Python

Python - Sommaire

Python - Présentation

Python est un langage de programmation inventé par Guido van Rossum. La première version de python est sortie en 1991.

Python est un langage de programmation interprété, c'est à dire qu'il n'est pas nécessaire de le compiler avant de l'exécuter.

Python - Présentation

Avantage:

- Interprété
- Orienté objet
- Haut niveau
- Libre
- Mehdi Approuve
- Best Language 2017
- Syntaxe simple

Python - Présentation

Inconvénients:

- Lent
- Absence de pointeurs
- Typage YOLO
- Python 2 & Python 3

Python - Présentation

A quoi cela sert il:

- Apprendre à développer
- Faire des programmes graphiques
- Faire du Web
- Big Data
- Sécurité (création de virus et autres payload)
- etc...

Python - Présentation

Des projets sous python:

- **Zope**, un serveur d'application innovant, et CPS, un framework de gestion de contenu et de travail collaboratif basé sur Zope.
- Les programmes d'administration système spécifiques à la distribution Red Hat Linux.
- Des moteurs de recherche comme Google ou Yahoo!.
- Youtube
- La Nasa
- En sécurité (kali tools…)

Python - Installation

Pour mac et Linux python est déjà installé.

Pour windows : https://www.python.org/

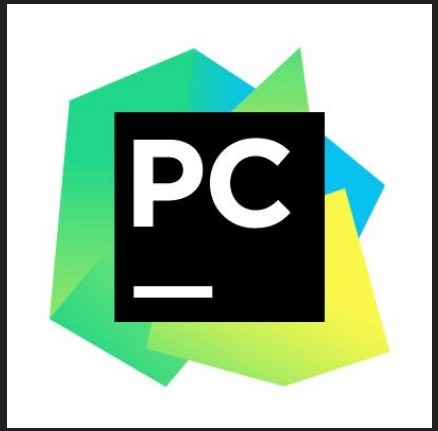
note: ne pas oublier de cocher Add python.exe to Path

Dans un terminal: python --version

Python - Installation

Installé aussi:

- un bon IDE : PyCharm
- un terminal (cmder ou terminator pour linux)



Python - Les bases - Algorithmes

Un algorithme est une suite d'instruction détaillées qui, si elles sont correctement exécutées, conduit à un résultat donné.

exemple:

1. Choisir un nombre entier
2. Le multiplier par lui-même
3. Énoncer le résultat obtenu

Python - Les bases - Complexité

L'analyse de la complexité d'un algorithme consiste en l'étude formelle de la quantité de ressources (par exemple de temps ou d'espace) nécessaire à l'exécution de cet algorithme. Celle-ci ne doit pas être confondue avec la théorie de la complexité, qui elle étudie la difficulté intrinsèque des problèmes, et ne se focalise pas sur un algorithme en particulier.

Python - Les bases - Pseudo-Code

En algorithmique, nous utiliserons un langage situé à mi-chemin entre le langage courant et un langage de programmation appelé pseudo-code. Il n'y a pas de norme concernant ce pseudo-code qui peut varier légèrement d'un enseignant à l'autre. Le but est surtout de mettre l'accent sur la logique de l'algorithme. L'avantage du pseudo-code est qu'il permet de rester proche d'un langage informatique sans qu'il soit nécessaire de connaître toutes les règles et spécificités d'un langage particulier.

Python - Les bases - les variables

Une variable est une sorte de boîte virtuelle dans laquelle on peut mettre une (ou plusieurs) donnée(s). L'idée est de stocker temporairement une donnée pour travailler avec. Pour votre machine une variable est une adresse qui indique l'emplacement de la mémoire vive où sont stockées les informations que nous avons liées avec.

Python - Les bases - les types

Dans la plupart des langages de programmation il est possible d'attribuer arbitrairement aux variables des types pour – entre autres – déterminer la nature des données qu'elle peut contenir et la manière dont elle sont enregistrées et traitées par le système. Concrètement le type d'un élément influe sur la taille que le compilateur ou l'interpréteur lui allouera en mémoire ; et les opérations qui ne devraient pas être permises si la syntaxe du langage était respectée.

Python - Les bases - les lignes d'instructions

Une instruction est constituée d'une combinaison d'expressions (identificateurs, variables, opérateurs, expressions littérales, appels de fonction etc…). Elle peut être suivie ou entrecoupée de commentaires.

Une instruction s'achève lorsqu'une autre instruction commence. Un saut de ligne permet de les séparer en python, mais cela peut être différent dans d'autre langage.

