



المدرسة العليا
للتكنولوجيا - الصويرة
L'ÉCOLE SUPÉRIEURE DE
TECHNOLOGIE – ESSAOUIRA

Programmation en python

premier pas en python

Assuré par:
Pr. Mustapha JOHRI
microstoph@gmail.com

A PROPOS?

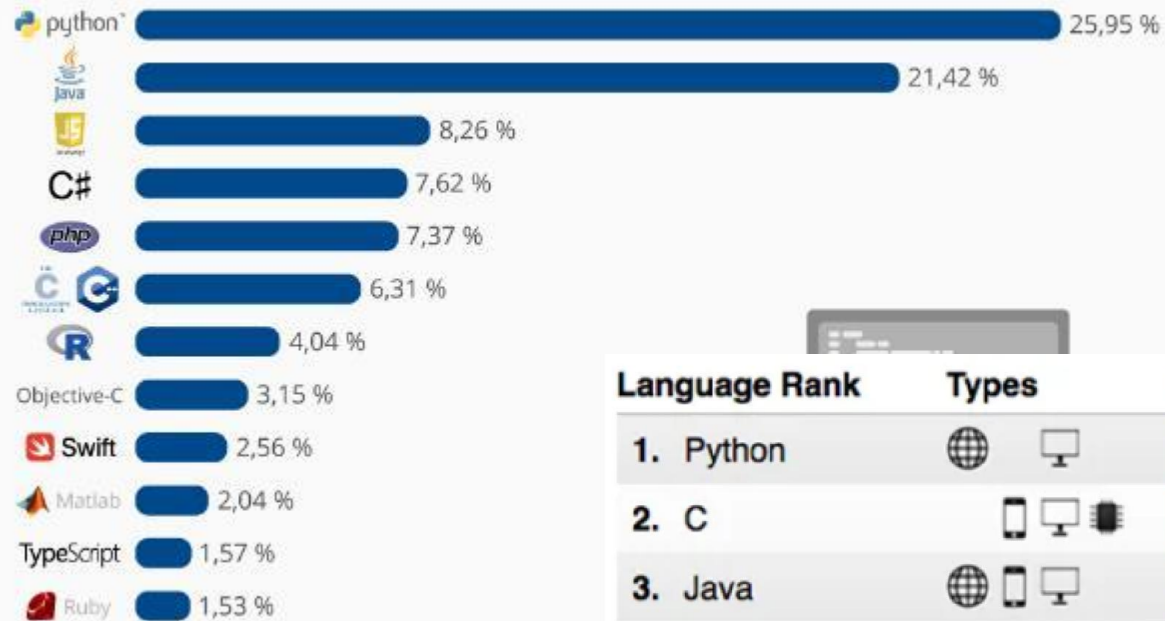
Pour expliquer l'origine du langage, revenons un peu en arrière. En 1989, par une froide nuit néerlandaise, un développeur du plat pays nommé **Guido van Rossum** s'ennuie. Il cherche un moyen de s'occuper pendant la période qui précède Noël car les bureaux de son entreprise sont fermés. Quand certains auraient décoré un sapin, lui se lance dans l'invention d'un langage. Etant un grand fan des **Monty Python** et d'humeur irrévérencieuse, il l'appelle **Python**. Voilà pourquoi les développeurs Python **ont de l'humour** et s'amusent à glisser des petites blagues dans leur code !



Pourquoi Python?

Les langages de programmation les plus populaires

Part des langages de programmation les plus populaires dans le monde selon le PYPL-Index *



* Basé sur une analyse de la fréquence de recherche des tutoriels de langage de programmation.

Source : PYPL

CC BY ND
@Statista_FR

Language Rank Types Spectrum Ranking

1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7

Pourquoi apprendre Python, le langage de programmation du futur

- ✓ Python est **un langage de programmation Open Source**, orienté **objet, de haut niveau**. Il s'agit d'un langage **généraliste**. Cela signifie qu'il peut être utilisé pour **développer à peu près n'importe quoi**, grâce à de nombreux outils et bibliothèques.
- ✓ Ce langage est particulièrement populaire **pour l'analyse de données et l'intelligence artificielle**, mais aussi pour le développement **web backend** et le **computing scientifique**.
- ✓ Python est aussi utilisé pour développer des **outils de productivité, des jeux** ou des applications. Des dizaines de milliers de sites web ont été développés avec ce langage, au même titre que plusieurs applications très connues comme **Dropbox, Netflix ou Spotify....**



