Introduction au Python

Mickaël BOLNET

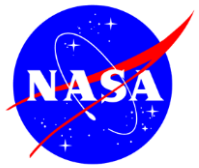
Historique

- **HISTORIQUE**
- Créé en 1989 par Guido van Rossum
- 1991 : première version publique (0.9.0)
- 2001 : Fondation Python
- 2008 : Python 3
- 2005 : Guido Van Rossum rejoint Google
- 2012 : Guido Van Rossum rejoint Dropbox


YouTube


Dropbox


reddit


NASA

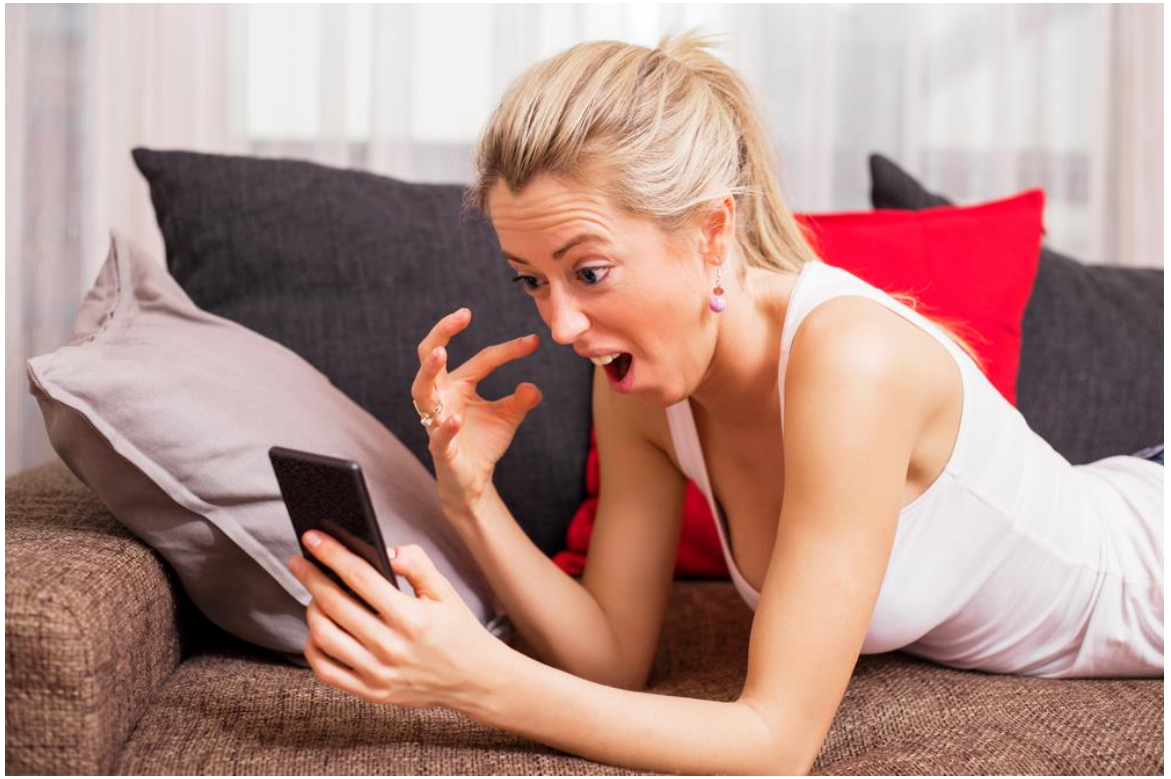

Instagram

Qu’est-ce que Python ?

- Open Source
- Langage interprété
- Multiplate-formes
- Multi-paradigmes
- Haut niveau
- 2 fois « programming language of the year » TIOBE (2007 et 2010)

Particularité

Python 2.7 ou Python 3 ?



