Les bases de Python

Python est un langage informatique développé fin 1989 par Guido van Rossum (CWI, Amsterdam, Pays-Bas).

1 Introduction

Avant de commencer [1], il faut savoir qu'un ordinateur communique avec nous et avec lui-même avec différents types de langage plus ou moins évolués. La base de tout ces langages repose sur les opérations classiques que sont l'addition de deux nombres, leur soustraction, leur multiplication, et leur division, entière ou non. La combinaison de ces cinq opérations dans des fonctions plus ou moins évoluées suffit amplement à faire fonctionner les logiciels de simulation les plus complexes ou les jeux super-réalistes. Tous ces logiciels fonctionnent en gros de la même façon : une suite d'instructions écrites en langage machine compose le programme ; lors de l'exécution du programme, ces instructions décrivent à l'ordinateur ce qu'il faut faire.

1.1 Les différents types de langage

L'ordinateur parle nativement en binaire, ce qui n'est pas très pratique à coder. Les différents interpréteurs développés ont pour but de transcrire des instructions dans un langage *faciles à coder* en binaire pour pouvoir communiquer avec la machine.

Il existe au moins deux grandes familles de langages informatiques :
- les langages dits **compilés** dans lesquels les instructions sont écrites dans un fichier puis compilée par un *traducteur*, logiciel spécialisé qui permet de les convertir en langage machine. Ex : C, C++
- les langages dits **interprétés** où les instructions que vous lui envoyez sont *transcrites* en langage machine au fur et à mesure de leur lecture. Ex : Python, java, php

Les avantages d'un langage interprété sont la simplicité et la portabilité (un langage tel que Python est censé fonctionner aussi bien sous Windows que sous Linux ou Mac OS, et on ne devrait avoir à effectuer aucun changement dans le code pour le passer d'un système à l'autre). Cela ne veut pas dire que les langages compilés ne sont pas portables, loin de là ! Mais on doit utiliser des compilateurs différents et, d'un système à l'autre, certaines instructions ne sont pas compatibles, voire se comportent différemment.

1.2 Qu'est-ce que Python

Python est un langage :
- interprété écrit en C.
- open-source : chacun peut ajouter sa pierre à l'édifice, le logiciel est gratuit et supporté par une très grande communauté.
- utilisé dans de nombreux domaines de l'industrie et de la recherche.

Python est de plus en plus utilisé car il s'interface avec un très grand nombre d'autres langages. Ainsi, il peut être utilisé aussi bien pour commander des machines, faire des simulations numériques analyser des données, tracer des courbes, ou créer des logiciels comme ... Libreoffice, Google, Gimp !

2 À vos claviers

2.1 Opérations simples

Commencer par attribuer une valeur à une variable :

```
a=5
b=2
```

1. cette introduction est largement inspirée de l'openclassroom https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python/230659-quest-ce-que-python

Tapez ensuite a puis entrez dans le terminal Python, le résultat est 5. Cette opération consiste à allouer une valeur à un objet a.

Maintenant utilisez la fonction type() appliquée à la variable a : type(a). Le résultat est int, ce qui signifie que a est un entier (integer en anglais).
En effet, par définition dans python, un nombre tapé sans virgule est un entier. Ceci est important pour la division car sous python il existe deux types de division : la division d'entier et la division classique.

Exemple : tapez b/a, le résultat est zéro : la division entière de b par a est nulle car b n'est pas un multiple de a. Maintenant, changez la valeur de b par b=2.0 , puis refaites les opérations précédentes. b est désormais un *float*, et la division est possible.