Python - Les bases - les variables

```
# Dans votre fichier script.py

#!/usr/bin/python3

a = 10

b = 5

print(a + b)

# Ceci est un commentaire
sur une ligne
```

```
# Dans un terminal
$ python script.py # Pour lancer le
script
```

Python - Les bases - les variables

```
age = "je suis vieux"

print(age)

age = "je suis jeune"

print(age)
```

```
text = "je suis du texte"

print(type(text))

nombre = 42

print(type(nombre))
```

Python - Les bases - les variables

```
text = "je suis du texte"

print(text * 3)

#Concaténation

text = text + " en plus"

print(text)
```

```
txt = "Hello World !!!!"

print(len(txt))

print(txt[1])

print(txt[0:5])

print("bonjour je m'appelle %s" %
("Mehdi"))
```

Python - Les bases - les variables

```
text = "je suis du \"texte\""

print(text)
```

A ne pas utiliser !!!

print in and or if del for is raise assert elif from lambda return break else global not try class except while continue exec import pass yield def finally

Python - Les bases - les listes

```python
maListe = []

print(type(maListe))

maListe = [1, 2, 3]

print(maListe)
```

```python
maListe = []

maListe.append(1)

print(maListe)

maListe.append("salut")

print(maListe)
```

Python - Les bases - les listes

```
maListe = ["1er", "deuxieme", "troisieme"]

print(maListe[1])

print(maListe[2])

maListe[1] = "changement"

print(maListe)
```

Python - Les bases - les listes

```
maListe = ["1er", "deuxieme", "troisieme"]

#Supprimé avec l'index

del maListe[1]

print(maListe)

#Supprimé avec la valeur

maListe.remove("troisieme")
```

Python - Les bases - les listes

```
maListe = ["1er", "deuxieme", "troisieme"]

#Inverser les valeurs d'une liste

maListe.reverse()

print(maListe)

#Compter le nombre d'item d'une liste

print(len(maListe))
```

Python - Les bases - les listes

```
maListe = ["1er", "deuxieme", "troisieme"]

#Compter le nombre d'occurences d'une valeur

print(maListe.count("1er"))

#Trouver l'index d'une valeur

print(maListe.index("1er"))
```

Python - Les bases - les listes

```
maListe = ["1er", "deuxieme", "troisieme"]

print(maListe[-1]) # Cherche la dernière occurrence

print(maListe[-2:]) # Affiche les 2 dernières occurrences

print(maListe[:]) # Affiche toutes les occurrences

maListe[:] = [] # Vide la liste

print(maListe)
```

Python - Les bases - les listes

```
maListe = ["1er", "deuxieme", "troisieme"]
for item in maListe:
    print (item)

for item in enumerate(maListe): # Avec l'index
    print(item)
```

Python - Les bases - les listes

```
# Bad

maListe = ["1er",
"deuxieme", "troisieme"]

secondeList = maListe

maListe[0] = "toto"

print(secondeList)
```

Python - Les bases - les listes

```
# Good

maListe = ["1er",
"deuxieme", "troisieme"]

secondeList = maListe[:]

maListe[0] = "toto"

print(secondeList)
```

Python - Les bases - les listes

```
maListe = ["1er",
"deuxieme", "troisieme"]

print("1er" in maListe)

print ("toto" in maListe)
```

```
maListe = list(range(15))

print(maListe)

liste = list(range(10))

liste.extend(maListe)

print(liste)
```

Python - Les bases - les tuples

```
# Un tuple est une liste qui ne peut plus être modifiée.

monTuple = ()

print(type(monTuple))

monTuple = ("toto",)

print(type(monTuple))

monTuple[0] = "error"
```

Python - Les bases - Les dictionnaires

```
# Un dictionnaire une liste avec des clés numériques.

monDico = {}

monDico["name"] = "Mehdi"

monDico["height"] = "1m90"

print(monDico)
```

Python - Les bases - Les dictionnaires

```
print(monDico.get("name"))
print("name" in monDico)
del monDico["name"]
for val in monDico.values(): # Affiche les valeurs
  print(val)

for key in monDico.keys(): # Affiche les index
  print(key)
```

Python - Les bases - Les fonctions

```python
def ma_function():
  print("salut les gens")

ma_function()

def ma_function2(param):
  print(param)

ma_function2("hey")
```

```python
def somme(a, b):
  return a + b

somme = somme(1, 2)

print(somme)
```

Python - Les bases - Les fonctions

```
def splat_function(*params):
  for item in params:
     print(item)

splat_function(1, 2, "salut")
```

Python - Les bases - Les fonctions

```
# Portée des variables
a = "salut"
c = 5
def test():
  b = "test"
  print(c)

test()
print(a)
print(b)
```

```
# Le return

def test():
  return "salut"
a = test()
print(a)
```