Les différentes versions

- Il existe 2 versions de Python : 2.x et 3.x.
- Python 3.x n'est pas une simple amélioration ou extension de Python 2.x.
- Tant que les auteurs de librairies n'auront pas effectué la migration, les deux versions devront coexister.
- **Nous nous intéresserons uniquement à Python 3.x.**



Python vs C: Know what are the differences

Metrics	Python	C
<i>Introduction</i>	Python is an interpreted, high-level, general-purpose programming language.	C is a general-purpose, procedural computer programming language.
<i>Speed</i>	Interpreted programs execute slower as compared to compiled programs.	Compiled programs execute faster as compared to interpreted programs.
<i>Usage</i>	It is easier to write a code in Python as the number of lines is less comparatively.	Program syntax is harder than Python.
<i>Declaration of variables</i>	There is no need to declare the type of variable. Variables are untyped in Python. A given variable can be stuck on values of different types at different times during the program execution	In C, the type of a variable must be declared when it is created, and only values of that type must be assigned to it.

Python vs C: Know what are the differences

<i>Error Debugging</i>	Error debugging is simple. This means it takes only one instruction at a time and compiles and executes simultaneously. Errors are shown instantly and the execution is stopped, at that instruction.	In C, error debugging is difficult as it is a compiler dependent language. This means that it takes the entire source code, compiles it and then shows all the errors.
<i>Function renaming mechanism</i>	Supports function renaming mechanism i.e, the same function can be used by two different names.	C does not support function renaming mechanism. This means the same function cannot be used by two different names.
<i>Complexity</i>	Syntax of Python programs is easy to learn, write and read.	The syntax of a C program is harder than Python.
<i>Memory-management</i>	Python uses an automatic garbage collector for memory management.	In C, the Programmer has to do memory management on their own.

Python vs C: Know what are the differences

Example of a C Program -

```
1 #include <stdio.h>
2 int main()
3 {
4     // printf() displays the string inside quotation
5     printf("Hello, World!");
6     return 0;
7 }
```

Example of a Python Program -

```
1 | print("Hello, World!")
```

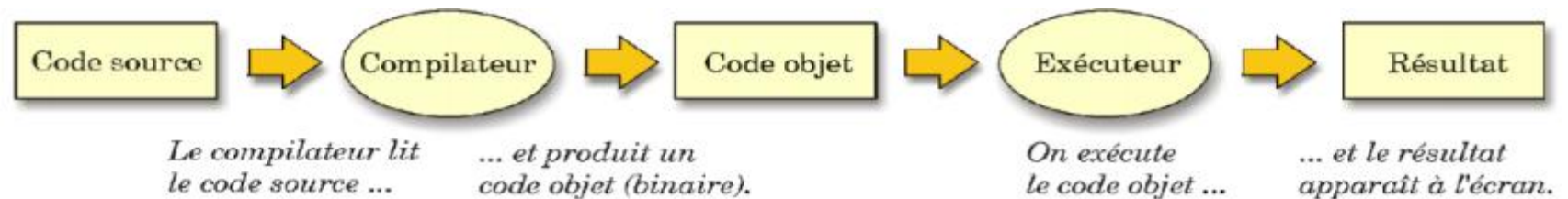
```
1  /* Simple C program */
2
3  #include <stdio.h>
4
5  int main() {
6      int year = 2021;
7      printf("Hello World!\n");
8      /* Notice that C's printf does not automatically put in a newline */
9      printf("Welcome IDSD %d.\n", year);
10     return 0;
11 }
```

```
1 # Simple Python program
2 year = 2021
3 print("Hello World!")
4 print("Welcome IDSD ",year)
5 |
```

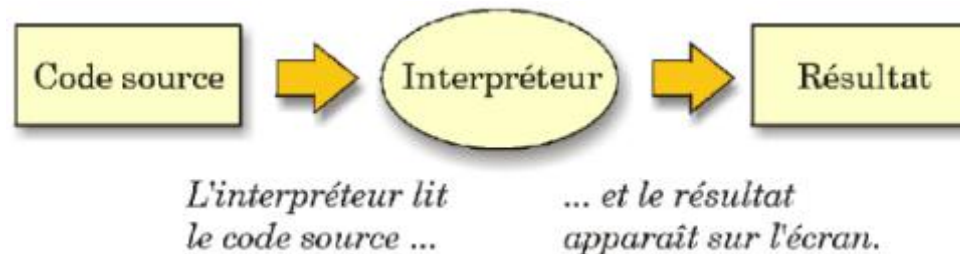

Comment faire fonctionner mon code source ?

Il existe 2 techniques principales pour effectuer la traduction en langage machine de mon code source :

- **Compilation :**



- **Interprétation:**



- **Python** est un langage **Interprété**

Et Python ?



