

# **Programmation en Python**

# 1. Pourquoi le langage Python ?

- **Facile à apprendre** avec des lignes de commande claires et concises
- Sa portée est **générale** et multiplateforme
- Utilisé en *open source*, pris en charge par une fondation et soutenu par une large **communauté**.
- Applications dans de divers et nombreux **domaines** (web, data science, intelligence artificielle, recherche ...)
- Utilisé dans l'**enseignement supérieur**.

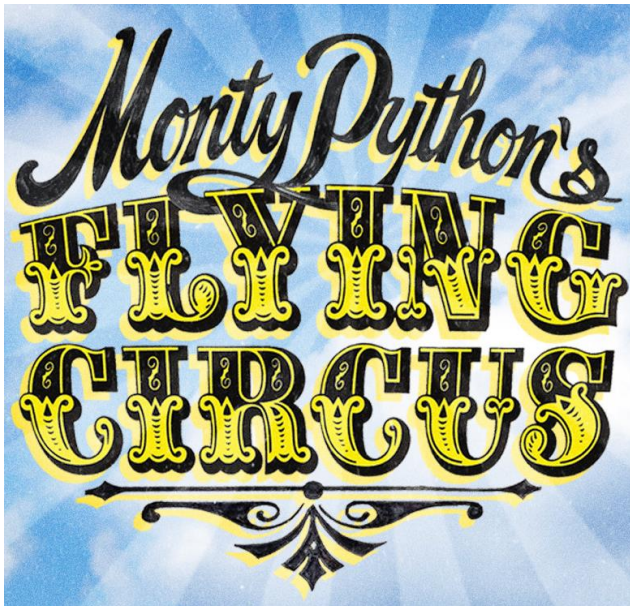
# Guido van Rossum

Mathématicien Néerlandais (1956)

Créateur du langage Python en **1991**

2002 : Prix du logiciel libre

A travaillé pour **Google** et **Dropbox**



## Monty Python's Flying Circus

Série TV humoristique britannique (1969-1974)

qui inspiré le nom du langage à GvR

## 2. Les éléments de base

- Variables et affectation

### Variable :

Une variable sert à **stocker** une donnée, on peut penser à une boîte sur laquelle est écrit un nom et dans laquelle on place des informations diverses. Le nom peut être n'importe quelle chaîne alphanumérique, exceptés certains mots réservés, sans commencer par un chiffre.

$a = 2$

La variable **a** prend la valeur **2**

$p = 3$

La variable **p** prend la valeur **3**

$c = a ** p$

La variable **c** prend la valeur **a<sup>p</sup>** soit donc **2<sup>3</sup>**

$2 + 3 = a$

Ceci n'a aucun sens et renverra une erreur

$a = 2 + 3$

OK, c'est une instruction d'affectation

## Expression :

Une expression a une **valeur** qui est le **résultat** d'une combinaison de variables ou d'objets, de constantes et d'opérateurs.

## Instruction :

Une instruction est une **commande** qui doit être exécutée par la machine.

$0.5 * x^{*2} + 1$

est une expression

45

est une expression

a = 2

est une instruction

## Affectation :

Une affectation est une instruction qui commande à la machine de créer une variable en lui précisant son nom et sa valeur.

$a = 2 + 3$  est une instruction d'affectation

$x = 0.5 * x ** 2 + 1$  est une instruction d'affectation

Mettre la valeur de l'expression  $0.5 * x ** 2 + 1$  dans la variable qui a pour nom  $x$ .

Remarque :  $a, b = 1, 2$  équivaut à  $a = 1$  et  $b = 2$

- **Types simples**

Le type d'une variable définit l'ensemble des **valeurs** qui peuvent lui être affectées ainsi que les **opérations** et les **fonctions** utilisables.

**int** : Les nombres entiers

*Valeurs possibles :*    1    24 789    -12    -64565373567857

**float** : Les nombres réels

Le point remplace la virgule utilisée en mathématiques en France.

*Valeurs/expressions possibles :* 1.5    2.6    -10.5    1 . 5+2 . 6 / -1 0 . 5

**bool** : Les valeurs booléennes **True** (vrai) et **False** (faux)

Ces valeurs peuvent être considérées comme les entiers 1 pour `True` et 0 pour `False`.

*Expressions possibles :*    `0 == 0`      `8 + 1 == 2 * 3`      `a >= 14`

**str** : Les chaînes de caractères

*Valeurs possibles :*    `'Bonjour'`      `'la valeur de a est'`      `'234'`

Pour affecter la variable de type `str` on écrira :

`ch = 'bonjour' ou ch = "bonjour"`

Pour connaître le type d'une variable, on peut utiliser la fonction "type":

`type (ch)`      renvoie    `<class 'str'>`



## ● Opérations sur les types numériques

Priorité basse

**+**   **-**

addition

soustraction

Priorité moyenne

**\***   **/**   **//**   **%**

multiplication

division

quotient (division euclidienne)

reste (modulo)