Python - Les bases - Les fonctions natives

Listes : https://docs.python.org/3/library/functions.html

```
# Récupérer une entrée dans le terminal
valeur = input("Enter your value")

# Retourne une valeur d'une liste aléatoirement.
random.choice([1,2,3,4,5])
```

Python - Les bases - Les conditions

```
a = 10
if a > 5:
  print("ok")
b = 1
if b > 5:
  print("ok")
else:
  print("pas ok")
```

```
a = 10
if a < 5:
  print("ok")
elif a == 9:
  print("a = 9")
else:
  print(a)
```

Python - Les bases - Les conditions

== égal à
!= différent de (fonctionne aussi avec)
> strictement supérieur à
>= supérieur ou égal à
< strictement inférieur à
<= inférieur ou égal à

Il est possible d'affiner une condition avec les mots clé AND qui signifie "ET" et OR qui signifie "OU".

Python - Les bases - Les boucles

```
# While
i = 0
while i < 10:
  print(i)
  i = i + 1
i = 0
while i < 10:
  print(i)
  i += 1
```

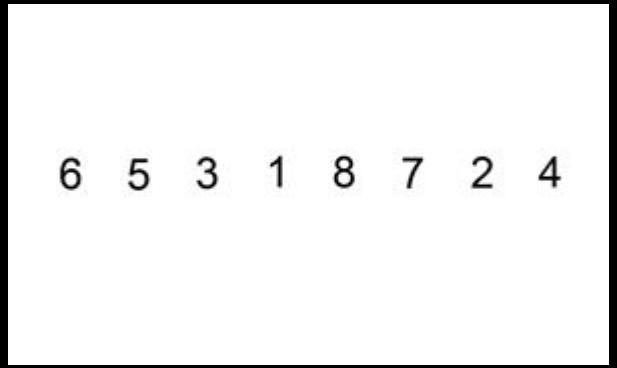
```
s = "salut les gens"
for lettre in s:
   print(lettre)
   if lettre == "t":
      break
# For
s = "salut les gens"
for lettre in s:
   print(lettre)
```

Python - Les bases - Le tri à bulle

Le tri à bulles ou tri par propagation est un algorithme de tri. Il consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés. Il doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.

```
[4, 8, 415, 1, 25, 75, 6]
```

Python - Les bases - Le tri par insertion




```
[4, 8, 415, 1, 25, 75, 6]
```

Python - Les bases - Les tries

```
def bubule():
    list = [4, 8, 415, 1, 25, 75, 6]
    last = True
    limit = len(list)
    while last:
        for i in range(0, limit-1):
            last = False
            if (list[i] > list[i+1]):
                list[i+1], list[i] = list[i],
list[i+1]
                last = True
        limit = limit - 1
    print(list)
```

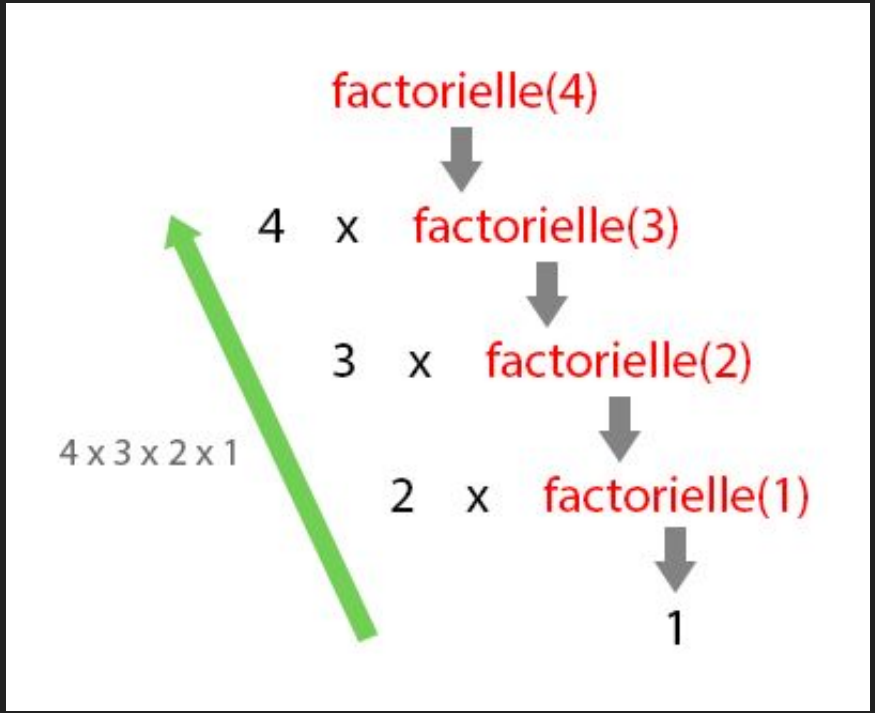
```
def insertion():
    list = [4, 8, 415, 1, 25, 75, 6]
    for i in range(1, len(list)):
        current = list[i]
        j = i
        while j > 0 and list[j - 1] > current:
            list[j] = list[j - 1]
            j = j - 1
        list[j] = current
    print(list)
```