- **Avantages** : interpréteur permettant de tester n'importe quel petit bout de code,...
- **Inconvénients** : peut être lent.

Comment lancer un script Python

Il y a différentes méthodes pour lancer un script Python

- Ecrire le code dans un fichier “**script.py**”. Et taper dans la console la commande “***python script.py***”
- Coder directement dans l’**interpréteur(console)** : instruction par instruction, un peu à la façon d’une calculatrice.
- Utiliser un **IDE** (eg. *Pyzo, PyCharm, Spyder, visual studio code,...*)



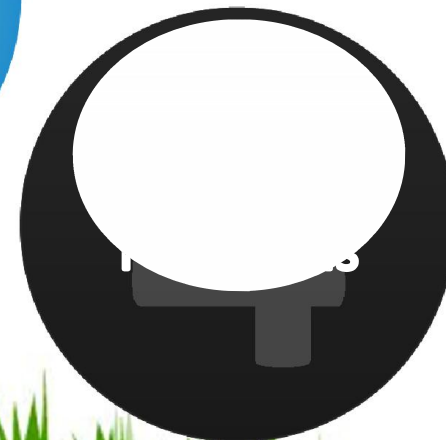
- Utiliser **Anaconda**(choix recommandé)



<https://www.anaconda.com/products/individual>



Plan



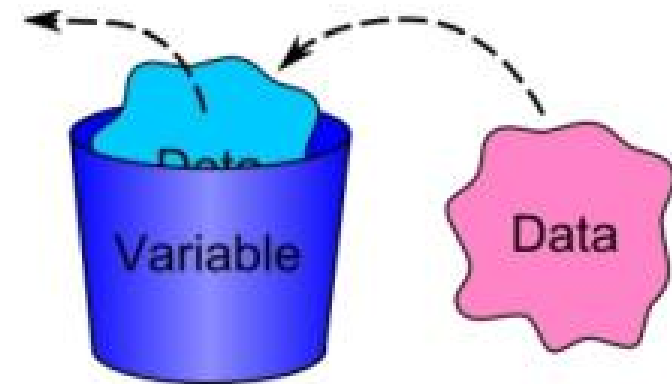


Variables, Affectations, Entrées/Sorties



Variables

- Conteneur d'information
- Identifié par un nom = un **identificateur**
- Ayant un « contenu »



- **Suite non vide de caractères**
 - commençant par une lettre ou le caractère _
 - contenant seulement des lettres, des chiffres et/ou le caractère _
 - Ne peut pas être un mot réservé de Python



Identificateurs en Python

➤ Exemples :

- **valides** : toto, proch_val, max1, MA_VALEUR, r2d2, bb8, _mavar
- **non valides** : 2be, C-3PO, ma var

➤ Les identificateurs sont **sensibles à la casse** :

ma_var != Ma_Var

➤ **Conventions** pour les variables en Python :

- utiliser des minuscules
- pas d'accents



Affectation

- Pour mémoriser une valeur dans une variable, on fait une affectation en utilisant le signe =

- **Exemples**

`n = 33`

`a = 42 + 25`

`ch = "bonjour"`

`euro = 6.55957`

- La première affectation d'une variable est aussi appelée **initialisation**
- En Python, le typage est **dynamique**
 - Pour connaître le type d'une variable: `type(ma_var)`



Examples

```
>>> a = 17
```

```
>>> type(a)
```

```
<class 'int'>
```

```
>>> a = "salut"
```

```
>>> type(a)
```

```
<class 'str'>
```

```
>>> a = 3.14
```

```
>>> type(a)
```

```
<class 'float'>
```

```
>>> type(21==7*3)
```

```
<class 'bool'>
```



Affectation vs Condition en Python

- Le signe “=” sert seulement à faire une affectation.
- Pour tester l'égalité, on utilise “==”

Exemples :

```
>>> a = 6
```

```
>>> a
```

```
6
```

```
>>> b = 9
```

```
>>> a == b
```

```
False
```



Expression

➤ C'est une formule qui peut être évaluée

➤ Exemples :

```
42 + 2 * 5.3  
3*2.0 - 5  
"bonjour"  
20 / 3
```

➤ Expression : des **opérandes** et des **opérateurs**.

