GANATAN

Comment utiliser Python avec Angular 15 ?

Guide Complet

09/12/2022

Python est le langage de programmation qui a connu la **plus forte progression** dans le monde du développement ces dernières années.

Il est régulièrement dans le Top 5 des langages à suivre.

Mais qu'est ce que Python, pourquoi est-il si populaire et à quoi sert -il ?

Et surtout quel est le rapport avec Angular ?

Dans ce **guide complet** je vais vous montrer tout ce qu'il est nécessaire de savoir sur Python.

Et surtout vous allez **découvrir comment** nous pourrons l'utiliser avec le **Framework Angular** en créant un **serveur d'API**

Ce que nous allons faire ?

- **Qu'est-ce que Python ?**
  C'est un langage de programmation interprété, nous verrons ce que cela signifie.
- **Les origines de Python**
  Ecrit en1989, il a été inventé pour simplifier la vie des programmeurs.
- **Que peut-on faire avec Python ?**
  La liste des possibilités est remarquable, nous en verrons quelques unes.
- **Installation de python sur votre poste de travail**
  Comment installer Python sur Windows ou linux, et quel éditeur de code choisir ?
- **Création de notre première application**
  Comment créer une simple application en quelques minutes ?
- **Python et dépendances (librairies)**
  Comment utiliser l'outil pip pour installer et gèrer des librairies ?
- **Les fonctionnalités de Python**
  Un survol des commandes proposées par Python.
- **Faire communiquer Python et Angular**
  Comment créer un simple serveur Http avec Python et l'utiliser avec Angular ?
- **Code source**
  Le code complet du projet **angular-app-multi-api** sur github.

Qu’est-ce que Python ?

Pour ceux qui débutent faisons un petit récapitulatif.

Si vous voulez communiquer avec un être humain le plus simple est de **parler la même langue.**
Si vous voulez communiquer avec un ordinateur le plus simple est de **parler le même langage informatique.**

Python est un **langage de programmation** qui va nous permettre de communiquer avec un ordinateur.

On pourrait regrouper les langages informatiques (ou **computer language**) en 3 groupes de langages

- **Les langages de requête**
  Ils permettent de manipuler des données.
  Le plus connu d'entre eux est le **SQL**.

- **Les langages de balisage**
  Citons les très connus **HTML** et **XML**

- **Les langages de programmation**
  Et là il en existe un très grand nombre.

Parlons en donc.

Ci dessous une liste choisie de langages parmi les plus connus et les plus utilisés de nos jours.

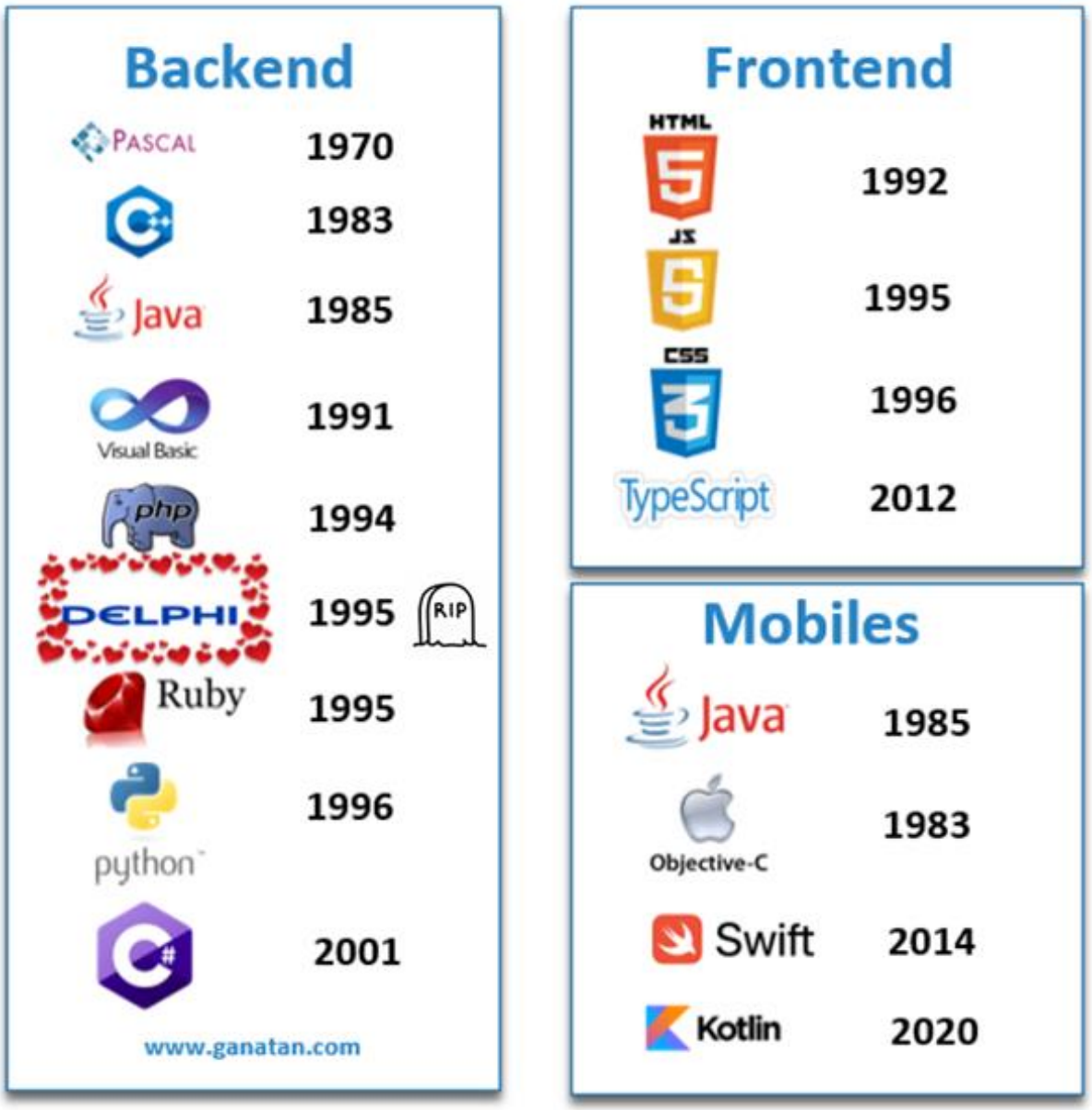
- C#
- C ++
- Delphi (**mon préféré, paix à son âme, RIP**)
- Java
- Javascript
- Kotlin
- Objective C
- Pascal
- Php
- **Python (le préféré des internautes ?)**
- Ruby
- Swift
- Visual Basic

La tendance est de dire qu'aucun langage n'est supérieur à un autre (*c'est sûr qu'entre une ferrari et une clio y'a pas de différence puisqu'elles roulent, allez pourquoi pas !*).

En tous cas tous ces langages permettent aussi bien de faire du frontend que du backend.

Néanmoins **certains sont plus adaptés que d'autres.**

Pour l'occasion je les ai classés en 3 groupes ciblés (**backend, frontend et mobiles**).



Langage interprété vs langage compilé

Il existe deux types de langages informatiques.

- Les langages **interprétés**
- Les langages **compilés**

Python est un **langage de programmation interprété** (en anglais **interpreted programming language**).

Quelques explications pour comprendre ce fameux Python.

Un ordinateur comprend uniquement un langage machine.
Un humain va utiliser un langage de programmation (java ou python par exemple).

Il va falloir traduire ce langage de programmation en langage machine.

On peut utiliser un **compilateur** ou un **interpréteur**.
Ce qui donnera

- Les langages **interprétés**
  Les lignes de code sont transcrites en langage machine au fur et à mesure.
  Les avantages sont la simplicité et la portabilité.
  Python, PHP, Basic ou Ruby sont des langages interprétés

- Les langages **compilés**
  Un logiciel spécialisé transforme tout le code du programme en langage machine en une seule fois
  L'avantage est la rapidité d'exécution.
  Java, C++ ou Pascal sont des langages compilés.