Python - Les bases - Algo de recherche

```
def search(arr, x):
  for i in range(len(arr)):

    if arr[i] == x:
      return i

  return -1
```

Python - Les bases - La récursivité

La récursivité est une démarche qui fait référence à l'objet même de la démarche à un moment du processus. En d'autres termes, c'est la propriété de pouvoir appliquer une même règle plusieurs fois en elle-même.



Python - Les bases - Exercices

TP 01:

- 01 Écrire un programme qui affiche 500 fois « Je dois faire des sauvegardes régulières de mes fichiers. »
- 02 Écrire un programme qui affiche tous les nombres impairs entre 0 et 1000, par ordre croissant : « 1 3 5 7 ... 995 997 999 »
- 03 Écrire un programme qui affiche la table de multiplication par 13
- 04 Écrire un programme qui demande un mot à l'utilisateur et qui affiche à l'écran le nombre de lettres de ce mot.
- 05 Ecrire un programme qui demande un nombre entier à l'utilisateur. L'ordinateur affiche ensuite le message "Ce nombre est pair" ou "Ce nombre est impair" selon le cas.

Python - Les bases - Exercices

- 06 Ecrire un programme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.
- 07 Ecrire un programme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.
- 08 Ecrire un programme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre.
- 09 Ecrire un programme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer : 1 + 2 + 3 + 4 + 5 = 15, afficher que le résultat
- 10 Écrire un programme qui demande l'âge d'un enfant à l'utilisateur. Ensuite il l'informe de sa catégorie :
    - "Poussin" de 6 à 7 ans
    - "Pupille" de 8 à 9 ans
    - "Minime" de 10 à 11 ans
    - "Cadet" après 12 ans

Python - Les bases - Exercices

- 11 Ecrivez un programme qui calcule le prix TTC d'un nombre donné d'articles de prix unitaire donné.
Avec une T.V.A. à 20%.
Les résultats devront se présenter ainsi :

    nombres d'articles : 5
    prix HT : 42.15 €
    Prix TTC : 252.06 €

- 12 Écrire un programme qui calcule la factorielle de n.
Exemple facto de 10 = 3.628.800

Python - Les bases - Exercices

- 13 Écrire un programme qui convertit un nombre décimal (base 10) en binaire (base 2). Fonction bin interdite
- 14 Si nous listons tous les nombres naturels inférieurs à 10 qui sont des multiples de 3 et 5, nous avons 3, 5, 6 et 9. La somme de ces multiples est 23. Trouvez la somme de tous les multiples de 3 et 5 inférieurs à 1000.
- 15 Écrire un programme qui affiche le 1500ème nombre de la suite de Fibonacci.
- 16 Écrire un programme qui affiche le plus petit nombre positif divisible par tous les nombres de 1 à 20 avec un résultat entier.

Python - Les bases - L'enfer du nommage

Une convention de nommage dans la programmation informatique est un ensemble de règles de codage destinées à choisir les identifiants logiciels (noms des éléments du programme) dans le code source et la documentation.
Objectif:

- Rendre le code plus facile à comprendre
- Rendre l'application plus facilement maintenable

De loin la chose la plus complexe dans le métier de développeur.

Python - Les bases - Les modules et packages

Créons un autre fichier que nous nommerons func.py dans le même dossier que le fichier script.py

```
# func.py

def addition(a, b):
  return a + b
```

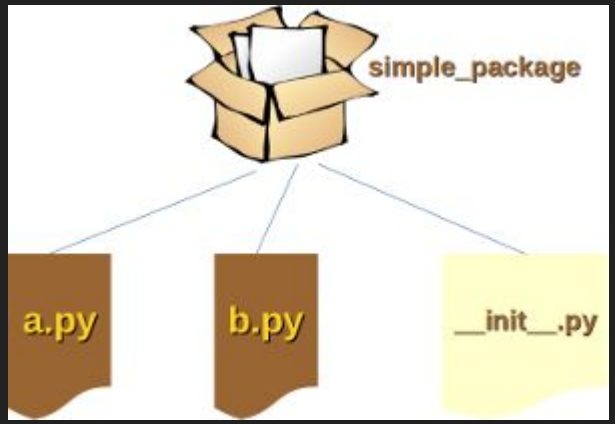
```
# script.py

from func import *

print addition(5, 10)
```

Python - Les bases - Les modules et packages

Module : Un fichier qui comprend plusieurs fonction.