Objets Python

Il existe plusieurs types d'objet en Python :
- integer : les entiers
- float : nombres relatifs
- Complex numbers : nombres complexes
- String : texte
- Booléen : True / False/ None
- array : tableau dont **la taille doit être déclarée avant utilisation** et ne peut pas varier, ne peut contenir que des "float" et "integer". Les opérations mathématiques usuelles se font éléments à élément entre deux tableaux de même taille.
- Matrix : matrice, idem tableau mais avec les opérations matricielles.
- list : liste, peut contenir tout ce qu'on veut, **sa taille peut varier**, impossible de faire des opérations mathématiques sur des listes.
- dictionnaire
- fonction
- classe
- objet

2.2 Hello World

L'apprentissage de tout langage informatique commence par le fameux "Hello world" : il s'agit de faire afficher par python "Hello world". La façon la plus simple est la suivante :

```
print( 'Hello world')
```

Les guillemets permettent de stipuler à python qu'il s'agit d'une chaîne de caractère : un string. La fonction print permet de faire afficher la chaîne de caractère.

Cette façon de procéder possède un intérêt très limité : le but de coder est d'écrire une suite d'instructions dans un script puis d'exécuter l'ensemble grâce à Python et non de tout taper ligne par ligne à chaque utilisation.

Pour ce faire, nous allons générer un fichier dans un éditeur de texte quelconque (autant utiliser celui de python) qui sera sauvegardé sous la forme suivante : **nom.py** .
La première ligne du fichier sera alors :

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

Le ♯ permet de commenter la ligne : tout ce qui apparaît après un ♯ n'est pas exécuté par Python. Cette ligne stipule à l'ordinateur qu'il faut utiliser Python pour lire le script. Elle n'est pas nécessaire pour les scripts de "base".

✓Ouvrez un éditeur et taper :

```
#!/usr/bin/env python
print( 'Hello world')
```

Enregistrez le fichier sous le nom **test.py** puis dans **Edit** cliquez sur run. Le script va s'exécuter dans le terminal Python et va imprimer à l'écran les mêmes instructions que précédemment : Hello world.

2.3 Création de fonction

2.3.1 Fonctions existantes

Passons maintenant aux choses sérieuses, au coeur de la programmation : les fonctions. Il existe sous python un certain nombre de fonctions dites natives comme :

- max(), min()
- type(), format()
- dict() pour les dictionnaires
- [] pour générer des listes
- range(n) pour créer un tableau de 0 à n-1 éléments
- **help(nomfonction)** pour obtenir de l'aide sur une fonction
- ...

Pour les autres fonctions il faut avoir recours aux bibliothèques.

La bibliothèque de fonctions Python s'élargie tous les jours un peu plus grâce à ses développeurs à travers le monde. Les fonctions disponibles sont rangées dans des bibliothèques qu'il faut souvent installer avant de pouvoir les utiliser. Pour les sciences, les bibliothèques les plus courantes sont numpy (maths) et matplotlib (pour les tracer de courbes et les figures).

À l'inverse de certain logiciel comme Matlab, les bibliothèques installées ne sont pas automatiquement chargées par python, ce qui le rend extrêmement rapide et léger. L'inconvénient est qu'il faut charger manuellement les bibliothèques que l'on veut utiliser.

Ainsi un script Python commencera souvent par :

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
```

as permet de donner un nom raccourci à la bibliothèque. Pour appeler la fonction racine sqrt appliquée à a il suffit alors de taper :

```
np.sqrt(a)
```

Le point entre le nom de la bibliothèque et la fonction symbolise l'hérédité. De même pour tracer une courbe y en fonction de x il suffit de taper :

```
plt.figure() # declare une nouvelle figure
plt.plot(x,y,'bo') # trace un serie de points 'o' bleus 'b'
plt.xlabel('x') #nom de la variable x
plt.ylabel('y') #nom de la variable y
plt.grid() # Met une grille en arrire plan
plt.show() # affiche la figure
```

Il est possible de faire un simple :