Pour faire un raccourci, Python est plus simple mais moins rapide que d'autres langages.

Tout dépend de ce que vous voulez faire et du temps dont vous disposez.

Faites vos jeux.

Choisir Python : Est-ce un bon choix ?

Quelle est la tendance actuelle sur le marché du développement ?

Où se positionne le langage **Python ?**

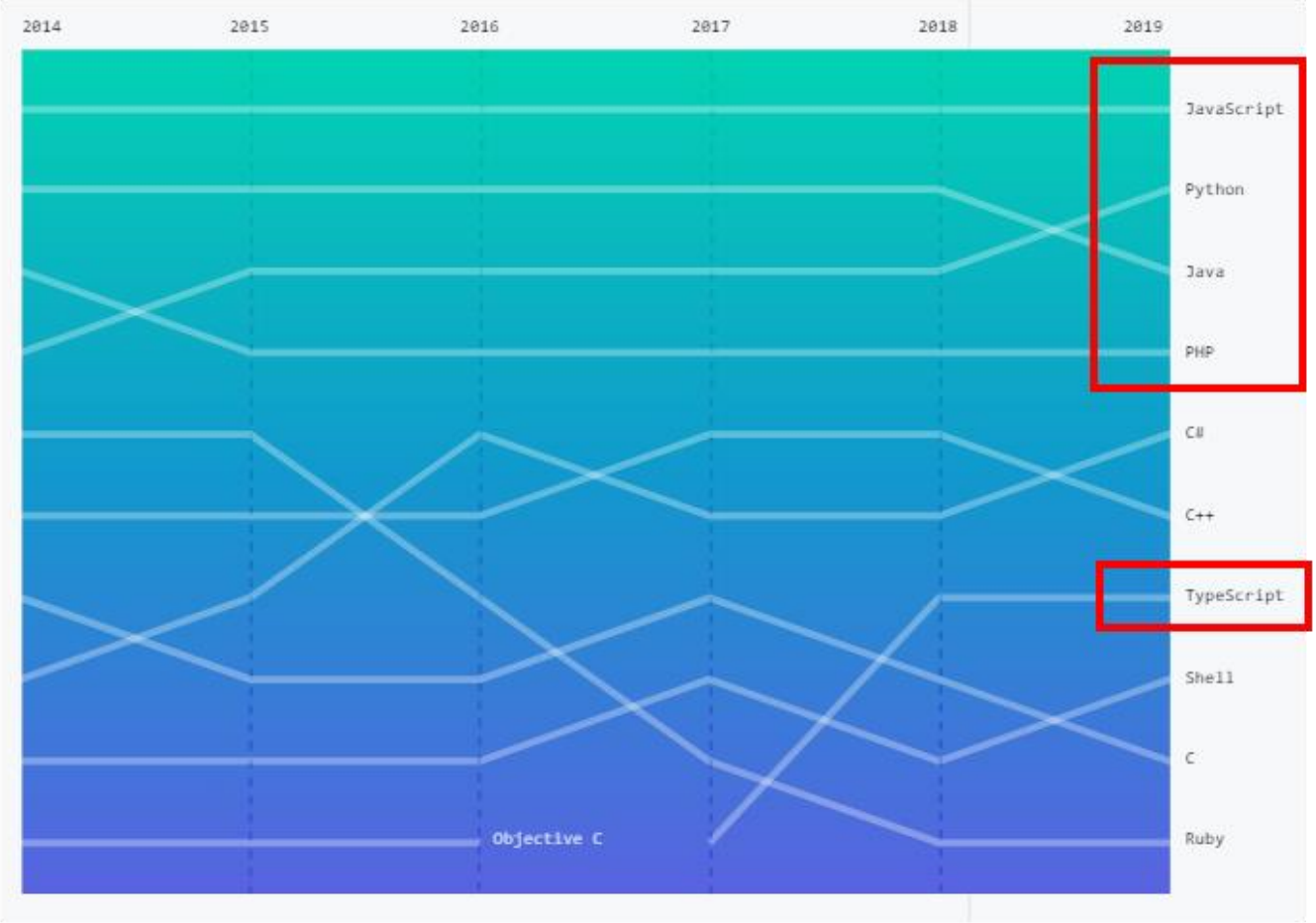
Voyons par exemple l'avis de **Github,**
LE service d'hébergement et de gestion de logiciels.

Github et son **Octoverse** nous donnent donc leur version.

Pour plus de détails c'est ici [Le top des langages informatiques]

Sinon je vous ai mis le petit schéma qui résume tout.
Et vous allez voir que Python n'est pas le dernier de leur liste.

Top Langages



Les origines

Choisir Python semble donc être une bonne idée si vous voulez trouver du travail.

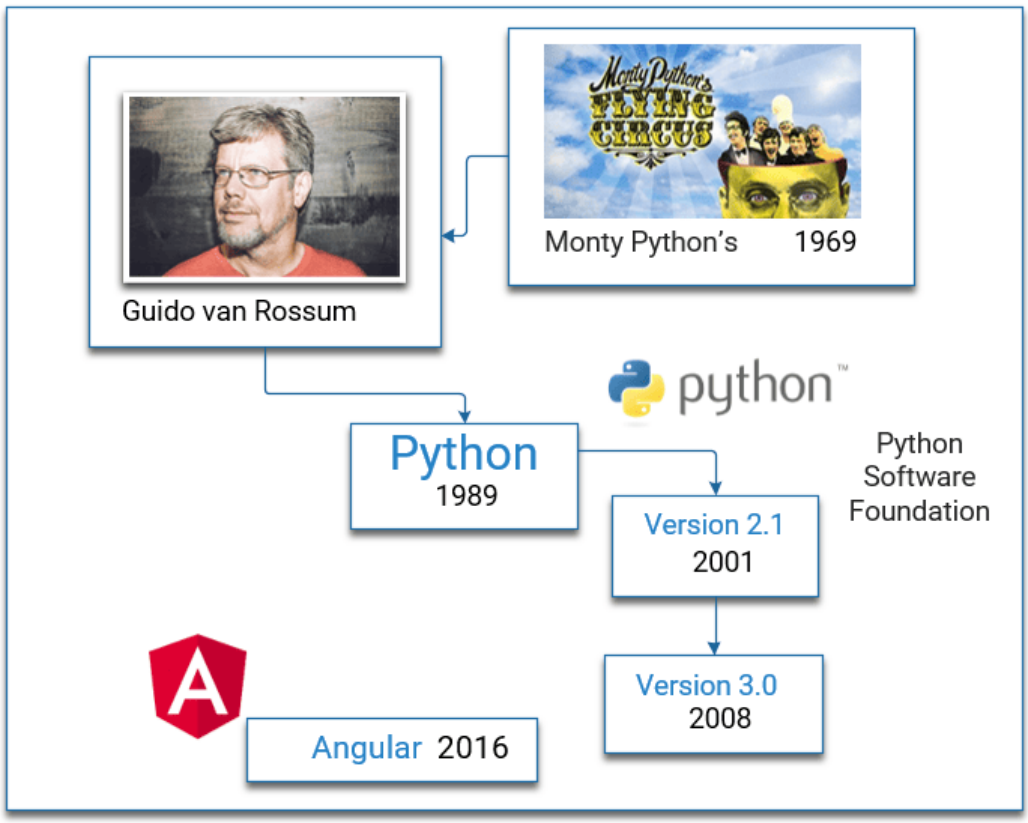
Remontons le temps et intéressons nous plus particulièrement à ses origines.

En **1989** aux Pays-Bas le programmeur **Guido van Rossum** écrit la première version du langage Python.
Il s'agissait pour lui de créer un langage de script utilisé comme **interpréteur de commandes** pour le système d'exploitation Amoeba.

Le projet est baptisé **Python** en hommage à la série télé **Monty Python's Flying Circus.**

La première version publique **0.9.0** est postée sur le forum **Usenet** en **février 1991**

Un bras coupé,une égratignure ? J'ai connu pire !



