

Licence Creative Commons    
MAJ: 18 juillet 2020

## Une « piquêre » de Python



Suite à la réforme du collège de 2016 et à la part importante de la partie informatique dans notre vie, vous avez découvert le logiciel de programmation scratch (l'an dernier). Ce logiciel libre a été conçu pour initier les élèves dès l'âge de 8 ans à des concepts fondamentaux en mathématiques et en informatique. Cela explique que beaucoup d'élèves de primaire commencent déjà le codage informatique à l'école primaire. Néanmoins, Scratch présente plusieurs défauts majeurs et va se montrer insuffisant au Lycée.

## 1 1 Pourquoi utiliser Python au Lycée ?

Le choix de Python est assez naturel pour différentes raisons :

- ★ **un langage simple et mathématiquement puissant** pour lequel on trouve des bibliothèques mathématiques très évoluées dans de multiples domaines (calcul numérique, tracé de courbes ...)
- ★ Le langage Python est **libre et gratuit**, et peut donc être installé dans les établissements ou chez vous.
- ★ Enfin, ce langage fait partie désormais dans les programmes des classes préparatoires aux grande écoles.

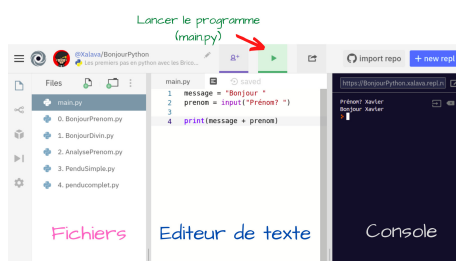
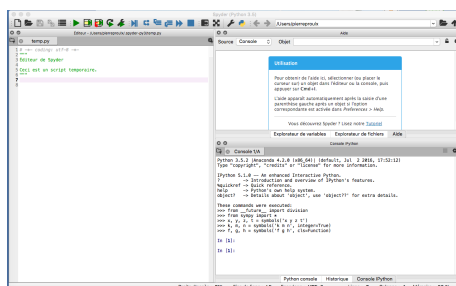
## 1 2 Pourquoi un tel nom ?



Le langage Python est un langage de programmation apparu vers 1990. Développé au départ par Guido van Rossum, un développeur néerlandais, il porte son nom en référence à la troupe d'humoristes britanniques des années 80 des Monty Python.

## 1 3 Spyder ou repl.it

Il existe plusieurs logiciels qui utilisent le langage Python. Cette année, nous utiliserons Spyder d'Anaconda en classe et vous pourrez utiliser, à la maison ou au lycée, l'IDE en ligne **repl** nécessitant une connexion internet.



### 2 1 La gestion des espaces

Dans le langage Python, on peut passer des lignes pour plus de clarté, ce qui n'est pas pris en compte lors de l'exécution du programme. Par contre, vous ne pouvez pas ajouter un espace en début de ligne comme bon vous semble, car cela a une signification.

On appelle **indentation** ce décalage d'un cran d'une ou de plusieurs lignes d'un programme. Elle permet de délimiter un bloc d'instructions dans une boucle ou lors d'une exécution conditionnelle. La ligne précédant l'indentation se finit toujours par deux points. Quand vous appuyez sur la touche après avoir tapé « : », l'**indentation** est automatiquement effectuée en même temps que le passage à la ligne.

### 2 2 Notion de variable

Dans un programme, une **variable** est repérée par son nom et possède une valeur qui évolue au cours de l'exécution du programme. Une variable peut être de type :

- **nombre entier** ;
- **nombre flottant**, c'est-à-dire nombre à virgule ;
- **chaîne de caractères**, sa valeur est alors une suite ordonnée de caractères ;
- **liste**, c'est-à-dire une suite ordonnée d'objets du langage comme par exemple `M=["a","e","i","o","u","y"]` ;
- **booléen**, elle n'a que deux valeurs possibles : **True**(Vrai) et **False**(Faux). Par exemple, `a < 5` est un booléen qui a la valeur `True` si `a` est strictement inférieur à 5 ou `False` sinon.