Priorité haute

**\*\***

puissance

$$10 \ // \ 3 = 3 \quad , \quad 10 \ \% \ 3 = 1$$

$$a - b + c * d ** e = a - b + (c * (d ** e))$$

$$5 - 8 + 4 * 2 ** 3 = 5 - 8 + (4 * (2 ** 3)) = -3 + (4 * 8) = -3 + 32 = 29$$

**PEMDAS** = **P**arenthèse **E**xposant **M**ultiplication **D**ivision **A**ddition **S**oustraction

## ● Comparaisons et opérateurs logiques

`==` égal      `!=` différent      `>` supérieur      `<=` inférieur ou égal

`or` ou      `and` et      `not` non

Si `a = 2` , `b = 3` , `c = True` **et** `d = False` alors

`a == 2`      renvoie `True`      `b <= 3`      renvoie `True`

`a == 3`      renvoie `False`      `a !> 2`      renvoie une erreur

`a == b`      renvoie `False`      `not c`      renvoie `False`

`b != 3`      renvoie `False`      `c and d`      renvoie `False`

`a > 2`      renvoie `False`      `c or d`      renvoie `True`

## ● Opérations sur le type str

Affectation : `ch = 'bonjour'` ou `ch = "bonjour"`

- Attention, si on écrit `ch=bonjour` alors `bonjour` est considéré comme le nom d'une variable et une erreur sera signalée si cette variable n'existe pas.
- La fonction **len** renvoie le nombre de caractères contenus dans la chaîne passée en paramètre : `len("bonjour")` ou `len(ch)` renvoie 7
- Les indices de position des caractères dans une chaîne commencent à 0 :

`ch[1]` renvoie `'o'`

`ch[0]` renvoie `'b'`

`ch[7]` renvoie une erreur

`ch[1:3]` renvoie `'on'`

- Opérations sur le type str, si `a = 'bon'` et `b = 'jour'` :

`a + b` renvoie `'bonjour'`

`2 * a` renvoie `'bonbon'`

- **Conversions de type**

Il existe plusieurs fonctions qui permettent de forcer le type d'une variable en un autre type pour permettre de la comparer ou effectuer des opérations.

`bool()` convertit en booléen : `0`, `""` ou `None` donnent `False` et le reste `True`.

`int()` permet de modifier une variable en entier (supprimera la partie décimale) : `int(390.8)` donnera `390`

`float(57)` convertira `57` en `57.0`

`str()` converti en chaîne de caractères `str(12)` donnera `'12'`

- **Exemples :**

a = 34.3                    type(a) donnera <class 'float'>

b = 23                    type(b) donnera <class 'int'>

c = str(a)                type(c) donnera <class 'str'>

d = float(b)            type(d) donnera <class 'float'>

e = 'Joueur 1' type(e) donnera <class 'str'>

```
print("Bravo " + e + " vous avez " + a + " points!")
```

Renverra une erreur :

Traceback (most recent call last):

File "<pyshell#21>", line 1, in <module>

```
    print("Bravo " + e + " vous avez " + a + " points!")
```

TypeError: can only concatenate str (not "float") to str

On corrigera avec :

```
print("Bravo " + e + " vous avez " + str(a) + " points!")
```

Qui renverra :

Bravo Joueur 1 vous avez 34.3 points!

- **Complément sur les opérations**

Le symbole `//` renvoie le quotient de la division euclidienne :

`4 // 2` renvoie 2

`4 // 3` renvoie 1

`4 // 5` renvoie 0

Le symbole `%` renvoie le reste de la division euclidienne (modulo) :

`10 % 5` renvoie 0

`10 % 3` renvoie 1

- **Complément sur les affectations**

`a , b = 1,2` équivaut à `a = 1` et `b = 2`

`x += y` équivaut à `x = x + y`

- **L'indentation**

**L'indentation**, décalage vers la droite du début de ligne, est un élément de syntaxe important en Python. Elle délimite des blocs de code et elle aide à la lisibilité en permettant d'identifier facilement ces blocs. La ligne précédant l'indentation se termine par le signe “:” (deux-points).



### 3. Instructions conditionnelles et boucles

- Instructions conditionnelles

La structure la plus simple est le “if” :

```
if condition :  
    instructions
```

Le mot `condition` désigne une expression et le mot `instructions` désigne une instruction ou un bloc d'instructions écrites sur plusieurs lignes à effectuer lorsque la condition est vraie.

```
if n % 2 == 0 :  
    n = n // 2
```

La structure “**if-else**” ajoute une alternative pour exécuter une autre instruction ou bloc d'instructions à effectuer lorsque la condition est fausse :

```
if condition :  
    instructions1  
else :  
    instructions2
```

Par exemple :

```
if n % 2 == 0 :  
    n = n // 2  
else :  
    n = 3 * n + 1
```

La structure “**if-elif-else**” ajoute plusieurs alternatives conditionnelles :

```
if condition1 :  
    instructions1  
elif condition2 :  
    instructions2  
elif condition3 :  
    instructions3  
...  
else :  
    instructions4
```