Depuis de nombreuses versions se sont succédé.

Parmi celles-ci quelques versions et quelques dates.

- 1.5 Avril 1999
- 1.6 Septembre 2000
- 2.0 Octobre 2000
- 3.0 Décembre 2008
- 3.6 Décembre 2016
- 3.7 Janvier 2018
- **3.8 Octobre 2019**

Que peut-on faire avec Python ?

Python est devenu en quelques années le langage de programmation **le plus populaire** et celui qui a connu la plus **forte progression** dans le monde du développement.

On est en droit de se poser la question: **Pourquoi** ?

A qui s'adresse t'il ?

- **Ingénieurs Logiciel**
- **Analystes Data**
- **Mathematiciens**
- **Scientifiques**
- **Comptables**
- **Ingénieurs Réseau**
- Et grâce à sa simplicité d'apprentissage on dit même les **enfants**
  Enfin faut qu'ils soient doués quand même !

  Ce langage est ainsi très largement utilisé dans le monde de l'éducation. Universitaires et professeurs en sont de fervents défenseurs.

  Et il est surtout devenu un élément moteur dans l’explosion du **Big Data** ces dernières années.

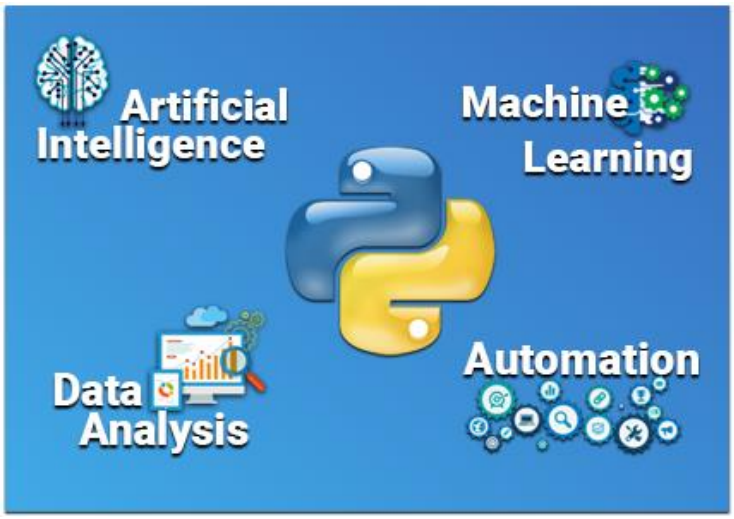
Python un langage pour tous ?

Peu de langage supporte donc la comparaison.

Quant aux domaines d'application ils sont aussi nombreux.

- **Analyse de données et Big Data**
- **Visualisation des données**
- **Intelligence artificielle**
- **Machine learning**
- **Automatisation de scripts**

Il peut faire tout ça le python



On peut construire

- **Des applications Mobile**
- **Des Applications Web**
- **Des Application Desktop**

Des avantages indéniables

- Pas besoin d'être développeur (Scientifiques et mathématiciens y trouvent leur compte)
- Résoudre des problèmes en codant beaucoup moins
- C'est un langage de **haut niveau**
- C'est un langage **Cross-platform**
  Il fonctionne aussi bien pour windows que pour Mac ou Linux
- Il dispose d'une **grosse communauté** particulièrement active qui manifeste un grand dynamisme.
- Il a un **ecosystème** très complet.
  La librairie standard forme ainsi un socle solide auquel s'ajoutent des centaines de modules.
  Pour notre bonheur Librairies et Frameworks s'avèrent ainsi extrêmement nombreux.

Et pour ne rien gâcher les débouchés sont prometteurs, ce qui vaut son pesant d'or.

- De multiples offres d'emploi.
- Des salaires parmi les plus élevés.

Python l'élu ?

Ce langage n'a donc que des qualités.

Mais trêve de plaisanteries et passons aux choses sérieuses.

Vérifions tout cela et commençons par le commencement.
Installons python sur notre poste de travail.

Installation de Python

Python est un langage interprété donc pour pouvoir **travailler il nous faut installer python sur notre poste de travail**.

Certains disent qu'il faut penser à la compatibilité avec d'anciennes versions, qu'il faut se montrer prudent dans le choix des nouvelles versions.

Pour ma part je pense que le monde du développement évolue sans cesse alors autant être le plus à jour possible.
De toute façon sur ganatan je préfère écrire des tutoriels très complets et les remettre à jour fréquemment.

Nous nous donc intéresserons uniquement à la **dernière version.**
Et au moment où j'ecris ce tuto c'est la **version 3.8.3**

Depuis 2008 deux versions coexistaient alimentant le débat entre les partisans de Python 2 et de Python 3.
Guido Van Rossum a mis un terme à ce débat puisque que le support de la version 2.7 prend fin officiellement en janvier 2020.

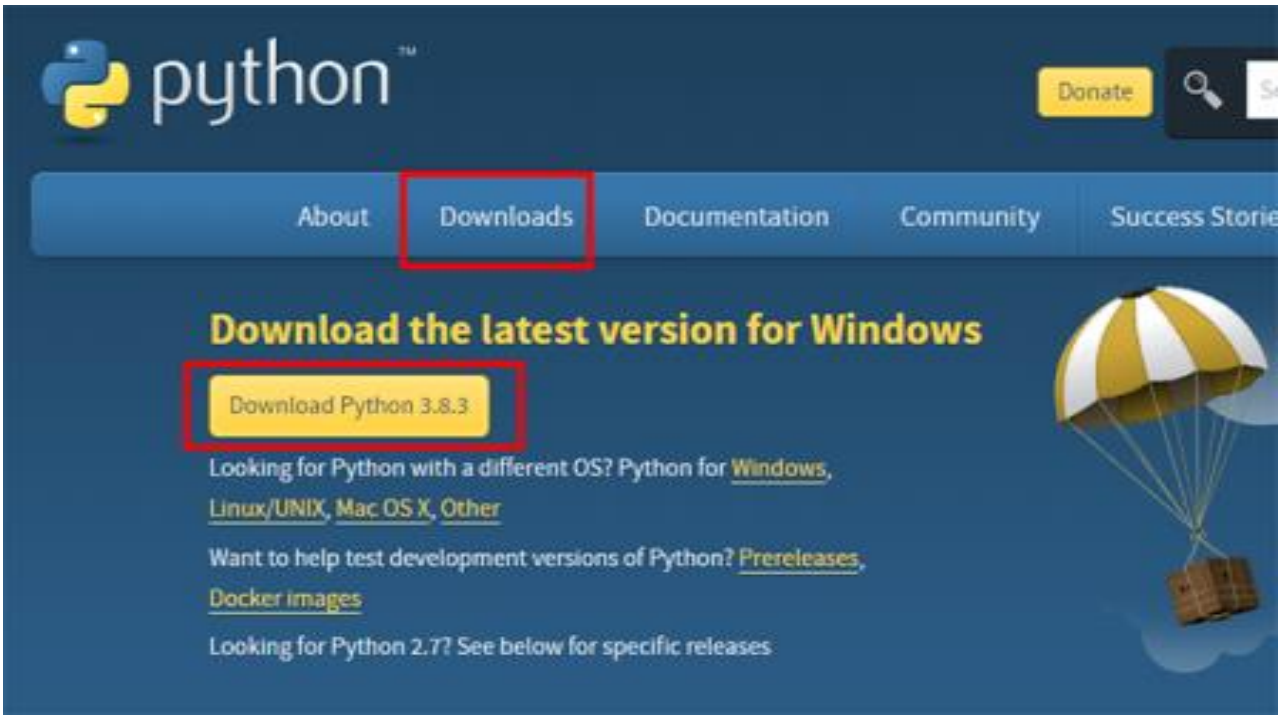
Toutes les commandes seront exclusivement réservées à cette version.
Pour les plus frileux ou les soucieux de compatibilité ce n'est peut-être pas le bon endroit !

