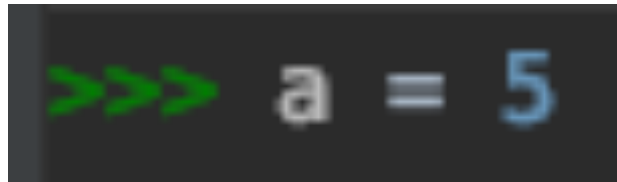


# *LA PROGRAMMATION EN PYTHON*

*Cours écrit et réalisé par Julien LENGLET*

1. Les bases du langage

a. Déclaration de variable



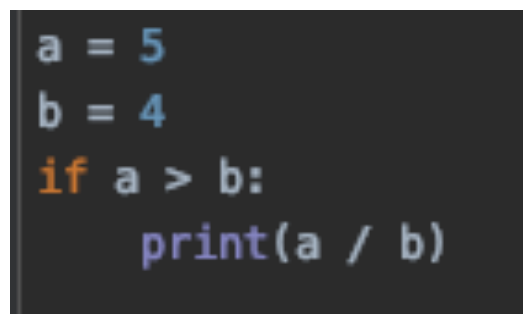
```
>>> a = 5
```

La déclaration d'une variable est très simple en Python. On définit le nom de la variable, ici « a » puis on affecte une valeur avec le signe égal (=), ici la valeur numérique 5.

Une des particularités de Python réside dans le fait de son typage dynamique fort. C'est à dire qu'à la compilation, nous allons laisser le compilateur gérer pour nous le typage des variables, mais qu'il sera aussi capable de nous dire nous avons des erreurs de typages à l'exécution.

b. Syntaxe

La syntaxe en Python est très particulière. Adieu les points virgules en fin d'instruction, adieu les accolades pour définir le contenu d'une condition, d'une boucle ou d'une fonction. Faites place à l'indentation. En Python, tout est question d'indentation. Voyons plutôt avec un exemple.



```
a = 5
b = 4
if a > b:
    print(a / b)
```

Vous voyez qu'ici pour notre if :

- Nous n'avons pas de parenthèses pour définir la condition, elles ne sont pas obligatoires pour des conditions simples.
- Après la condition nous avons un signe « : » qui définit le début du traitement si la condition est OK.
- Puis nous avons notre traitement. Tant que le code est indenté comme ici le print, il appartiendra à cette condition if.

Vous trouvez le code pour les conditions et les boucles en annexe.

De plus, il faut faire attention à la déclaration de nos éléments. Le code Python est lu de haut en bas, c'est à dire que toute fonction, classe, variable qui n'est pas définie AVANT l'utilisation de ces derniers ne sera pas connu et donc nous aurons une erreur lors de la compilation (et même avant dans l'IDE)

## c. Les entrées et sorties

Comme vous avez pu le voir dans l'exemple précédent, nous avons utilisé la fonction `print`. Cette fonction permet d'afficher dans la console des données.

Si on souhaite récupérer une valeur entrée par l'utilisateur, il faudra utiliser la fonction `input`. Voici un exemple pour récupérer une donnée tapée par l'utilisateur :

```
userinput = input("Veuillez entrer un nombre\n")
print(userinput)
```

Ici, on récupère dans la variable `userinput`, la valeur entrée au clavier.

Nous pouvons aussi forcer l'utilisateur par exemple à renseigner un entier.

```
userinput = int(input("Veuillez entrer un nombre\n"))
print(userinput)
```

Comme Python est un langage fortement typé, si l'utilisateur entre une valeur autre qu'un entier, le programme lancera une exception à l'exécution.

## d. Définition d'une fonction

```
def maprocedure():
    print("C'est une belle procedure")

def mafonction(arg1, arg2, arg3):
    return arg1 + arg2 + arg3
```

Comme pour les variables, les fonctions n'ont pas de type. On utilise la même méthode d'indentation que pour les conditions et les boucles.

Pour appeler les fonctions :

```
maprocedure()
print(mafonction(1, 2, 3)) #3 Arguments positionnels
print(mafonction(1, "4", 4)) #Plantera le programme (+ sur un int et un str)
print(mafonction(1, arg3=5, arg2=10)) # 1 Argument positionnel + 2 arguments mot-clés
print(mafonction(arg3=14, 2, arg1=23)) #On ne peut pas mettre d'argument positionnel après un
#argument mot-clé
```

On peut appeler une fonction classique dans l'ordre de ces arguments, on parle alors d'arguments positionnels. Mais aussi on peut faire appel aux arguments dans n'importe quel ordre avec le nom qui leur est associé. Attention, on ne peut pas utiliser d'argument positionnel après la définition d'un argument par mot-clé.