Premiers pas

Installation

- Python
- Pip
- Jupyter
- Anaconda
- PyCharm / Sublime Text

Premier programme

```
print('hello world!')
```

Les variables

```
a = 10
b = 11
print(a)
print('a')
print(b)
print(a + b)
```

Les types

- Int (1, 2, 3 …)
- Float (1.2, 3.14 …)
- Complex (1+2i, 2+2i …)
- Bool (True or False)

- Chaine de caractères ('10’, "10")
- List : ie [1,2,3,4]
- Tuple : ie (2,2)
- Dictionary : {‘nom’ : ‘BOLNET’, ‘prenom’ : ‘Mickae’}

Opérations

x + y	**Addition**
x - y	Soustraction
x * y	Multiplication
x / y	Division
x // y	Division entière
x % y	Reste
-x	Opposé
+x	
x ** y	Puissance

Opérateurs binaires

x | y	**Ou binaire**
x ^ y	Ou exclusif
x & y	Et binaire
x << y	Décalage à gauche
x >> y	Décalage à droite
~ x	Inversion

Opérateurs sur les séquences

x not in s	**False si s contient x, sinon True**
s1 + s2	Concaténation
s * n	Répétition
s[i]	Élément à l'indice ou clef i
len(s)	Taille de la chaine
min(s)	Plus petit élément de la séquence
max(s)	Plus grand élément de la séquence
s.index(x)	Indice de la première occurence de x
s.count(x)	Nombre total d'occurrences de x

Les séquences

- Accès à un caractère

```
ma_chaine = "Bonjour"
print(ma_chaine[0])
print(ma_chaine[1])
```

- Modification

```
mon_tableau = [3, 5, '1', False]
mon_tableau[0] = "D"
```

ATTENTION: Les chaines de caractères et les tuples ne sont pas modifiables

Les séquences

Slicing

```
mon_tableau = [3, 5, '1', False]
print(mon_tableau[0:2])
print(mon_tableau[:2])
print(mon_tableau[2:])
print(mon_tableau[2:-1])
print(mon_tableau[0:4:2])
print(mon_tableau[::-1])
print(mon_tableau[4:0:-1])
```

Interractions et affichage

```
name = input('Quel est votre nom ? ')
age = int(input("quel est votre âge ? "))

"Ma variable : %type" % var
"Mes variables : %type, %type" % (var1, var2)
"Resultat : %(val)type %(unit)type" % {'val':var1, 'unit':var2}

type est d : entier - f : flottant - s : chaîne de caractère - c : caractère - o : octal - x : hexadécimal - c : caractère

Précision pour les float :
• "Resultat: %.2f" % 3.141592653589793
```

Avec format

- `Syntaxe : string.format(*args)`
- `"Résultat : {}".format(var)`
- `"Résultat : {}, {}".format(var1, var2)`
- `"Résultat : {1} {0}".format(var1, var2)`
- `"Résultat : {value} {unit}".format(unit=var1, value=var2)`
- `"Résultat : {:5.2f}".format(var)`
- `"Résultat : {value:5.2f} {unit}".format(unit=var1, value=var2)`

Les conditions

Structure conditionelle

```
name = 'Mickael'
if name == 'Mickael':
    print('Bonjour Mickael')
elif name == 'Laetitia':
    print('Bonjour Laetitia')
else:
    print('Vous n\'avez pas le droit de rentrer')
```

Opérateurs de comparaison

<	Strictement inférieur à
>	Strictement supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

Logique booleene

- OR
- NOT
- AND

BONUS : opérateur ternaire

```
gender = 'masculin' if name == 'Mickael' else 'feminin'
```

Les boucles

Les boucles (while)

```
nb = 7
i = 0

while i < 10:
    print(i + 1, "*", nb, "=", (i + 1) * nb)
    i += 1
```

Les boucles (for)

```
for i in range(5):
    print(i)

for i in range(3, 6):
    print(i)

for i in range(4, 10, 2):
    print(i)

for i in range(0, -10, -2):
    print(i)
```