Pour procéder à l'installation, autant aller sur le site officiel, il y a tout ce qu'il faut.

https://www.python.org/

- Downloads
- All Releases
- Download Python 3.8.3

- Choisissez la version disponible pour votre système d'exploitation.



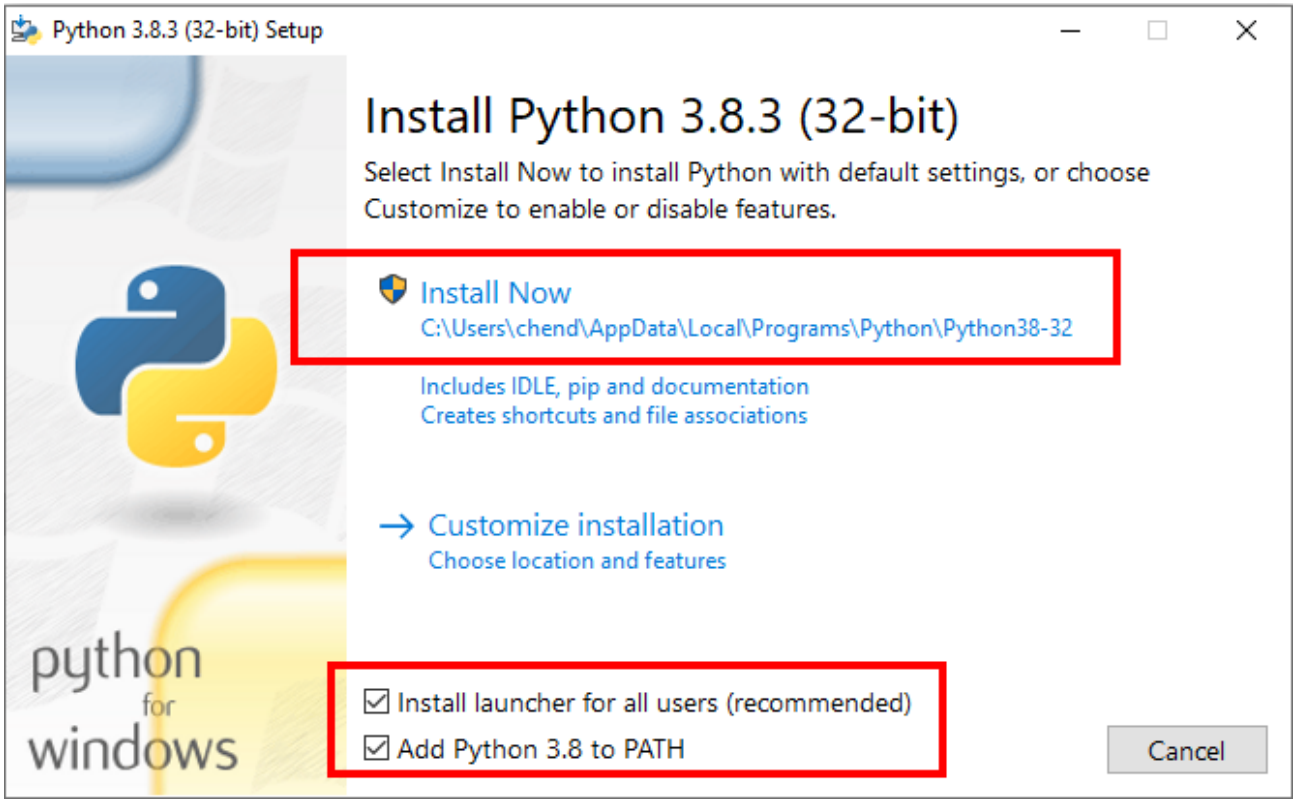
Commençons par **Windows**

- On télécharge le programme d'installation (**python-3.8.3.exe**)

- On exécute ensuite ce programme.

- Lors de l'installation sous windows n'oubliez pas de valider le path. C'est ce qui permettra de faire fonctionner l'interpréteur Python.

Ne pas oublier le PATH sous Windows



Passons à l'installation sous **linux** (ubuntu ou debian).

Python est installé par défaut sur les systèmes Linux.

Par exemple Ubuntu 18.04 propose la version 3.6 par défaut.
Ubuntu 20.04 propose la version 3.8 par défaut.

Mais au cas où vous en avez besoin les infos nécessaires sont là.

- Nous utiliserons les lignes de commandes suivantes.

- Nous utiliserons pour cela un **P**ersonal **P**ackage **A**rchives (**PPA**) ou dépôt personnel de paquets logiciels.
- Installons aussi la librairie **pip**, je vous expliquerai pourquoi un peu plus loin.
- Et si besoin était la commande permettant de désinstaller python 3

Remarque et pour info
Il ne vous sera pas possible de mettre à jour directement une installation de Python 2.X vers une installation 3.X.

```
# On en profite pour mettre à jour notre serveur
# Mise à jour de la liste des fichiers disponibles
sudo apt-get update

# Mise à jour de tous les paquets installés sur le système
sudo apt-get upgrade --yes

# Installation du package software-properties-common pour contrôler les PPA
sudo apt-get install software-properties-common --yes

# Utilisation du PPA deadsnakes
sudo add-apt-repository ppa:deadsnakes/ppa

# Mise à jour
sudo apt-get update

# Installation de Python
sudo apt-get install python3.8

# Installation de la librairie pip sous linux
sudo apt-get install python3-pip --yes

####### Commandes sous Windows ##############
# Installation du Pip
pip install pip

####### Commandes utiles ##############
# Désinstallation de python 3
sudo apt autoremove python3
```

Vérifier l’installation de Python

Il ne reste qu'à vérifier que l'installation s'est bien passée.

Utilisons pour cela une commande de prompt (cmd).

1. On vérifie que nous disposons de **Python**.
2. On vérifie que nous disposons de **pip**.
3. On éxécute la commande permettant de lancer **l'interpréteur** Python.

```
#### Commandes sous Windows ####

# 1- Test de la version de python
py --version

# 2- Test de la version de pip
pip --version

# 3- Lancer l'interpréteur en mode help
py --help

#### Commandes sous Linux/macOS ####

# 1- Test de la version
python3 --version

# 2- Test de la version de pip sous Linux
pip3 --version

# 3- Lancer l'interpréteur en mode help  sous linux
python3 --help
```

Quel éditeur de code choisir ?

Pour écrire un programme en Python il nous faut un éditeur de code.

Je vous propose deux solutions

- **Pycharm**
  L'éditeur Python le plus connu et le plus utilisé (?)

- **Visual studio code**
  Depuis plusieurs mois déjà l'incontournable pour tout développeur.

Commençons tout d'abord avec **pycharm**
L'adresse de téléchargement est la suivante

- **https://www.jetbrains.com/fr-fr/pycharm/download/#section=windows**

Il existe **deux versions** du logiciel.

- **Professionels**
  Pour le développement Python Web ou scientifique.
  Cette version prend en charge HTML, JS et SQL. Elle est payante.

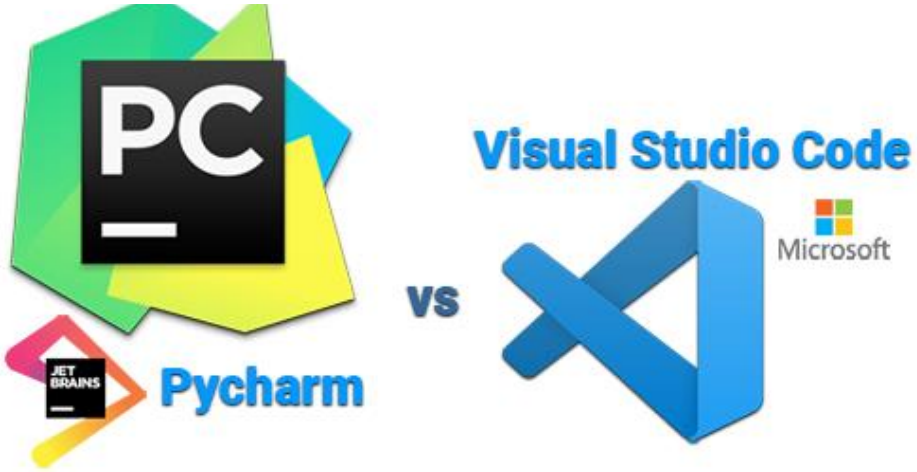
- **Community**
  Pour le développement Python pur avec une version gratuite.
  C'est celle que nous utiliserons

Pour créer une application il suffit de cliquer sur

- Create new project

Mais pour ma part je fais un autre choix.