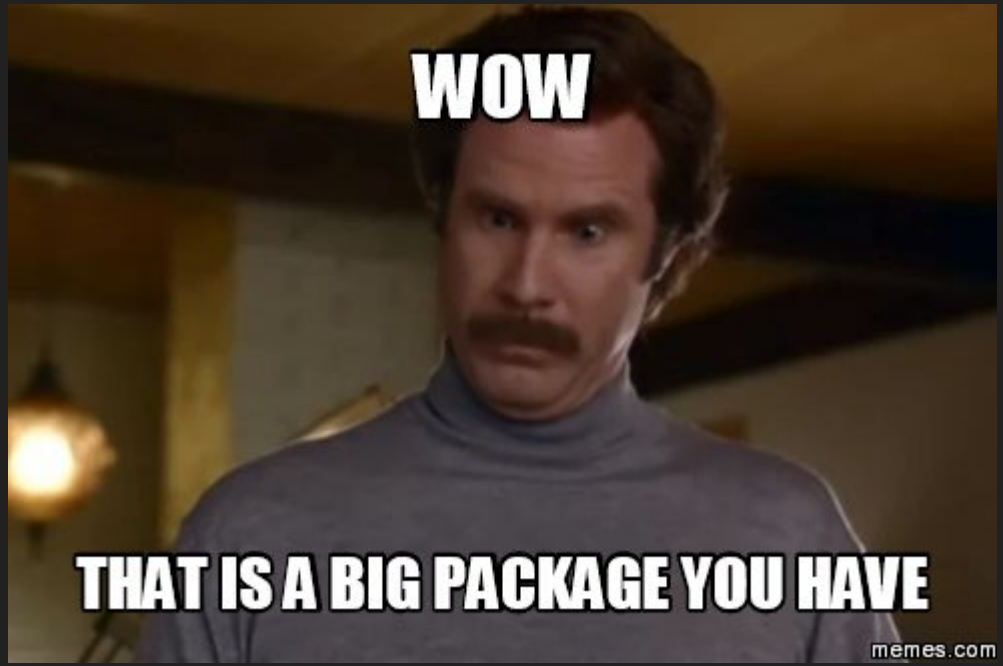
Package : Ensemble de Module.

Création d'un package:
A côté du fichier script:
On crée un dossier tools.
Un fichier __init__.py vide.
Avec un fichier a.py et la fonction addition.



Python - Les bases - Les modules et packages

```python
# script.py

from tools.a import
addition

print(addition(5, 10))
```



Python - Les bases - Les modules et packages

```python
import time

def sleep():
  print("sleep")

sleep()
time.sleep(3)
```

```python
from time import *

def sleep():
  print("sleep")

sleep()
time.sleep(3)
```

Python - Les bases - Les exceptions

Try signifie "essayer" en anglais, ce mot clé permet d'essayer une action et si l'action échoue on peut lui donner d'autres instructions dans un bloc except.

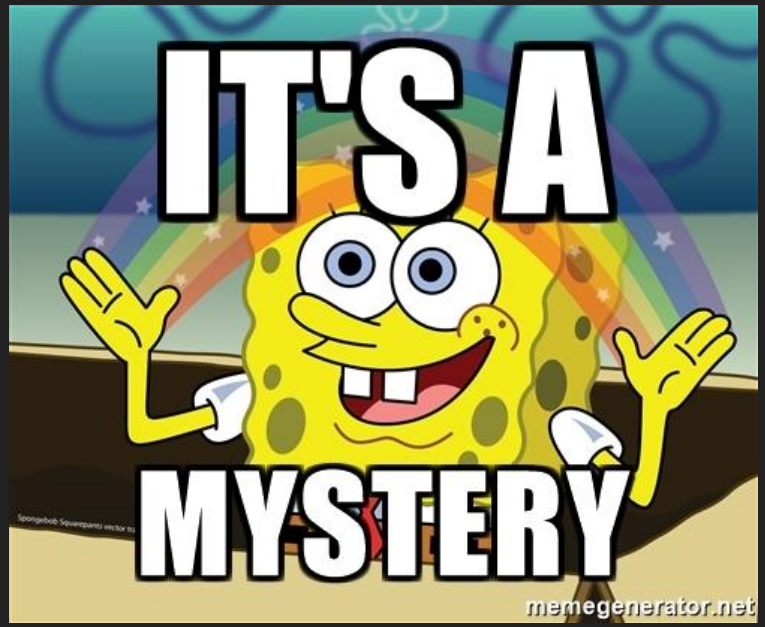
```
a = 1
b = 0

try:
  a/b
  print("oki")
except:
  print("Error")
```

```
a = 1
b = 0

try:
  a/b
  print("oki")
except TypeError:
  print("Error")
except ZeroDivisionError:
  print("Error division by 0")
```

Python - Les bases - Le jeu

TP 02: Jeu du nombre mystère.