Break and continue

```
while 1:
    lettre = input("Tapez 'Q' pour quitter : ")
    if lettre == "Q":
        print("Fin de la boucle")
        break
    elif lettre == "N":
        print("Vous avez tapé N")
        continue
```

Les fonctions

Les fonctions

```
def dire_bonjour():
    print('Bonjour Monsieur!')

def dire_bonjour(name):
    print('bonjour ' + name)

def dire_bonjour(name, name2=''):
    print('bonjour ' + name + ' ' + name2)
```

Portée des variables

```
foo = 1
def test_local():
foo = 2 # new local foo
def test_global():
global foo
foo = 3 # changes the value of the global foo
```

Fonctions lambda

```
def print_result(var, function):
    print(function(var))

print_result(4, lambda x: x * 2)
```

Fonctions génératrices

- Elles ne peuvent être parcourues qu'une seule fois
- On ne peut accéder à un élément par un indice

```
def countfrom(x):
    while True:
        yield x
        x += 1

for n in countfrom(10):
    print n
    if n > 20: break
```

Gestion des fichiers

Ouvrir, lire et écrire dans un fichier

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


fichier = open("data.txt", “a")
fichier.write("Bonjour monde")
fichier.close()

with open("data.txt", "r") as fichier :
    print(fichier.read())
```

Types d’ouvertures

- **r**, pour une ouverture en lecture (READ).
- **w**, pour une ouverture en écriture (WRITE), à chaque ouverture le con tenu 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

Les répertoires

- os.mkdir(chemin, mode) : crée répertoire, mode UNIX
- os.remove(chemin) : supprime fichier
- os.removedirs(chemin) : supprime répertoires récursivement
- os.rename(chemin_old, chemin_new) : renomme fichier ou répertoire
- os.renames(chemin_old, chemin_new) : renomme fichier ou répertoire en créant les répertoires si ils n'existent pas
- os.chdir(chemin) : change le répertoire de travail
- os.getcwd() : affiche répertoire courant

Les répertoires

- os.path.exists(chemin) : est-ce que le fichier ou répertoire existe
- os.path.isdir(chemin) : est-ce un répertoire
- os.path.isfile(chemin) : est-ce un fichier
- os.listdir(chemin) : liste un répertoire

Utiliser le module glob qui permet l'utilisation de wildcards

- glob.glob(pattern) : liste le contenu du répertoire en fonction du pattern

Les répertoires

- shutil.move(src, dest) : déplace ou renomme un fichier ou un répertoire
- shutil.copy(src, dest) : copie un fichier ou un répertoire
- shutil.copy2(src, dest) : copie un fichier ou un répertoire avec les métadonnées
- os.chmod(path, mode) : change les permissions
- os.path.dirname(path) : retourne l'arborescence de répertoires
- os.path.basename(path) : retourne le nom du fichier
- os.path.split(path) : retourne un tuple des deux précédents
- os.path.splitext(path) : retourne un tuple pour obtenir l'extension

Les exceptions en bref

```
annee = input()
try: # On essaye de convertir l'année en entier
    annee = int(annee)
except:
    print("Erreur lors de la conversion de
l'année.")
finally:
    print("S'affiche de toute manière.")
```

Les exceptions en bref

```
raise TypeDeLException("message à afficher")
```

Les exceptions en bref

```
try:
    resultat = numerateur / denominateur
except NameError:
    print("La variable numerateur ou denominateur
n'a pas été définie.")
except TypeError:
    print("La variable numerateur ou denominateur
possède un type incompatible avec la division.")
except ZeroDivisionError:
    print("La variable denominateur est égale à 0.")
```

Modules et Packages

Modules et Packages

Commencent par :

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

Doit contenir un __init__.py :