Pycharm est payant , VS code est gratuit à vous de choisir



Cet autre choix sera **Visual studio code**.

Un éditeur de code de grande qualité qui nous permettra de travailler simultanément sur nos 2 projets.
Le **frontend** avec **Angular** et le **Backend** avec Python.

La documentation est ici
https://code.visualstudio.com/docs/python/python-tutorial

Ce qui est donc nécessaire

1. L'interpréteur **Python 3**
2. Visual Studio code (**VS Code**)
3. **L'extension Python** pour VS code

Nous avons précédemment déjà installé l'interpréteur.
Il nous reste les deux étapes suivantes.

- **VS Code**
  https://code.visualstudio.com/

- **L'extension Python**
  https://marketplace.visualstudio.com/items?itemName=ms-python.python

---

Premier programme

A ce stade là nous pourrions utiliser directement Python.

On utilise pour cela son interpréteur de commandes.

Faites le test.

```
# Lancer l'interpréteur Python
python

# Taper une ligne de commande
print ('gladiator')

# Quitter l'interpréteur
Ctrl + Z
```

Mais le mieux est d'écrire des fichiers contenant notre code.

Puisque l'on a tout ce qu'il faut, alors écrivons un programme simple.

Par convention tous les fichiers écrits en Python se termine par **l'extension .py**

On crée notre premier fichier **starter.py**

Quant au code tout aussi simple et facile à comprendre.

starter.py

```
# Double quote possible
print("Python Application Title")
print("File Starter.py")

# Simple quote possible
print('First Line Example')
print('Second Line Example')
```

Sur **Pycharm** on éxécute la commande **Run** (ou touche de raccourci **Alt + Maj + F10**)
On obtient le résultat dans la console

Sur **Visual studio code** on éxécute la commande **Run** (icône verte ou touche de raccourci **Ctrl + F5**)

On peut aussi éxécuter la commande (en ligne de commande)

- ***python3 starter.py*** *(linux)*
- ***py starter.py*** *(windows)*

Comment fonctionne le langage Python ?

Comme beaucoup d'autres langages de programmation Python permet d'utiliser de nombreuses **dépendances** ou **bibliothèques** disponibles.

C'est bien pratique car on ne va pas réinventer la roue à chaque fois.

Evidemment nous serons confrontés à ces questions

- **Trouver** les dépendances.
- **Installer** les dépendances
- **Gèrer** les différentes **versions**

Il nous faut donc un moyen de gèrer ces bibiliothèques ou dépendances.

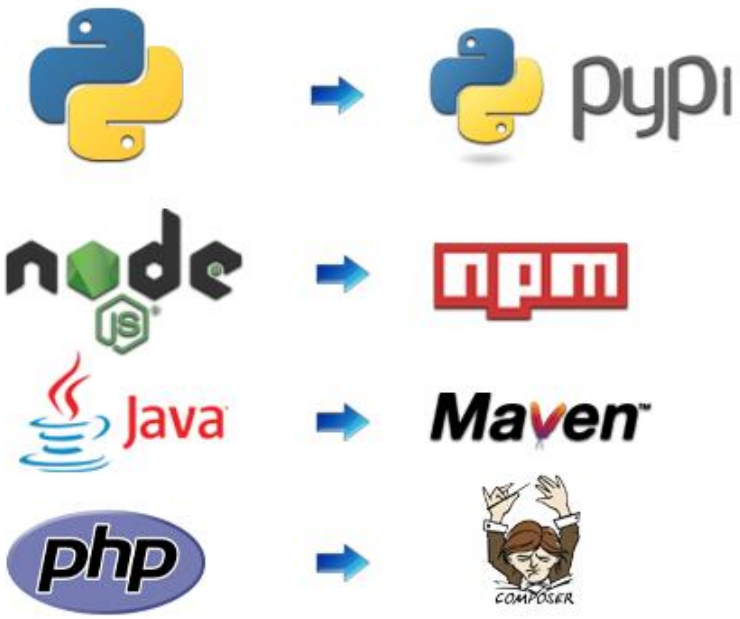
La solution idéale se nomme **pip** pour The **P**ython **P**ackage **I**ndex (**PyPI**).

pip est un système de gestion de paquets.
Il va nous permettre d'installer et de gèrer des librairies écrites en Python.

Vous retrouvez le même type d'outils avec

- **npm** (node package manager) sous **Node.js**
- **Maven** sous **Java**
- **Composer** sous **Php**

pip, maven, npm ou composer même combat



Les adresses utiles liées à pip sont

- Le dépôt **open source** sur github
  https://github.com/pypa/pip
  Utile si vous voulez contribuer à la communauté Python.

- Le site permettant de rechercher les **librairies python disponibles**
  https://pypi.org/

L'installation de pip est celle d'une simple librairie python.

```
# Installation du Pip
pip install pip

# Installation de la librairie pip sous linux
sudo apt-get install python3-pip

# Test de la version de pip sous windows
pip --version
```

Utiliser les librairies avec Python ?

Nous allons appliquer immédiatement cette notion de librairie et d'utilisation de pip.

Notre environnement fonctionne, nous disposons

1. d'un interpréteur **python3**
2. d'un éditeur de code **VS code**
3. du gestionnaire de paquets **pip**

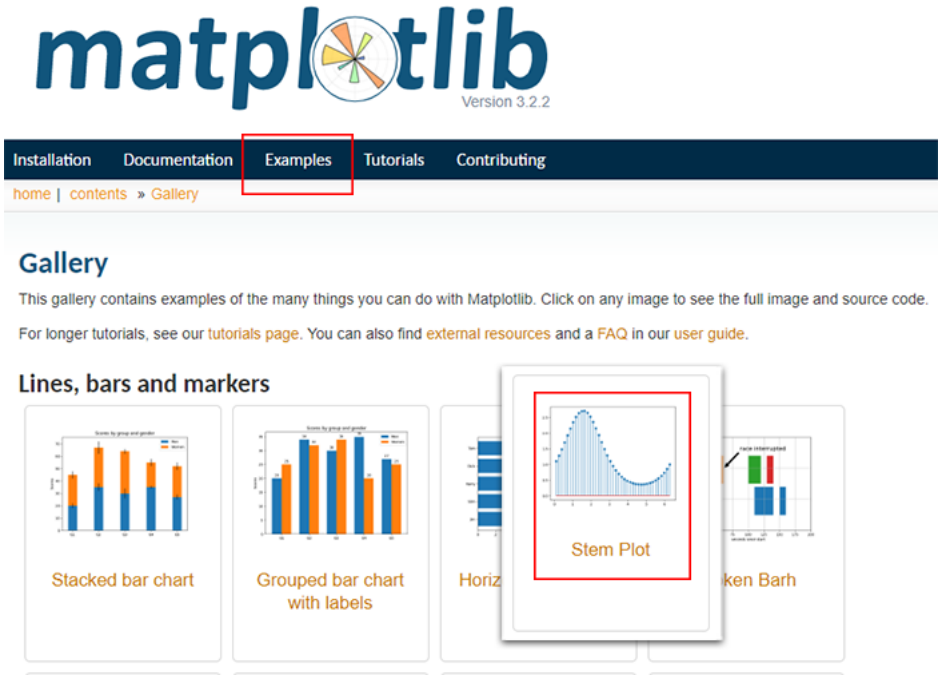
Avant d'aller plus loin voyons le problème des **dépendances** et l'**utilisation de pip**.
Imaginons que nous voulions utiliser une librairie.