## e. Les Exceptions

Même si du code est syntaxiquement correct, il peut générer une erreur lors de son exécution. On appelle cela des exceptions. Les exceptions peuvent être gérées comme ceci :

```
try:
    x = 5 / 0
except ArithmeticError:
    print("Attention !! Division par zero")

try:
    x = int(input("Veuillez entrer un nombre"))
except ValueError:
    print("Valeur invalide, ceci n'est pas un nombre")
```

Il existe un grand nombre d'exceptions, `ArithmeticError` et `ValueError` sont quelques exemples de ces exceptions.

On peut aussi rajouter un bloc « finally » après le bloc « except ». Ce bloc est exécuté avant de quitter le bloc « try » qu'il y ait eu une exception ou non.

## f. Lecture / Écriture dans un fichier texte

```
#Simple ecriture
fichier = open("fichier.txt", "w")
fichier.write("Hello World")
fichier.close()

#Boucle d'écriture
fichier = open("fichier.txt", "w")
for i in range(5):
    fichier.write("Hello World\n")
fichier.close()

#Lecture tout le fichier
fichier = open("fichier.txt", "r")
print(fichier.read())
fichier.close()

#lecture premiere ligne
fichier = open("fichier.txt", "r")
print(fichier.readline())
fichier.close()
```

Voici les arguments possibles après le nom du fichier :

**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

Il ne faut surtout pas oublier de fermer votre connexion au fichier après utilisation !

#### g. Exercices

**Exo 1** : Demandez deux entrées claviers qui seront forcément des nombres puis afficher la moyenne

**Exo 2** : Demandez des entrées au clavier tant que l'utilisateur n'a pas entré une certaine chaîne de caractères spécifiée. Affichez toutes les entrées séparées d'un espace.

**Exo 3** : Créez une fonction qui aura 3 paramètres et qui permettra de renvoyer la valeur de la moyenne des 3 paramètres. La fonction sera appelée avec des arguments positionnels. Créez une autre fonction qui aura pour but de faire la médiane de 3 paramètres également. Écrire un programme qui demande à l'utilisateur 3 valeurs numériques et ensuite afficher la moyenne et la médiane.

**Exo 4** : Écrivez un programme qui lit le fichier ficEntree.txt et qui compte le nombre de caractères 'e' dans ce fichier, puis affiche le résultat à l'écran.

**Exo 5** : Écrivez un programme qui permet de récupérer des données saisies par l'utilisateur et les enregistrer dans un fichier texte. Ensuite copier le fichier texte dans un autre fichier texte en écrivant chaque mot à l'envers.

## 2. La programmation orientée objet en Python

### a. Déclaration d'une classe et d'un objet

Pour déclarer une classe il suffit d'utiliser le mot clé `class` suivi du nom de la classe qui par convention commencera toujours par une majuscule.

Pour déclarer votre objet, il suffira d'appeler le constructeur. Voici une déclaration de classe ainsi qu'une instantiation d'objet de la classe :

```
class Voiture:
    pass

voiture = Voiture()
```

Nous avons ici une classe qui se nomme `Voiture` et nous avons construit un objet de la classe `Voiture` qu'on contient dans la variable `voiture`.

Note : Le mot clé « `pass` » signale que la classe est vide.

### b. Constructeurs et attributs

Nous allons maintenant définir le constructeur de notre classe `Voiture`. Nous souhaiterons ajouter une marque à notre voiture et pour cela nous allons définir un attribut qui s'appellera `marque`. Et là c'est maintenant que nous allons voir les différences avec les langages orienté objet comme `JAVA` ou `C#`.

```
class Voiture:
    #Définition du constructeur
    def __init__(self, marque):
        #Déclaration des attributs
        self.marque = marque
        self.roues = 4
```

Les attributs sont déclarés dans le constructeur directement avec le mot clé `self` avant ! `self` représente l'objet en lui-même. La fonction à définir pour le constructeur est donc `__init__`. Elle prend en paramètre `self` puis un nombre de paramètres en fonction de la classe bien sûr.