```
MyPackage/
        __init__.py
        MyModule.py
        MyModule2.py

__all__ = [ 'MyModule', 'MyModule2']
```

Importer les modules

```
import MyModuleLibrary.MyModule
import MyModuleLibrary.MyModule2

MyModuleLibrary.MyModule.function_welcome()
MyModuleLibrary.MyModule2.function_welcome_bis()
```

Module __name__

```
if __name__ == '__main__':
    giveAnswer()
```

La Programmation Orientée Objet (POO)

Les paradigmes de programmation

Il s'agit des différentes façons de raisonner et d'implémenter une solution à un problème en programmation.

- La programmation impérative : paradigme originel et le plus courant
- La programmation orientée objet (POO) : consistant en la définition et l'assemblage de briques logicielles appelées objets
- La programmation déclarative consistant à déclarer les données du problème, puis à demander au programme de le résoudre
- Fonctionnelle …

Les Objets

Caractérisés par

- Un état : ses attributs
- Des comportements : ses méthodes

Les Classes

Sont les définition des objets

- Un objet est une instance d'une classe
- En POO, nous définissons des classes
- En POO, nous manipulons des instances des classes
- Le type d'un objet est sa classe

Les Classes

```
class Personne:
"""Classe définissant une personne caractérisée par :
    - son nom
    - son prénom
    - son âge
    - son lieu de résidence"""



    def __init__(self):
        """Pour l'instant, on ne va définir qu'un seul attribut"""
        self.nom = "Dupont«
        self.prenom = "Jean"
        self.age = 33
        self.lieu_residence = "Paris"
```

Les Classes

```
class Personne:
"""Classe définissant une personne caractérisée par :
    - son nom
    - son prénom
    - son âge
    - son lieu de résidence"""


    def __init__(self, nom, prenom):
        ""constructeur""
        self.nom = nom
        self.prenom = prenom
        self.age = 33
        self.lieu_residence = "Paris"
```

Les Classes

```
class Compteur:
    """Cette classe possède un attribut de classe qui s'incrémente à chaque
    fois que l'on crée un objet de ce type"""