Le but du jeu est de deviner un nombre généré de manière aléatoire par l'ordinateur (nombre entre 1 et 100). Pour jouer c'est très simple, tu écris à l'aide de ton clavier le nombre que tu penses être le bon et l'ordinateur t'indique si celui-ci est plus grand ou plus petit que le nombre à deviner. Tu peux jouer autant de fois que tu veux et le nombre de tentatives est limité à 9.



Python - Les bases - Les décorateurs

```
def enhancement(func):
  print("decorateur")
  return func


@enhancement
def action():
   print("action en
action")

action()
```

```
def enhancement(func):
  def other():
     print("other")
  return other


@enhancement
def action():
   print("action en action")

action()
```

Python - Les bases - Les itérateurs

```
class MonIter():
    current = 0
    def __init__(self, stop):
        self.stop = stop
    def next(self):
        self.current += 1
        if self.current > self.stop:
            raise StopIteration
        if self.current == 5:
            print("Quoi déjà 5eme tour?")
        return self.current

for i in MonIter(10):
  print(i)
```

Python - Les bases - Les générateurs

```
def generateur(n):
    for i in range(n):
        if i == 5:
            print("Quoi déjà 5eme tour?")
        yield i + 1


for i in generateur(10):
    print(i)
```

Python - Les bases - Les fichiers

```
fichier = open("data.txt", "r")
print(fichier.read())
fichier.close()

fichier = open("data.txt", "a")
fichier.write('salut les gens\n')
fichier.close()
```

r, pour une ouverture en lecture (READ).
w, pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas python le crée.
a, pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée.
b, pour une ouverture en mode binaire.
t, pour une ouverture en mode texte.
x, crée un nouveau fichier et l'ouvre pour écriture

Python - Les bases - Le XML (berk)

pip install lxml

```
from lxml import etree

users = etree.Element("users")
user = etree.SubElement(users, "user")
user.set("data-id", "1")
nom = etree.SubElement(user, "nom")
nom.text = "Mehdi"
metier = etree.SubElement(user, "metier")
metier.text = "Developpeur"

fichier = open("data.xml", "w")
fichier.write(etree.tostring(users,
pretty print=True))
fichier.close()
```

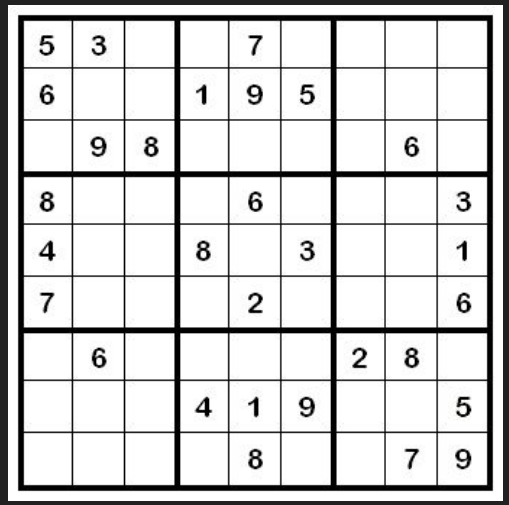
```
from lxml import etree

tree = etree.parse("data.xml")
for user in
tree.xpath("/users/user/nom"):
   print(user.text)
```

Python - Les bases - Le Web (miam)

```
import http.server
import socketserver

PORT = 8000
Handler = http.server.SimpleHTTPRequestHandler
httpd = socketserver .TCPServer(("", PORT), Handler)

httpd.serve_forever()
```

Python - Les bases - Exercices

Créer un algo qui génère une grille de sudoku avec 9 cases.

