si elle est utilisée sans argument, la fonction renvoie un objet vide : 'None'.

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'ARGUMENTS, lesquels peuvent être des valeurs, des variables, ou même des expressions.