    objets_crees = 0 # Le compteur vaut 0 au départ
    def __init__(self):
        """À chaque fois qu'on crée un objet, on incrémente le compteur"""
        Compteur.objets_crees += 1
```

Méthodes spéciales

- __init__(self) : initialiseur appelé juste après l'instanciation d'un objet
- __del__(self) : destructeur, appelé juste avant la destruction de l'objet
- __str__(self) -> str : est appelé par la fonction de conversion de type str() et par la fonction print(). Elle doit donc retourner une chaine de caractères représentant l'objet.
- __repr__(self) -> str : est appelé par la fonction repr() et doit retourner une chaine de caractères contenue entre des chevrons et contenant non, type de l'objet et informations additionnelles.

Méthodes spéciales

Méthode	Opération
__lt__(self, other)	x < y
__le__(self, other)	x <= y
__eq__(self, other)	x == y
__ne__(self, other)	x != y
__ge__(self, other)	x >= y
__gt__(self, other)	x > y

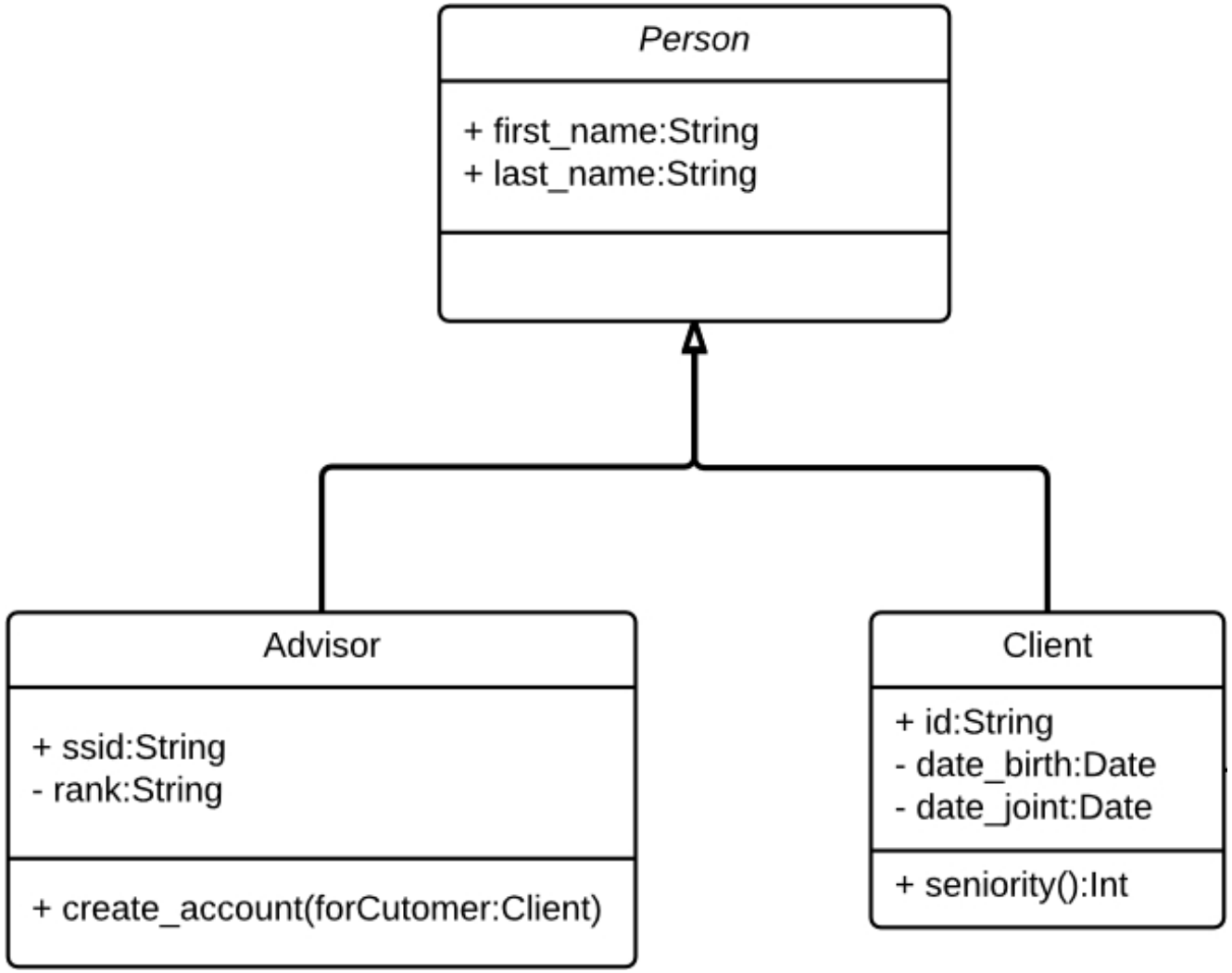
Méthodes spéciales

Méthode	Opération
__neg__	-x
__add__	x + y
__sub__	x - y
__mul__	x * y
__div__	x / y

Héritage Abstraction

- Propriété de généraliser ou spécialiser des états ou comportements
- Généralisation : définition unique, évite duplication
- Spécialisation : adapter caractéristiques et comportements
- Abstraction
- Polymorphisme

Héritage Abstraction



Polymorphisme

- Possibiliter de redéfinir « a posteriori » un comportement »
- Le système choisit dynamiquement la méthode à exécuter sur l'objet en cours, en fonction de son type réel.

Exemple :

- Pour Mercedes, accélère() augmente la vitesse de 10 km/h
- Pour Clio, accélère() augment la vitesse de 2km/h

Les Bases de données

Base de données

Principe général

- Établir une connexion
- Créer un curseur et lui attribuer une requête
- Exécuter la requête
- Itérer sur les éléments retournés
- Fermer la connexion

Base de donnée SQL