### 2 3 L'affectation

L'instruction d'affectation permet d'attribuer une valeur à une variable. Dans l'exemple suivant, la variable `X` prend la valeur 2 :

Algorithme	Langage Python
$X \leftarrow 2$	<code>X = 2</code>

### 2 4 Les instructions d'entrée-sortie

Les instructions d'entrée-sortie permettent de saisir en entrée et d'afficher en sortie les valeurs des variables :

Algorithme	Langage Python
Saisir <code>X</code>	<code>X = float(input())</code>
Afficher <code>X</code>	<code>print(X)</code>

Dans le langage Python, l'instruction d'entrée précise le **type** de la variable : `int` (nombre entier), `float` (nombre à virgule) ou chaîne de caractères lorsque rien n'est précisé.

## 3

## Définir une fonction

Une fonction est un **bloc d'instructions** qui a reçu un nom, dont le fonctionnement dépend d'un certain nombre de paramètres (les **arguments** de la fonction) et qui renvoie un résultat (au moyen de l'instruction **retourne**).

```
def ma_fonction():  
    instruction_1  
    instruction_2  
    ...  
    return
```

Exemple d'une fonction possédant un seul argument  $a$  :

```
1 def calcule_cube(a):  
2     cube = a * a * a      # ou bien a**3  
3     return cube
```

## 4

## Les instructions conditionnelles

### 4 1 Condition

Une **condition** est un énoncé qui peut être vrai ou faux. Par exemple,  $a < b$  est une condition vraie ou fausse suivant les valeurs de  $a$  et  $b$ .


### 4 2 Comparateurs

- Le symbole `==` désigne **l'égalité dans un test**.
- Les symboles `>` et `<` désignent les inégalités strictes habituelles.
- Les symboles `<=` et `=>` désignent les inégalités  $\leq$  et  $\geq$  habituelles.
- Le symbole `!=` signifie « différent de ».

### 4 3 Si...alors

**if test :**

effectue (une fois) les **instructions indentées** qui suivent lorsque le test est vérifié.

 Le « alors » n'apparaît pas en Python, c'est l'indentation qui délimite le bloc à exécuter.

On peut imbriquer les boucles, à la manière d'un arbre qui se dessinerait avec les indentations.

```
ifcondition:
    instruction_1
    instruction_2
    ...
instructions suivantes
```

Diagram illustrating the syntax of an if statement:

- mot réservé "if"
- une condition
- deux points
- bloc d'instructions indenté sera exécuté uniquement si la condition est vérifiée
- suite du programme

Voici un exemple :

```
1 if vitesse > 110:
2     print("Attention, tu roules trop vite.")
```

#### 4 4 Si...alors...sinon ...

**if test : ... else : ...** effectue les instructions indentées lorsque le test est vérifié, sinon effectue les instructions alternatives indentées.

```
ifcondition:
    instruction
    instruction
    ...
else:
    instruction
    ...
instructions suivantes
```

Diagram illustrating the syntax of an if-else statement:

- bloc exécuté si la condition est vérifiée
- bloc exécuté si la condition n'est pas vérifiée

Voici un exemple :

```
1 if x >= 0:
2     print("Le nombre est positif.")
3 else:
4     print("Le nombre est négatif.")
```



Le « else » est **aligné** avec le « if » qui lui correspond.

Taper « : » puis appuyer sur la touche « entrée » pour passer à la ligne, provoque l'indentation automatique.

### 5 1 Répéter $n$ fois

Une **boucle** permet de répéter plusieurs fois de suite un même traitement. Lorsque le nombre  $n$  d'itérations est connu à l'avance, on parle de **boucle bornée**.

### 5 2 Compteur

Afin de compter le nombre d'itérations, on utilise un compteur initialisé par exemple à 1 qui s'incrémente automatiquement de 1 à chaque itération jusqu'à la valeur  $n$ .