Il ne peut y avoir qu'un seul et unique `__init__` par classe.

Ce n'est pas tout avec les attributs puisqu'on peut déclarer des attributs à la volée sur un objet spécifique. En voici un exemple :

```
voiture1 = Voiture("Ferrari")
voiture2 = Voiture("4L")
voiture2.estPourrie = True

print(voiture2.estPourrie)
print(voiture1.estPourrie)
```

On a utilisé la classe Voiture avec le constructeur défini plus haut. Puis on a ajouté directement l'attribut « estPourrie » à l'objet voiture2 !

L'attribut est alors connu de voiture2, mais en aucun cas de voiture1.

### c. L'encapsulation

En Python, il n'y a pas de mot clé (comme private en Java/C#) pour gérer la protection des attributs. Pour déclarer un attribut comme privé, il faut alors rajouter deux fois le symbole underscore avant le nom de notre attribut.

```
def __init__(self, marque):
    #Déclaration des attributs
    self.__marque = marque #Attribut privé
    self.roues = 4
```

Pour pouvoir accéder à cette variable, nous devons alors déclarer des accesseurs et des mutateurs, plus communément appelés Getters et Setters. Il s'agit tout simplement de fonction qui donneront accès aux membres privés d'une classe. On les trouve sous la forme suivante :

- get\_XXXX() pour les Getters
- set\_XXXX() pour les Setters

```
def get_marque(self):
    return self.__marque

def set_marque(self, marque):
    self.__marque = marque
```

## d. Attributs de classe

Les attributs de classe sont des variables qui seront communes à tous les objets qui vont être instancié via cette classe.

```
immatriculation = 0

# Définition du constructeur
def __init__(self, marque):
    # Déclaration des attributs
    self.__marque = marque # Attribut privé
    self.roues = 4
    Voiture.immatriculation += 1
```

Les attributs de classe sont déclarés directement dans la classe et non pas dans le constructeur. Ici, nous avons rajouté l'attribut de classe immatriculation qui permettra de savoir combien de véhicule sont en circulation.

```
voiture1 = Voiture("Ferrari")
print(voiture1.immatriculation)
voiture2 = Voiture("4L")
print(voiture2.immatriculation)
voiture3 = Voiture("CR-Z")
print(voiture3.immatriculation)
```

Ici, nous aurons bien les immatriculation 1, 2 et 3 respectivement pour les voitures 1, 2 et 3.

NOTE : Attention, la modification d'un attribut de classe peut se faire en dehors d'un objet du type qui contient l'attribut de classe !

## e. Héritage Simple

Pour faire hériter une classe en Python, il suffit de rajouter le nom de la classe mère entre parenthèses après le nom de la classe que l'on souhaite faire hériter.

```
class Voiture(Vehicule):
```

Ici, notre classe Voiture hérite de la Classe Vehicule. Nous allons voir un peu le comportement en Python en refactorant notre classe Voiture afin qu'elle hérite de Vehicule.



```

class Vehicule:
    immatriculation = 0

    def __init__(self, marque):
        self.__marque = marque

    def get_marque(self):
        return self.__marque

class Voiture(Vehicule):
    def __init__(self, marque):
        super().__init__(marque)
        self.roues = 4
        Vehicule.immatriculation += 1

```

Nous avons sorti la variable de classe de Vehicule ainsi que l'attribut marque que nous avons gardé en privé dans la classe Vehicule. Dans le constructeur de notre classe Voiture, nous sommes alors obligés d'appeler le constructeur de la classe mère par le biais de super().

Ensuite, nous allons instancier 3 objets de voiture comme tout à l'heure.

```

voiture1 = Voiture("Ferrari")
print("Marque : {0} Immatriculation : {1}".format(voiture1.get_marque(), voiture1.immatriculation))
voiture2 = Voiture("4L")
print("Marque : {0} Immatriculation : {1}".format(voiture1.get_marque(), voiture1.immatriculation))
voiture3 = Voiture("CR-Z")
print("Marque : {0} Immatriculation : {1}".format(voiture3.get_marque(), voiture3.immatriculation))