```
import MySQLdb
try:
    conn = MySQLdb.connect(host='localhost’,
        user='test user’,
        passed='test pass’,
        db='test’)
    cursor = conn.cursor()
    cursor.execute("SELECT VERSION()")
    row = cursor.fetchone()
    print('server version', row[0])
finally:
    if conn:
        conn.close()
```

Quelques requêtes PostGre courantes

```
CREATE TABLE COMPANY( ID INT PRIMARY KEY NOT NULL,
            NAME TEXT NOT NULL,
            AGE INT NOT NULL,
            ADDRESS CHAR(50),
            SALARY REAL,
            JOIN_DATE DATE );
```

```
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY,JOIN_DATE)
            VALUES  ('Mark', 25, 'Rich-Mond ', 65000.00, '2007-12-13' ),
                    ('David', 27, 'Texas', 85000.00, '2007-12-13');
```

```
DELETE FROM COMPANY WHERE ID = 2;
```

```
DELETE FROM COMPANY;
```

```
SELECT column1, column2, columnN FROM table_name LIMIT 10 OFFSET 20 ORDER BY AGE ASC
```

```
UPDATE COMPANY SET salary = 15000 WHERE ID = 2;
```

Quelques requêtes PostGre courantes

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME ORDER BY NAME;
```

Autres fonctions : COUNT / MAX / MIN / AVG

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY
INNER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

Autres JOIN : LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN

PyQT

Les interfaces graphiques

Introduction à PyQT

- QtCore
- QtGui
- QtNetwork
- QtXml
- QtSvg
- QtOpenGL
- QtSql

```
from PyQt4 import QtCore .
from PyQt4 import QtGui .
from PyQt4 import QtNetwork .
from PyQt4 import QtXml .
from PyQt4 import QtSvg .
from PyQt4 import QtOpenGL .
from PyQt4 import QtSql .
```

Introduction à PyQT

```
def main():

    app = QtGui.QApplication(sys.argv)

    w = QtGui.QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Simple')
    w.show()

    sys.exit(app.exec_())
```

PyQT with OOP

```
class Example(QtGui.QWidget):

    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Icon')
        self.setWindowIcon(QtGui.QIcon('web.png'))
        self.show()
```

PyQT – Add a button

```
qbtn = QtGui.QPushButton('Quit', self)
qbtn.clicked.connect(QtCore.QCoreApplication.instance().quit)
qbtn.resize(qbtn.sizeHint())
qbtn.move(50, 50)
```

PyQT – Add menu and toolbar

```
exitAction = QtGui.QAction(QtGui.QIcon('exit.png'), '&Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(QtGui.qApp.quit)

        self.statusBar()

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(exitAction)

        self.toolbar = self.addToolBar('Exit’)
        self.toolbar.addAction(exitAction)
```

PyQT – Le positionnement

```
# Positionnement absolute
lbl1 = QtGui.QLabel('ZetCode', self)
lbl1.move(15, 10)

# Positionnement par box
hbox = QtGui.QHBoxLayout()
hbox.addStretch(1)
hbox.addWidget(okButton)
hbox.addWidget(cancelButton)

# Positionnement par grille
grid = QtGui.QGridLayout()
self.setLayout(grid)
grid.addWidget(button, *position)
```

PyQT – Les événements

```
def keyPressEvent(self, e):
        if e.key() == QtCore.Qt.Key_Escape:
            self.close()

btn1.clicked.connect(self.buttonClicked)
def buttonClicked(self):
        sender = self.sender()
        self.statusBar().showMessage(sender.text() + ' was pressed')
```

PyQT – Les événements

```
class Communicate(QtCore.QObject):
    closeApp = QtCore.pyqtSignal()

def initUI(self):
        self.c = Communicate()
        self.c.closeApp.connect(self.close)

def mousePressEvent(self, event):
        self.c.closeApp.emit()
```