L'une des plus utilisées concerne les graphiques.

Le site de cette librairie est le suivant **https://matplotlib.org/**

Nous allons utiliser un exemple simple, je vous conseille dans la liste proposée de sélectionner **Stem Plot**
https://matplotlib.org/gallery/index.html

En cliquant sur l'exemple nous pourrons récupèrer le code Python correspondant.
Code que l'on copie dans un fichier **stem-plot.py**



stem-plot.py

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

x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))

plt.stem(x, y, use_line_collection=True)
plt.show()
```

Si on essaie d'exécuter le fichier,
Vous obtiendrez une erreur puisque le module **matplotlib.pyplot** n'est pas reconnu.

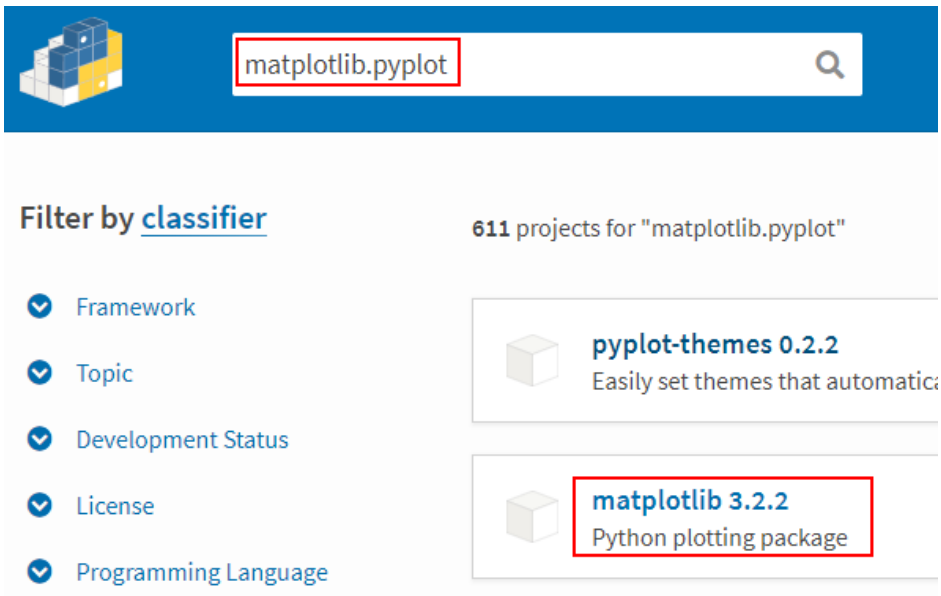
```
# Exécution du fichier sous linux
python3 stem-plot.py

# Exécution du fichier sous windows
py stem-plot.py

# Message d'erreur obtenu
# ModuleNotFoundError: No module named 'matplotlib'
```

Nous avons donc besoin d'une librairie ou module **matplotlib.pyplot**

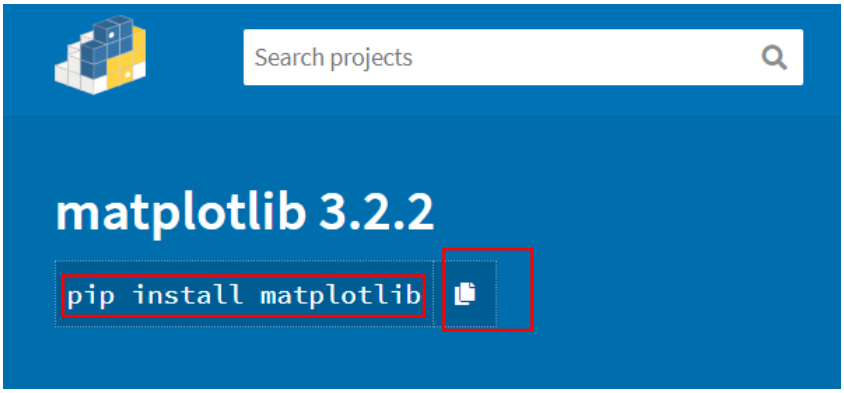
On résoud le problème en 3 étapes

1. Vérifier que la librairie existe sur https://pypi.org/
2. Récupèrer le code d'installation
3. Installer la librairie et exécuter le fichier
4. Vérifier le résultat

1/ Rechercher la librairie



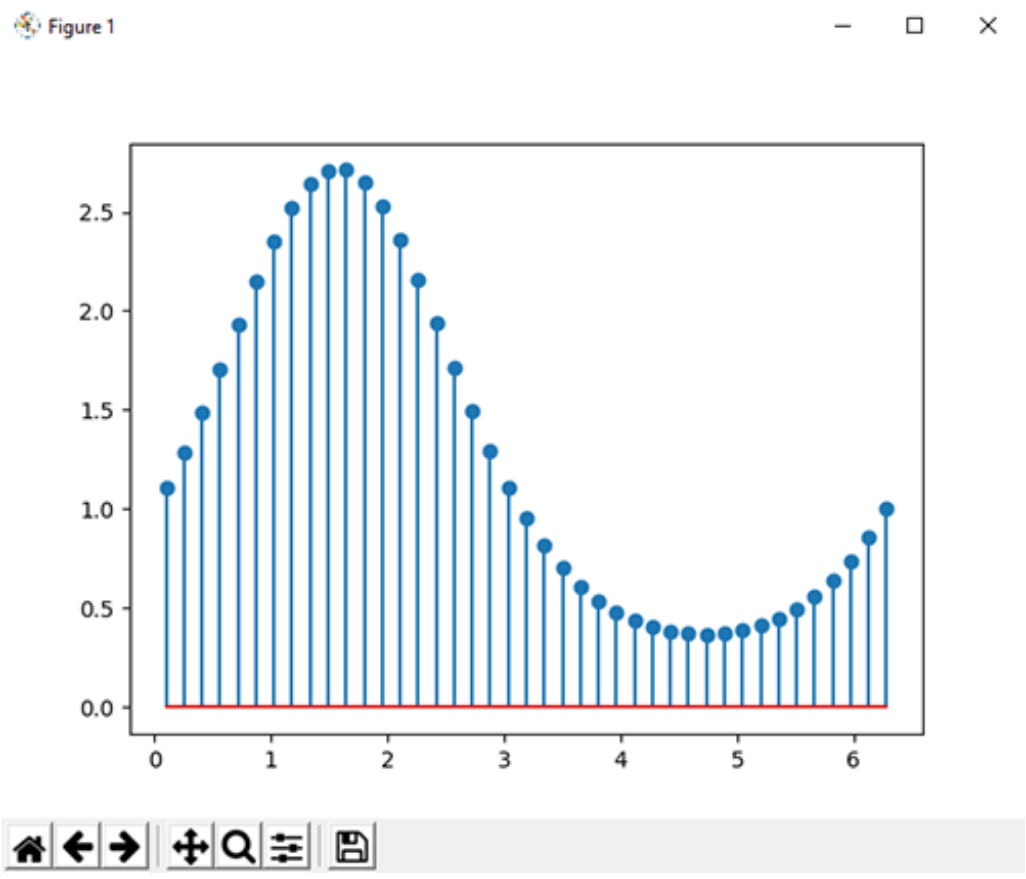
2/ Récupérer le code



3/ Installer la librairie

```
# Installation sous Windows
python -m pip install matplotlib
pip install matplotlib