```

## f. Les méthodes spéciales

Catégorie	Nom des méthodes spéciales
Représentation	<code>__repr__</code> , <code>__str__</code> , <code>__format__</code> , <code>__bytes__</code>
Conversion en nombre	<code>__abs__</code> , <code>__bool__</code> , <code>__complex__</code> , <code>__int__</code> , <code>__float__</code> , <code>__hash__</code> , <code>__index__</code>
Collections	<code>__len__</code> , <code>__getitem__</code> , <code>__setitem__</code> , <code>__delitem__</code> , <code>__contains__</code>
Itérateurs	<code>__iter__</code> , <code>__reversed__</code> , <code>__next__</code>
Création et destruction d'instances	<code>__new__</code> , <code>__init__</code> , <code>__del__</code>
Gestion des attributs	<code>__getattr__</code> , <code>__getattribute__</code> , <code>__setattr__</code> , <code>__delattr__</code> , <code>__dir__</code>
Comparaison	<code>__lt__</code> <, <code>__le__</code> <=, <code>__eq__</code> ==, <code>__ne__</code> !=, <code>__gt__</code> >=, <code>__ge__</code> >=
Opérateurs arithmétiques	<code>__add__</code> +, <code>__sub__</code> -, <code>__mul__</code> *, <code>__truediv__</code> /, <code>__floordiv__</code> //, <code>__mod__</code> %, <code>__round__</code>

Toutes ces méthodes peuvent être redéfinies dans une classe afin de définir le comportement de ces dernières.

Exemple :

```
print(repr(voiture1))
```

affiche ceci par défaut :

```
<__main__.Voiture object at 0x100a4c250>
```

Après redéfinition de la méthode spéciale repr dans la classe, nous aurons ceci :

```
def __repr__(self):
    return "Cette {0} a l'immatriculation {1}".format(self.get_marque(), self.immatriculation)
```

Et affichera : Cette Ferrari a l'immatriculation 1

3. Les interfaces graphiques avec Tkinter
  - a. Création d'une fenêtre

```
from tkinter import *  
root = Tk()  
  
root.mainloop()
```

- b. Les Widgets Tkinter

Pour instancier un widget nous allons de voir définir l'endroit où on veut qu'il soit affiché mais aussi différentes options. Voici la liste des Widgets :

- Button : Pour créer des boutons
- Canvas : Pour dessiner des formes, comme des carrés et des rectangles
- Checkbutton : Pour pouvoir afficher des boîtes de choix
- Entry : Pour récupérer les entrées utilisateur
- Frame : Pour créer un conteneur pour d'autres widgets
- Label : Pour afficher du texte sur une seule ligne, peut aussi contenir des images
- Listbox : Pour afficher une liste de choix à l'utilisateur
- Menubutton : Pour afficher un menu déroulant
- Menu : Pour afficher un menu en haut de l'application (Fichier, Edition, etc ...)
- Message : Pour afficher plusieurs lignes de texte
- Etc ..

Par exemple, voici la déclaration pour un bouton :

```
w = Button ( master, option = value, ... )
```

Pour plus d'info : [https://www.tutorialspoint.com/python3/python\\_gui\\_programming.htm](https://www.tutorialspoint.com/python3/python_gui_programming.htm)

Pour placer un widget, on utilisera ensuite une des 3 méthodes suivantes :

- La méthode pack(): On organise les widgets en blocks avant de les placer dans le parent

[https://www.tutorialspoint.com/python3/tk\\_pack.htm](https://www.tutorialspoint.com/python3/tk_pack.htm)

- La méthode grid() : On organise les widgets comme un tableau

[https://www.tutorialspoint.com/python3/tk\\_grid.htm](https://www.tutorialspoint.com/python3/tk_grid.htm)

- La méthode place() : On organise les widgets comme on le souhaite et on force les positions

[https://www.tutorialspoint.com/python3/python\\_gui\\_programming.htm](https://www.tutorialspoint.com/python3/python_gui_programming.htm)

Annexe :

## Les conditions et les boucles

```
#Condition
a = 1
b = 2
if a > b:
    print("OK")
    #Moncode
    #Moncode
elif b > a:
    print("NOK")
else:
    print("Abort")

#Boucle While
boucle = 0
while boucle < 10:
    boucle += 1
    #Moncode

#Boucle For it rative
liste = [1,2,3]
for i in liste:
    print(i)

#Boucle For (While-like)
for i in range(5):
    liste.append(i)
```