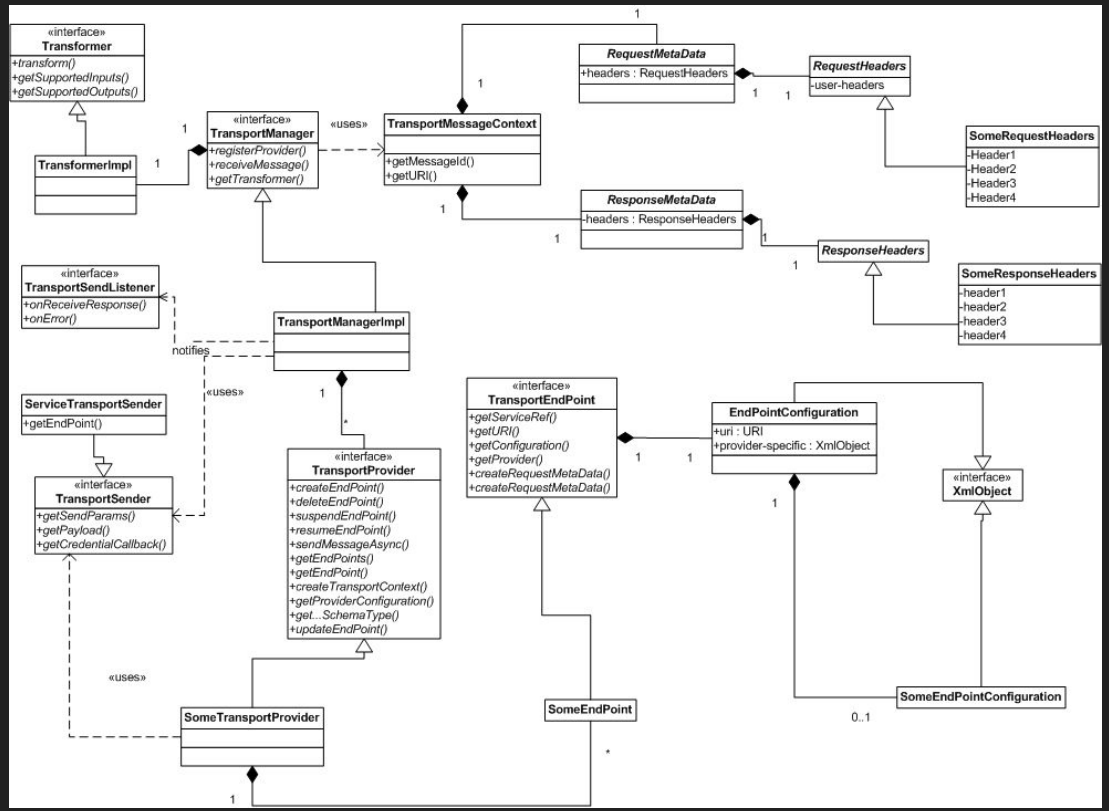


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Python - Programmation orientée objet (POO)

La programmation orientée objet (POO) permet de créer des entités (objets) que l'on peut manipuler. La programmation orientée objet impose des structures solides et claires. Les objets peuvent interagir entre eux, cela facilite grandement la compréhension du code et sa maintenance. On oppose souvent la programmation objet à la programmation procédurale, la première étant plus "professionnelle" que l'autre car plus fiable et plus propre.

Python - Programmation orientée objet (POO)



Python - Programmation orientée objet (POO)

Une classe regroupe des fonctions et des attributs qui définissent un objet. On appelle par ailleurs les fonctions d'une classe des "méthodes".

Créons une classe Voiture

```
class Voiture:
  def __init__(self):
      self.nom = "Ferrari"

ma_voiture = Voiture()
print(ma_voiture.nom)
```

Python - Programmation orientée objet (POO)

Les attributs de classe permettent de stocker des informations au niveau de la classe. Elle sont similaires aux variables.

```
ma_voiture = Voiture()
print(ma_voiture.nom)

ma_voiture.couleur = "Rouge"
print(ma_voiture.couleur)
```

Python - Programmation orientée objet (POO)

Les méthodes sont des fonctions définies dans une classe.
Une propriété est juste un déguisement qu'on met sur une méthode pour qu'elle ressemble à un attribut.

```
class Voiture:
  def vitesseMax(self):
      return "500km/h"

ma_voiture = Voiture()
print(ma_voiture.vitesseMax())
```

Python - Python 2 VS Python 3

Il y a eu beaucoup de changements avec la sortie de python 3.
Au niveau de:

- Du print
- Des exceptions
- Des changements au niveaux des noms de module
- L'organisation des projets
- etc…

Python 3 est plus sympa, mais pas révolutionnaire.
Fin de support pour python en 2020

Python - Encodage

Un jour ou l'autre vous tomberez sur une erreur d'encodage et vous y passerez des heures pour comprendre d'où vient le problème.