# Installation sous Linux (Ubunt/Debian)
apt-get install python3-tk
python3 -m pip install matplotlib

# Exécution du fichier sous linux
python3 stem-plot.py

# Exécution du fichier sous windows
py stem-plot.py

############ Commande utile #####
# Désinstallation avec la commande
pip uninstall matplotlib
```

4/ Vérifier le résultat



Bonnes pratiques

Les bonnes pratiques ou Best practices sont essentielles.

Elle vous permettront de relire votre code des années après l'avoir écrit.
Elles permettront aussi à d'autres développeurs de le faire facilement.

Dans le cas de python on dispose d'une documentation officielle.
Elle est regroupée sous l'acronyme **PEP**

- Python Enhancement Proposals (**PEPs**)
  https://www.python.org/dev/peps/pep-0008/

Je vous fais pour l'occasion un résumé de l'essentiel de la PEP 8.

Alors voici quelques recommandations.

Indentation

- L'indentation est obligatoire en Python pour séparer les blocs d'instructions.
- La syntaxe de chaque niveau d'indentation est simple : **4 espaces**.

Règles de nommage

- Nom des **variables**, **fonctions** et **modules**
  my_variable
  my_function_01()
  my_function_02()
  my_module
- Nom de **constantes**
  MY_CONSTANT
- Nom de **classes**
  MyClasse

Exemples de code

Nous pouvons maintenant tester **notre première application**.

il nous reste à expérimenter ce sympathique langage.

La documentation complète sur Python

- **https://docs.python.org/3.8/**

Le but de ce turoriel n'est pas d'étudier en détail le langage Python mais juste d'avoir un aperçu.

C'est surtout le lien entre Angular et Python qui nous intéressera au final.

Je vous livre quelques essentiels à travers chaque fois un fichier exemple.

Comme vous l'avez peut-être deviné j'aime beaucoup le cinéma.
Alors autant se faire plaisir en utilisant des exemples choisis de films.
Et comme l'anglais est la langue officielle des développeurs on s'en tiendra à la langue de Shakespeare.

Utilisez chaque fichier et faites des tests c'est le meilleur moyen de comprendre un langage.

Donc une petite trousse de survie avec une liste choisie

- **Commentaires**
  Comment écrire des commentaires sur une ou plusieurs lignes de code.

- **Fonctions**
  Créer une fonction et l'utiliser.

- **Variables**
  Des variables à utiliser dans des fonctions, utiles pour formatter du texte
  .

- **if ... else**
  Comment utiliser des conditions simples.

comments.py

```
print ('Example source code')

print ('a Line Comment')

# Braveheart
# Gladiator

print ('Multi Line Comment')

'''
Mel Gibson
Ridley Scott
```

functions.py

```
def movieName(id, name):
    print(id)
    print('Movie Name : ' )
    if name == 'Alien':
        print('Movie Title:Alien')
    else:
        print('Movie Title:else')
        if name == 'Gladiator':
            print('Movie Title:Gladiator')

movieName(1, 'Alien')
movieName(2, 'Gladiator')
```

variables.py

```
def showVariables(id, name):

    info = "The Movie identifier is {}"
    print(info.format(id))

    info = "The Movie name is {}"
    print(info.format(name))

    domestic = 187705427
    international = 272878533
    info = "The total is {:.2f} dollars"
    print(info.format(domestic + international))

    name = "Gladiator"
    director = "Ridley Scott"
    print(name + ' - ' + director)

showVariables(1, 'Gladiator')
```

if-else.py

```
def selectMovie(id, name):
    info = "- The Movie identifier is {}"
    print(info.format(id))

    if name == 'Alien':
        print('Movie Selected : Alien')
    else:
        print('Movie Selected : Not Alien')
        if name == 'Apocalypto':
            print('Movie Selected : Apocalypto')
        else:
            print('Movie Selected : Not Apocalypto')
            print('Movie Selected : ' + name)

selectMovie(1, 'Alien')
selectMovie(2, 'Apocalypto')
selectMovie(3, 'Avatar')
```

Python frontend ou Backend ?

Grande question.

Quel rapport entre python et Angular ?
Aucun si ce n'est que ce sont deux langages qui permettent de créer des sites web.

Honnêtement faire des sites avec Python pourquoi pas. Mais Angular, React ou vue c'est quand même mieux.

Donc ce que nous allons faire

- une **application frontend** avec **Angular** pour la partie graphique.
- une **application backend** avec **Python** pour la gestion des données.

Commençons avec le Backend.

Nous procéderons par étapes du plus simple au plus compliqué.

1. un serveur HTTP avec **SimpleHTTPRequestHandler** .
2. un serveur HTTP avec **BaseHTTPRequestHandler**
3. un serveur HTTP qui génère des **pages HTML**
4. un serveur HTTP qui utilise des **fichiers HTML** et du **routing**
5. un serveur HTTP avec **json**
6. un serveur HTTP avec **multi json**

Serveur HTTP simple

La première étape va nous amener à créer notre backend avec Python

Il s'agira d'un serveur HttpServer simple.

La documentation suivante nous sera utile
https://docs.python.org/3/library/http.server.html

Nous n'utiliserons aucun serveur Http comme Apache ou nginx.
Mais tout simplement nous utiliserons les fonctionnalités natives du langage python.

On va utiliser un module

- **http.server**

Nous commencerons par un serveur statique.

1. Créons 1 fichier **http-server-simple.py**
   Il contient le code Python qui simule notre serveur http

2. Exécuter le fichier avec l'intrepréteur Python

3. Tester avec notre navigateur

http-server-simple.py

```
import http.server

class Server(http.server.SimpleHTTPRequestHandler):

    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b'First page')

httpd = http.server.HTTPServer(('localhost', 80), Server)
print ("Server listening on port 80")
httpd.serve_forever()
```

Exécuter et tester

```
# Exécuter le code sous Windows
py http-server-simple.py

# Exécuter le code sous Linux (sudo permet de passer en mode admin)
sudo python3 http-server-simple.py

# Tester le progamme par défaut le port 80
http://localhost/
```

Serveur HTTP avec BaseHTTPRequestHandler

La librairie **SimpleHTTPRequestHandler** permet de gèrer le verbe GET mais ne permet pas de gèrer le verbe POST.
Nous utiliserons la librairie **BaseHTTPRequestHandler**.

Dans la suite du tutoriel je vous expliquerai plus en détail.

Et remanions un peu le code.

Le principe sera toujours le même

- Ecrire le code
- Exécuter le fichier
- Tester

http-server-simple-base.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler

class Server(BaseHTTPRequestHandler):

    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b'First page')

port = 80
httpd = HTTPServer(('localhost', port), Server)
print(f"Server listening on port {port}")

httpd.serve_forever()
```

Exécuter et tester

```
# Exécuter le code sous Windows
py http-server-simple-base.py

# Exécuter le code sous Linux (sudo pemert de passer en mode admin)
sudo python3 http-server-simple-base.py

# Tester le progamme par défaut le port 80
http://localhost/
```

Serveur HTTP qui génère des pages HTML

Ci-dessous une version qui simule deux pages html via le code.

http-server-html.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler


class Server(BaseHTTPRequestHandler):

    def do_GET(self):
        if self.path == '/':
            self.send_response(200, 'OK')
            self.send_header('Content-type', 'html')
            self.end_headers()
            self.wfile.write(bytes(
                "<html> <head><title> Http Server </title> </head> <body>Main Page</body>", 'UTF-8'))
        else:
            self.send_response(200, 'OK')
            self.send_header('Content-type', 'html')
            self.end_headers()
            self.wfile.write(bytes(
                "<html> <head><title> Http Server </title> </head> <body>Unknown page</body>",
'UTF-8'))


port = 80
httpd = HTTPServer(('localhost', port), Server)
print(f"Server listening on port {port}")

httpd.serve_forever()
```

Commandes de test du serveur

```
# Exécuter le programme Python
py http-server-html.py

# Tester le résultat dans votre navigateur
http://localhost

# Tester une page inconnue
http://localhost/bad_page
```

Un serveur HTTP avec un peu de routing

Je vous livre une version un peu plus étoffée avec l'utilisation de fichiers html et la gestion du routing.

Créons 3 fichiers

- **http-server-routing.py**
  Il contient le code Python qui simule notre serveur http

- **index.html**
  Contient la page principale

- **movies.html**
  Contient les données à afficher sous forme de liste.

http-server-routing.py