Par exemple :

```
if n % 4 == 0 :
```

```
    n = n // 4
```

```
elif n % 4 == 1 :
```

```
    n = ( 3 * n + 1 ) // 4
```

```
elif n % 4 == 2 :
```

```
    n = n // 2
```

```
else :
```

```
    n = ( 3 * n + 1 ) // 2
```

## Remarques :

- Les mots `if` et `elif` sont toujours suivis par une expression qui prendra la valeur `True` ou `False`, la ligne se termine par le symbole “:” (deux points)
- Le mot `else` est toujours immédiatement suivi par le symbole “:” (deux points)
- C’est l’indentation qui permet de délimiter les blocs d’instructions à exécuter lorsque les conditions sont vérifiées

## Note :

Les mots `True`, `False` et `None` sont les rares mots en Python dont l’écriture commence par une lettre majuscule.

- **Boucles conditionnelles**

La structure est la suivante :

```
while condition :  
    instructions
```

La structure est identique à celle du “`if`”. Le bloc d’instructions indenté qui suit est exécuté tant que la condition est vérifiée. C’est ce bloc d’instructions qui doit modifier la valeur de la condition pour qu’elle ne soit plus vérifiée et que le programme sorte du bloc et continue son exécution.

```
while a >= b :  
    a = a - b  
    n = n + 1
```

- **Boucles non conditionnelles**

Une boucle non conditionnelle sera répétée  $n$  fois,  $n$  étant connu à l'avance, la structure est la suivante :

```
for i in range(n) :  
    instructions
```

Ce code est équivalent à :

```
i = 0  
  
while i < n :  
    instructions  
  
    i = i + 1
```

Dans les deux cas, une variable `i` est créée et prendra successivement les valeurs  $0, 1, 2, \dots, n-1$  pour la boucle `for` et  $0, 1, 2, \dots, n$  pour la boucle `while` et les instructions seront exécutées  $n$  fois.

La boucle “for” en Python propose d’autres possibilités, si on considère objet comme une séquence de plusieurs éléments (chaine de caractères, liste...) on pourra utiliser :

```
for element in objet :  
    instructions
```

Exemple :

```
for car in "Bonjour" :  
    print(3 * car)
```



Lorsqu'on utilise la fonction range avec des entiers :

`for i in range(f) :` i prendra la valeur des entiers successifs de 0 inclus à f exclu si  $f > 0$

`for i in range(d, f) :` i prendra la valeur des entiers successifs de d inclus à f exclu si  $f > d$

`for i in range(d, f, p) :` i prendra la valeur des de d inclus à f exclu avec un pas de p.

Attention, erreur classique :

```
for x in liste :  
    x = 2*x
```

Ce code ne modifiera pas la liste existante.

## Remarques :

L'instruction `break` permet d'interrompre une boucle

L'instruction `continue` permet d'éviter un passage dans la boucle

```
for i in range(10):  
    if i == 3 :  
        continue  
    if i == 8 :  
        break  
    print(i)
```

Renverra : 0, 1, 2, 4, 5, 6, 7

## 4. Les fonctions

- **Définition d'une fonction**

Une fonction se définit de la manière suivante :

```
def nomDeLaFonction(argument1, argument2) :  
    """ aide sur la fonction : docstring  
    (facultatif) """  
    corpsDeLaFonction
```

`def` est un mot clé du langage Python, les arguments sont séparés de virgules.

Le corps de la fonction est un bloc de code indenté.

Si le corps de la fonction contient l'instruction `return`, alors l'appel de la fonction est une **expression** qui a donc une valeur.

S'il n'y a pas d'instruction `return` dans le corps de la fonction, alors l'appel de la fonction renverra la valeur `"None"`. Ce type de fonction s'appelle une **procédure**.

Exemple :

```
def volume(longueur, largeur, hauteur):  
    """ renvoie le volume d'un parallélépipède  
    dont on fournit les trois dimensions en  
    arguments. """  
    return longueur * largeur * hauteur
```

- **Espace et portée des variables**

**L'espace local** d'une fonction contient les paramètres qui lui sont passés, et les variables définies au sein de celle-ci.

Pour qu'une variable soit utilisée dans une fonction, il faut que la variable appartienne à son espace local ou à l'espace qui appelle la fonction.

Une fonction ne peut pas modifier, par affectation, la valeur d'une variable extérieure à son espace local.

```
x=3
def f(x):
    x+=2
    print(x)

f(x) # affiche 5
print(x) # affiche 3
```

La variable `x` utilisée dans la fonction est distincte de la variable `x` définie au début du programme (`x=3`) et n'existe plus après l'appel de la fonction.

Après l'instruction `f(x)`, l'espace local de la fonction `f` est détruit.

Il existe un moyen de modifier avec une fonction des variables extérieures à celle-ci. On utilise pour cela des variables globales avec le mot-clé **global**.

Le code suivant permet de modifier la variable x extérieure à la fonction :

```
x=3
def f():
    global x
    x+=2
    print(x)

f() # affiche 5
print(x) # affiche 5
```

**Ce procédé est à éviter car il nuit à la portabilité d'une fonction.**