```
from numpy import *
```

ce qui a pour effet de charger d'un coup toute la bibliothèque, une fonction peut alors s'appeler directement sans repréciser l'appartenance à une bibliothèque mais ce n'est pas conseillé. En effet, certaines fonctions peuvent appartenir à plusieurs bibliothèques et, tout en ayant le même nom, elles peuvent présenter de petites variations. Il est donc préférable de stipuler où Python doit aller chercher la fonction.

✓Construire un tableau appelé t de trois lignes rempli de 1 (np.ones()), multiplier le tableau par 4 et prendre la racine.

✓Générer la liste [1 2 3 4 5 6] et extraire le 3ème élément. Multiplier la liste par 3.

Bilan : Seul un tableau est soumis aux opérations mathématiques élément par élément. La numérotation des éléments en Python commence à zéro.

2.3.2 Créer de nouvelles fonctions

Les bibliothèques proposées sont souvent très complètes, mais pour des opérations simples et spécifiques il est souvent plus rapide et plus efficace de coder soit même ses fonctions.

Indentation Parmi les exemples de code présentés, vous avez dû remarquer l'alignement des lignes et les sauts de lignes entre chaque instruction. L'organisation des lignes est en effet très importante puisque Python est un langage **indenté**.

Indentation

La première chose à savoir c'est que Python utilise les sauts de ligne pour passer d'une instruction à l'autre. Le saut de ligne est à python ce que la virgule et le point sont au français. La position sur la ligne est aussi importante :

- Indenter c'est décaler horizontalement les instructions d'une ligne à l'autre de manière à indiquer une filiation. C'est une sorte d'alinéa sauf qu'à l'inverse du français, cet alinéa marque l'appartenance à un groupe d'instructions et non un changement de thème.
- une indentation = 4 espaces.
- une instruction peut être indentée plusieurs fois : 4, 8, 12 espaces consécutifs.

Nous allons voir un exemple tout de suite avec la création de boucles.

Boucles Les boucles font partie des outils les plus utilisés en programmation et permettent entre autre d'automatiser la répétition d'une opération simple. En python, l'intentation des boucles est primordiale !

Une boucle commence toujours par une condition d'entrée (de départ), continue par une action simple et doit se finir par une condition de sortie, sinon la boucle est infinie et le code ne s'arrête jamais de tourner.

En Python, les boucles commencent souvent par :

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

L'instruction for donne à la fois le début et la fin de la boucle : ici i va prendre toutes les valeurs entre 0 et 4.

Vous pouvez par exemple taper :

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

Les entiers de 0 à 4 s'imprime à l'écran. Ici l'instruction print est en retrait de 4 espaces : ce retrait (cette indentation) signifie que l'instruction print appartient à la boucle. Le retour à *la marge* comme dans l'exemple suivant avec l'instruction a=5 marque la fin de la boucle, a=5 n'appartient pas à la boucle.

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

Un exemple d'utilisation simple de la boucle est le calcul de la somme des 10 premiers entiers :

```
somme = 0

for i in range(11):
    somme = somme + i
print(somme)
```

Il faut noter que le démarrage d'une boucle, tout comme la définition d'une fonction est suivi de : . Une boucle est initiée par l'instruction de début : $i = 0$ et la condition d'arrêt $i = 10$. Les boucles peuvent être beaucoup plus complexes que celle présentée, et peuvent utiliser d'autres opérateurs. Les opérateurs logiques les plus courants sont :

- for

— if : exprime la condition.
— else : permet de mettre une autre action si la condition if n'est pas validée
— elif : contraction de else + if
— while : permet de continuer une action jusqu'à la vérification d'une condition

Exemples d'instruction if :

```
if n> 0 :
if n== 5 : # n egale une valeur fixe
if n != 5 : # n different de 5
if n >= 5 : # n supperieur ou egal  5
if n <= 5 : # n inferieur ou egal  5
```

Exemple : ✓Générer une boucle pour trouver les 15 premiers nombres premiers.

Aide : Un nombre premier est un nombre qui n'est divisible que par 1 et par lui-même. Cependant, il est aussi possible de dire qu'un nombre est premier si aucun des nombres avant lui ne le divise. Cette seconde assertion est plus simple à coder !

Il faut générer deux listes avant de créer la boucle. Il faut commencer par tester les diviseurs à partir de 2 car tous les nombres sont divisibles par 1.

Solution : Il existe plusieurs façon de procéder pour générer cette boucle, voici l'une des solutions possible :