```
import http.server

class Serv(http.server.BaseHTTPRequestHandler):

    def do_GET(self):
        if self.path == '/':
            self.path = '/index.html'
        if self.path == '/movies':
            self.path = '/movies.html'
        try:
            file_to_open = open(self.path[1:]).read()
            self.send_response(200)
        except:
            file_to_open = "File not found"
            self.send_response(404)
        self.end_headers()
        self.wfile.write(bytes(file_to_open, 'utf-8'))

port = 80
httpd = http.server.HTTPServer(('localhost', port), Serv)
print(f"Server listening on port {port}")

httpd.serve_forever()
```

index.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, user-
scalable=no">
  <title>Page Title</title>
</head>

<body>
  Example HttpServer with Python
  <a href="movies">Movies</a>
</body>

</html>
```

movies.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, user-
scalable=no">
  <title>Page About</title>
</head>

<body>
  <h1>Example HttpServer with Python</h1>
  <h2>About Page</h2>
  <ul>
    <li>Gladiator</li>
    <li>Alien</li>
    <li>Blade Runner</li>
    <li>Legend</li>
  </ul>
</body>

</html>
```

Commandes de test de notre serveur

```
# Exécuter le programme Python
py http-server-routing.py

# Test d'affichage de la page index.html
http://localhost

# Test d'affichage de la page movies.html
http://localhost/movies

# Test d'affichage d'un url inconnue
http://localhost/url_error
```

Serveur HTTP et API Json

Vous avez peut-être déjà entendu parler des termes suivants

- **API**
- **Resful**
- **CRUD**

Un Backend va nous permettre de créer des API.

Là aussi on ne va pas rentrer dans les détails cela demanderait un tutoriel complet.
Mais on veut juste voir que cela marche.

De toute façon il serait plus judicieux d'utiliser
un **Framework** comme **Django** ou **Flask**.

Mais essayons de voir ça.

Remarque :

- *Angular utilise par défaut le port 4200 pour ses applications.*
- *Nous utiliserons dans notre backend le port 5201 (c'est arbitraire vous pouvez utiliser celui que vous voulez)*
  Le code source contient plusieurs types d'API.
  J'ai utilisé un port spécifique pour chaque API, on pourra ainsi les utiliser distinctement dans notre Frontend.

Dans les exemples qui suivent

- **http-server-json.py**
  Pour tester l'affichage de données json

- **http-server-api.py**
  Pour simuler une API json

http-server-json.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
import json

class Server(BaseHTTPRequestHandler):

    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(json.dumps(
            [
                {'name': 'Gladiator'},
                {'name': 'Alien'},
                {'name': 'Gladiator'},
                {'name': 'Avatar'}
            ]
        ).encode())

port = 5201
httpd = HTTPServer(('localhost', port), Server)
print(f"Server listening on port {port}")

httpd.serve_forever()
```

http-server-api.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
import json

class Server(BaseHTTPRequestHandler):

    def do_GET(self):
        items = [{'name': 'movies'},
                 {'name': 'actors'}]
        if self.path == '/movies':
            items = [{'name': 'Gladiator'},
                     {'name': 'Alien'},
                     {'name': 'Gladiator'},
                     {'name': 'Avatar'}]
        if self.path == '/actors':
            items = [{'name': 'Mel Gibson'},
                     {'name': 'Russel Crowe'},
                     {'name': 'Joaquim Phoenix'},
                     {'name': 'Sam Wortington'}]
        try:
            self.send_response(200)
            self.end_headers()
            self.wfile.write(json.dumps(items).encode())
        except:
            self.send_response(404)
            self.end_headers()
            self.wfile.write(json.dumps(items).encode())

port = 5201
httpd = HTTPServer(('localhost', port), Server)
print(f"Server listening on port {port}")

httpd.serve_forever()
```

```
# Exécuter L'affichage de données json
py http-server-json.py

# Tester L'affichage de données json
http://localhost:5200/


# Exécuter L'API
py http-server-api.py

# Tester L'affichage de l'API
http://localhost:5201/

http://localhost:5201/movies

http://localhost:5201/url_unknow
```

Serveur HTTP version finale

Bien évidemment construire une API avec Python n'est pas aussi simple que ça.

La notion de CORS(*Cross-origin* resource sharing) ou partage des ressources entre origines multiples doit être traitée.
Les verbes GET,POST, DELETE pour une API CRUD(Create Read Update Delete) sont à prendre en compte.

Tout ça mériterait un tutoriel plus complet et plus détaillé.
N'hésitez pas à en parler dans vos commentaires.

Je vous livre enfin une version finale qui pourra fonctionner avec notre Frontend Angular.

http-server.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
import json


class Server(BaseHTTPRequestHandler):

    def do_GET(self):
        items = [{'name': 'movies'},
                 {'name': 'actors'}]
        if self.path == '/movies':
            items = [{'id' : 1001,'name': 'Python Gladiator'},
                     {'id' : 1002,'name': 'Python Alien'},
                     {'id' : 1003,'name': 'Python Gladiator'},
                     {'id' : 1004,'name': 'Python Avatar'}]
        if self.path == '/actors':
            items = [{'id' : 2001,'name': 'Python Mel Gibson'},
                     {'id' : 2002,'name': 'Python Russel Crowe'},
                     {'id' : 2003,'name': 'Python Joaquim Phoenix'},
                     {'id' : 2004,'name': 'Python Sam Wortington'}]
        try:
            self.send_response(200)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            self.wfile.write(json.dumps(items).encode())
        except:
            self.send_response(404)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            self.wfile.write(json.dumps(items).encode())


port = 5201
httpd = HTTPServer(('localhost', port), Server)
print(f"Server listening on port {port}")

httpd.serve_forever()
```

```
# Exécuter L'API
py http-server.py

# Tester L'affichage de l'API
http://localhost:5201/

http://localhost:5201/movies

http://localhost:5201/actors

http://localhost:5201/url_unknow
```

Création du Frontend

Maintenant créons notre application Frontend avec Angular.

Le site **ganatan** est dédié au **Framework Angular.**
Pour plus de détails je vous conseille de lire ces 4 tutoriaux.

1. **https://www.ganatan.com/tutorials/demarrer-avec-angular**
2. **https://www.ganatan.com/tutorials/routing-avec-angular**
3. **https://www.ganatan.com/tutorials/lazy-loading-avec-angular**
4. **https://www.ganatan.com/tutorials/bootstrap-avec-angular**

Mais pour aller plus vite voici les étapes pour créer cette application.

- Installer **Angular CLI**
- Créer une **application from scratch**
- Modifier le fichier **app.component.ts**
- Exécuter l'API Backend
- Exécuter le frontend

```
# Installation d'angular-cli
npm install -g @angular/cli

# Générer un projet appelé angular-starter avec options par défaut
ng new angular-starter --defaults
```

app.component.ts

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  title = 'angular-starter';
  items: any;

  constructor(private http: HttpClient) {
    this.getItems();
  }

  onGet() {
    this.getItems();
  }

  getItems() {
    this.http.get('http://localhost:5201/movies')
      .subscribe(
        data => {
          this.items = data;
        }
      );
  }

}
```

```
# Lancer le frontend Angular
npm run start
```

Naturellement on n'obtient pas de données.

Car l'API Backend n'a pas été lancée.

Il va nous falloir exécuter l'API et retester notre frontend.

```
# Aller dans le répertoire contenant l'API
cd api
cd python
cd http-server

# Exécuter l'API
py http-server.py

# Vérifiez le fonctionnement de l'API backend en lançant dans votre navigateur la commande
http://localhost:5201

http://localhost:5201/movies

# Vérifiez le fonctionnement du frontend en lançant dans votre navigateur la commande
http://localhost:4200/
```

Pour aller plus loin

Python est bien pratique pour créer rapidement des API.

Mais évidemment les librairies sont là pour nous simplifier le travail.
Et plus encore les Frameworks.

Un framework est une sorte de boite à outils encore plus pratique.

Les deux **Frameworks Python** les plus utilisés sont **Django** et **Flask**

Je vous propose deux tutoriels pour créer des API

1. https://www.ganatan.com/tutorials/django-avec-angular
2. https://www.ganatan.com/tutorials/flask-avec-angular