#### Algorithme

```
Pour  $i$  allant de 1 à  $n$ 
  | Traitement
Fin Pour.
```

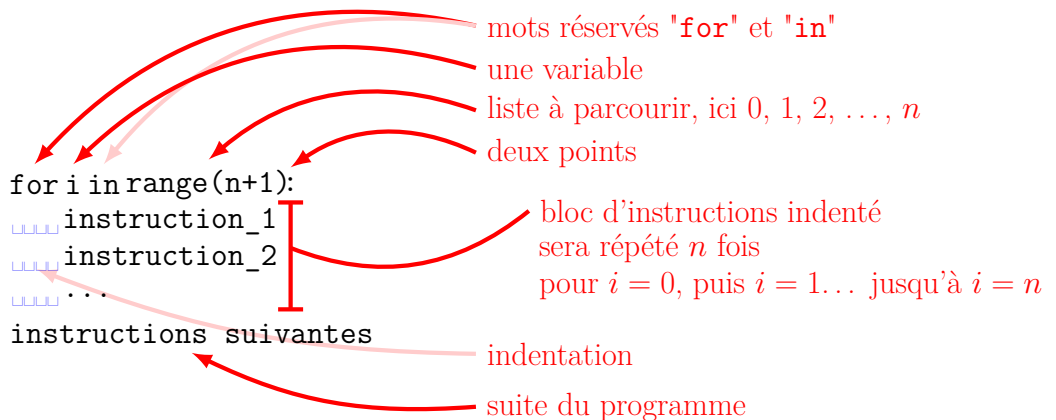
#### Langage Python

```
for  $i$  in range(1,  $n + 1$ ) :
...
...
```

Voici un exemple :

```
1  def nbre_a(phrase):
2      nbre_a=0
3      for lettre in phrase:
4          if lettre=="a":
5              nbre_a=nbre_a+1
6      return nbre_a
```

Qu'obtient-on dans la console en écrivant `>>> nbre_a("j'aime les maths")` ?



**6 1 Répéter tant que...**

Dans une boucle, le nombre d'itérations peut dépendre d'une condition. Le traitement est alors répété tant que la condition est vraie. Lorsque la condition est fausse, on sort de la boucle et la suite des instructions des programmes s'applique.

**6 2 Nombre d'itérations**

Le nombre d'itérations n'est donc en général pas prévu à l'avance et on parle de **boucle non bornée**.

**Algorithme**

```
Tant que condition :
    | traitement
Fin de Tant que.
```

**Langage Python**

```
While ...
...
...
```

```
whilecondition:
    instruction_1
    instruction_2
    ...
instructions suivantes
```

mot réservé "while"  
 une condition  
 deux points  
 bloc d'instructions indenté  
 sera exécuté tant que  
 la condition est vérifiée  
 suite du programme

Voici un exemple :

Ce bout de code cherche la première puissance de 2 plus grande qu'un entier  $n$  donné. La boucle fait prendre à  $p$  les valeurs 2, 4, 8, 16, ... Elle s'arrête dès que la puissance de 2 est supérieure ou égale à  $n$ , donc ici ce programme affiche 128.

```
1 n = 100
2 p = 1
3 while p < n:
4     p = 2 * p
5 print(p)
```

Entrées :  $n = 100$ ,  $p = 1$

$p$	« $p < n$ » ?	nouvelle valeur de $p$
1	oui	2
2	oui	4
4	oui	8
8	oui	16
16	oui	32
32	oui	64
64	oui	128
128	non	

Affichage : 128

### 7 1 Addition de $a$ et $b$

La syntaxe est  $a + b$ . Addition de  $a$  et  $b$ . Si les variables  $a$  et  $b$  sont des chaînes de caractères, on parle de **concaténation**.

### 7 2 Produit de $a$ et $b$

La syntaxe est  $a * b$ . Multiplication de  $a$  par  $b$ . Si les variables  $a$  et  $b$  sont des chaînes de caractères, on parle de **répétition**.

### 7 3 Division de $a$ par $b$ non nul

La syntaxe est  $a/b$ .

### 7 4 Élévation de $a$ à la puissance $b$

La syntaxe est  $a^{**}b$ .

### 7 5 Racine carrée de $a$

La syntaxe est  $\text{sqrt}(a)$  mais il faut importer la bibliothèque `math`.

### 7 6 Quotient de la division euclidienne de $a$ par $b$

La syntaxe est  $a//b$ .

### 7 7 Reste de la division euclidienne de $a$ par $b$

La syntaxe est  $a\%b$ .

Le module **turtle** permet de tracer facilement des dessins en Python. Il s'agit de commander une tortue à l'aide d'instructions simples comme « avancer », « tourner »... C'est le même principe qu'avec Scratch, avec toutefois des différences : on ne déplace plus des blocs, mais on écrit les instructions ; et en plus les instructions sont en anglais !

La tortue c'est l'ancêtre de Scratch ! En quelques lignes, on peut faire de beaux dessins.