UTF-8 est un encodage universel qui a pour objectif de réunir les caractères utilisés par toutes les langues.

Par défaut dans python 2.7 l'encoding est ASCII, il est donc nécessaire d'indiquer l'encodage UTF8 à chaque fois.
Pour cela il vous faudra indiquer dans l'en tête l'encodage UTF-8 les lignes suivantes:

```
#!/usr/bin/env python
# coding: utf-8
```

Python - PIP

Une des forces de python est la multitude de bibliothèques disponibles -près de 6000 bibliothèques gravitent autour du projet Django par exemple-. Installer une bibliothèque peut vite devenir ennuyeux: trouver le bon site, la bonne version de la bibliothèque, l'installer, trouver ses dépendances, etc.

Il existe une solution qui vous permet de télécharger très simplement une bibliothèque: pip

Install : **pip install django**

Remove: **pip uninstall django**

Python - tKinter

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

from tkinter import *

fenetre = Tk()

label = Label(fenetre, text="Hello World")

label.pack()

fenetre.mainloop()
```

Python - Réseau

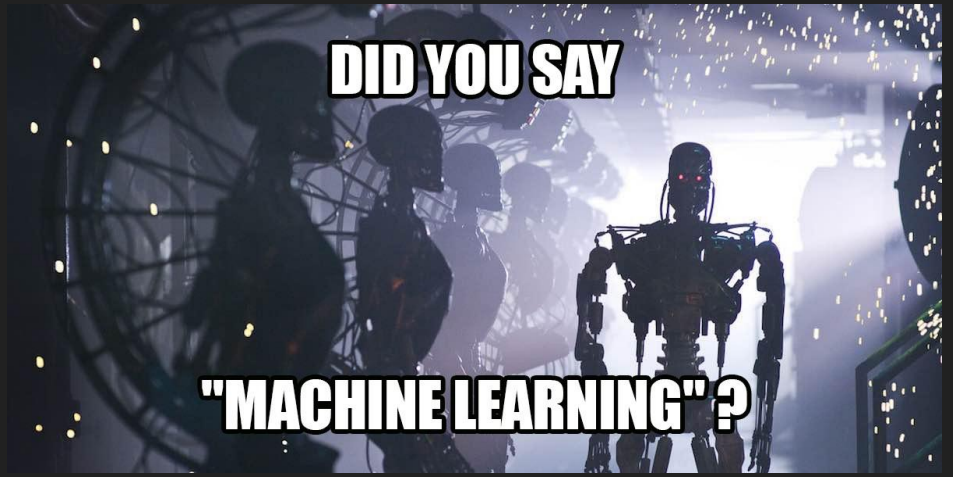
```
#!/usr/bin/python3
# coding: utf-8

import socket

ip = socket.gethostbyname(socket.gethostname()) # Ip local de la machine

print(ip)
```

Python - Réseau

```
import os

subnet = "192.168.0."

for i in range(1, 255):
    hostname = subnet + str(i)
    response = os.system("ping -n 1 " + hostname)
    if response == 0:
        print(hostname, 'is up!')
```

Python - Machine Learning

L'apprentissage automatique (en anglais machine learning, littéralement « l'apprentissage machine ») ou apprentissage statistique, champ d'étude de l'intelligence artificielle, concerne la conception, l'analyse, le développement et l'implémentation de méthodes permettant à une machine (au sens large) d'évoluer par un processus systématique, et ainsi de remplir des tâches difficiles ou problématiques par des moyens algorithmiques plus classiques.



Python - Sécurité - Génération d'un exe

Avec py2exe

```
# script.py
import easygui

easygui.fileopenbox()
```

```
# setup.py
from distutils.core import
setup
import py2exe

setup(console=['script.py'])
```

```
$ python setup.py install
```

Python - Le web avec python

pip install django

django-admin.py startproject tuto_django

cd tuto_django

python manage.py runserver

Ensuite cliquez ici : http://127.0.0.1:8000/

Python - Lexique

API: Application Programming Interface

REST: Representational State Transfer

SOAP: Simple Object Protocol

N-Tiers: Architecture 2 et/ou 3 tiers, Couche présentation (HTML/CSS), couche de traitement/logique(Python/PHP/Java) et couche d'accès aux données (bdd).

Back-end / Front-end: Dépend du scope de travail

Paradigme: Un paradigme de programmation est une façon d'approcher la programmation informatique et de traiter les solutions aux problèmes et leur formulation dans un langage de programmation approprié.

Python - Les outils

- Hug
- Zappa
- Scapy
- Scipy
- Beautiful
- tKinter
- Pygame
- Pywin32
- Py2exe
- PyInstaller
- etc...