```
nb = []
counter = [2]
for i in range(3,16):
    c=0
    for j in counter:
        a = i/j
        if int(a)*j==i:
            c = c+1

    counter.append(i)
    if c==0:
        nb.append(i)
print(nb)
```

En modifiant légèrement ces quelques lignes, nous allons maintenant créer une fonction qui vérifie si un nombre est premier ou non.

2.3.3 Définition de fonctions

Une fonction est définie par la commande def, les arguments de la fonction sont notés entre parenthèses. Il existe deux façons d'enregistrer des arguments :
— sans mot clé (comme arg1 et arg2 de l'exemple ci-dessous). Les arguments doivent alors toujours être donnés dans le même ordre.
— avec mot clé (ex : option1 =) et dans ce cas l'ordre n'est plus important, seul le mot clé compte.

Il est possible, comme dans l'exemple d'utiliser les deux méthodes, à condition que les arguments sans mot clé soient positionnés avant les autres et toujours dans le même ordre.

La spécificité de l'utilisation des mots clés est qu'elle permet de définir des valeurs par défaut de ces arguments. Lors de l'utilisation de la fonction, ces arguments deviennent alors optionnels (sauf si la valeur par défaut doit être changée).

```
def fonction(arg1,arg2, option1=arg3 ):
   """
   Description

   Parameters :
```

```
    Returns :
    """
    bla
```

L'indentation est toujours la clé dans les fonctions : toutes les instructions qui appartiennent à la fonction sont indentées. Une fonction commence généralement par un petit texte qui explique ce qu'elle va faire, donne les paramètres utilisés ainsi que ce que retourne la fonction.

Par exemple, la fonction qui affiche *Hello world* s'écrira de la façon suivante :

```
def hello():
    """
    Print Hello world
    """
     print('Hello world')
```

✓Sans regarder la solution, écrire la fonction qui vérifie si un nombre est premier.

Solution :

```
def nombre_premiers(x):
    """
    Cette fonction donne True si x est premier,
    False sinon

    Parameter :
    ----------     x interger

    Returns :
    -------        True or False
    """
    c = range(2,x)
    b = True
    for i in c:
        a = x/i
        if int(a)*i==x :
            b = False
            print(b)
            break
        else :
            pass
    print(b)
```

Un autre exemple de fonction, plus utile en physique que les nombres premiers est la fonction qui calcule la force exercée par la Terre sur un objet :

```
def earth_gravity_force(m,distance=0):
    """
    Compute the earth gravity force on objects

    Parameters : m mass of the object in kg
    ----------  distance : distance from the surface in meter

    Returns : Force F in Newton
    --------
    """
    M = 5.98*10**(24)
    Rt = 6371000
    G = 6.67*10**(-11)

    d = distance + Rt

    F = G * M*m/d**2
```

```
    print('F=%0.2f N' %F)
```

Bien sûr ces exemples sont non exhaustifs et la création de fonctions sous Python n'a de limite que votre imagination.
Parmi les exemples simples à donner aux élèves on peut trouver :

- Le calcul de la position d'une image grâce aux relations de conjugaison.
- Le calcul des réglages d'un appareil photo automatique
- Une fonction qui fait toute seule les conversions les plus courantes
- Un analyseur de fréquence (spectromètre, fft)
- Le calcul de dilution, d'avancement
- Le calcul de vitesse instantanée, d'accélération
- Le calcul de taille d'instrument pour obtenir une note, où placer les trous sur une flûte
- La Loi de wien et le corps noir
- Une introduction à la physique quantique avec l'effet tunnel et les probabilités
- Le calcul du gradient d'indice (ou saut d'indice) dans une fibre optique.
- La prédiction un angle de réfraction en fonction de la longueur d'onde et de l'indice du milieu
- La résolution du problème de la chute libre et de la vitesse limite.
- Coder en morse

3 Orienté Objet

L'orienté Objet est une façon de coder qui permet de définir des objets qui possèdent des propriétés, des méthodes et des paramètres qu'ils conservent tout au long de l'exécution d'un programme. Ces objets peuvent aussi être transformés ou utilisés en/par un second objet en **héritant** des propriétés du premier. Ceci peut paraître complexe mais il s'agit d'une méthode très utilisée en informatique qui se comprend mieux avec un schéma et qui devient complètement évidente avec l'usage.

Exemple : Je dois programmer pour un industriel qui produit des voitures de toutes marques, des camions et des motos. Je vais d'abord classer ces objets dans trois *class* (ensemble de propriétés) différentes. Prenons la *class* voiture. Elle commence par initialiser les caractéristiques intrinsèques de la voiture que sont le nombre de roues et le fait qu'une voiture possède un moteur dans une fonction init.

```
def class Voiture(marque):
   """ Description de la class"""
   def __init__():
      """
      Parmetres de la class
      """
      self.marque = marque
      self.nb_roues = 4
      self.motor = True
   def type(model=citadine):
       """
       Definit les caracteristiques de la marque
       """
       if model == citadine :
         self.masse = 1000
         self.longueur = 3
         self.largeur = 1.50

    def color(default = gray, paint_ thickness=0.003) :
        """Compute the price and the paint quantity needed
        """
        self.surface = self.longueur*self.largeur
        self.paint_quantity = self.surface * paint_thickness
```

Il existe deux types de paramètres utilisés dans les fonctions des classes :

- les paramètres qui définissent l'objet (qui doivent être conservés) d'une étape à l'autre. Ils seront alors définis par self.paramètre (ex : self.masse).

— les paramètres annexes qui n'ont d'utilité qu'à l'intérieur d'une seule fonction (Ex : $paint_thickness$).

Les fonctions qui suivent le init peuvent soit définir de nouveaux paramètres, soit définir des méthodes qui permettent de modifier l'objet. Ces fonctions peuvent être directement appelée dans le init permettant ainsi d'automatiser à l'appel de la class un certain nombre de méthodes.

Pour utiliser cette class, il suffit de nommer l'objet :

```
v = Voiture('Renaud')
v.type()
v.couleur('rouge')
```

Ainsi l'objet voiture contient tous les paramètres définis par les différentes fonctions. L'utilisation du point entre voiture et la fonction (ex : voiture.type()) indique non seulement l'hérédité mais fait aussi agir la fonction type sur l'objet voiture.

L'utilisation des class sera praticulièrement utile pour interfacer python avec des capteurs et faire de la robotique. Par exemple, dans le cas d'une diode on pourra construire la class diode :

```
def class Diode() :
    """connecte, actionne la diode
    """
    def __init__() :

    def connect():

      """
       Connection python-diode
      """
    def lignt_ON():
      """Allume la diode
      """
    def light_OFF() :
      """Eteint la diode
      """
    def close_conection():
      """
      Ferme la connection entre python et la diode
      """
```

4 Exemple de projets à faire en classe

4.1 Robotique

Commander de petits moteurs et des capteurs de mouvement et d'arrêt pour créer un robot capable de réaliser des actions simples.

— grue
— système pilote automatique pour bateau
— voiture téléguidée qui s'arrête quand elle rencontre un obstacle
— Système émetteur-récepteur de code morse

https://www.gotronic.fr/art-kit-de-40-capteurs-sen-x40-25414.htm
https://www.gotronic.fr/art-module-lopy-1-0-25374.htm#complte_desc
Labjack

4.2 Simulations numériques

Simuler une équation dont la solution n'est pas forcément analytique et répondre à un problème simple.

Exemples :
— écoulement 1D dans un tube
— Propagation d'une onde

4.3 Création de logiciel

Créer de petits logiciel qui permettent de résoudre des problèmes simples vus en classes.
Créer un programme qui traduit le langage (anglais en français) en morse puis envoie des signaux lumineux via une diode. Un photorécepteur récupère l'information et un programme effectue la traduction inverse.

5 Installation de python sous windows

Installer Python, choisir la version

5.1 Installation de librairie

Pip est déjà installé dans python3. Ouvrir un terminal : touche windows +R, taper cmd
Dans le terminal, ouvrir le dossier qui contient python cd path
Puis taper pip install pakage il faut installer numpy, matplotlib, SciPy