➤ Les opérateurs que l'on peut utiliser **dépendent du type** des valeurs qu'on manipule

➤ Une expression qui ne peut prendre que les valeurs **True** ou **False** est appelée **expression booléenne**



Quelques opérateurs

➤ **Arithmétiques** (sur des nombres) :

$+$, $-$, $*$, $**$, $/$, $\%$, $//$

↓
Parenthesis
Power
Division
Multiplication
Soustraction
Addition
Left to Right

➤ **De comparaison** (résultat booléen) :

$==$, $!=$, $<$, $>$, $<=$, $>=$

➤ **Logiques** (entre des booléens, résultat booléen) :

or, and, not



Exercice

- Quelle est la réponse de l'interpréteur après chaque expression ?

<code>>>> 2 + 3</code>	<code>5</code>
<code>>>> 2*3</code>	<code>6</code>
<code>>>> 2**3</code>	<code>8</code>
<code>>>> 20/3</code>	<code>6.666666666666667</code>
<code>>>> 20//3</code>	<code>6</code>
<code>>>> 20%3</code>	<code>2</code>
<code>>>> 2 > 8</code>	<code>False</code>
<code>>>> (2 <= 8) and (8 < 15)</code>	<code>True</code>
<code>>>> 2 <= 8 < 15</code>	<code>True</code>
<code>>>> (x % 2 == 0) or (x >= 0)</code>	<code>NameError: name 'x' is not defined</code>



Les commentaires

En Python, une ligne d'instructions qui contient le symbole # (*dièse*) signifie un commentaire. Le reste de la ligne sera ignoré quand le programme sera exécuté.

a=2 voici un commentaire

b=5 # b=2

print(b) # un autre commentaire @!#!@\$

- Pour les commentaires sur plusieurs lignes en [python](#), nous aurons besoin d'utiliser le symbole('') ou (") **trois fois** au début et à la fin.

s = 1+2

'''

print("la somme est = ")

print () '''

print(s+1)



Entrées / Sorties

- On a généralement besoin de pouvoir **interagir** avec un programme :
- Pour lui fournir les données à traiter, par exemple au clavier : **entrées**
- Pour pouvoir connaître le résultat d'exécution ou pour que le programme puisse écrire ce qu'il attend de l'utilisateur, par exemple, texte écrit à l'écran : **sorties**



Les entrées : fonction `input()`

- A l'exécution, la fonction **`input`** :
 - interrompt l'exécution du programme
 - affiche éventuellement un message à l'écran
 - attend que l'utilisateur entre une donnée au clavier et appuie Entrée.
- C'est une saisie en **mode texte**
 - valeur saisie vue comme une **chaîne de caractères**
 - on peut ensuite changer le type



Les entrées : fonction input()

- Récupérer la données ???

➤ La fonction **input** : affiche un message (optionnel) à l'utilisateur et récupère la données saisie par l'utilisateur:

Syntaxe : **variable=input("message")**

Exemple

```
# demandez une valeur à l'utilisateur
nom = input(" Donnez votre nom : ")
prenom = input(" Donnez votre prenom : ")

# Affichage du nom complet
print( " Bienvenue :", nom, prenom)
```

Les entrées : fonction input()

- Récupérer la données ???

- Fonction **input** : retourne une valeur de type texte

- **Attention** : sous python '3' est différente de 3

- Fonction **eval** : évaluer et convertir en une valeur numérique une valeur contenu dans un texte

Exemple :

▪ <code>eval("34.5")</code>	retourne	34.5
▪ <code>eval("345")</code>	retourne	345
▪ <code>eval("3 + 4")</code>	retourne	7
▪ <code>eval("51 + (54 * (3 + 2))")</code>	retourne	321

On peut aussi utiliser les fonctions : **int, float, str,...**



Les entrées

```
>>> texte = input()  
123  
>>> texte + 1 # provoque une erreur  
>>> val = eval(texte)  
>>> val + 1 # ok  
124  
>>> x = float(input("Entrez un nombre :"))  
Entrez un nombre :  
12.3  
>>> x + 2  
14.3
```



Les sorties : fonction `print()`

- affiche la **représentation textuelle** de n'importe quel nombre de valeurs fournies entre les parenthèses et séparées par des virgules
- à l'affichage, ces valeurs sont séparées par un **espace**
- l'ensemble se termine par un retour à la ligne
 - modifiable en utilisant les options *sep* et/ou *end*
- Possibilité d'insérer
 - des **sauts de ligne** en utilisant `\n` et
 - des **tabulations** avec `\t`



Les sorties : fonction print()

```
>>> a = 20
>>> b = 13
>>> print("La somme de", a, "et", b, "vaut",
        a+b, ".")
La somme de 20 et 13 vaut 33.
>>> print(a,b,sep= ";")
20;13
>>> print("a=",a, "b=",b, sep="\n")
a=
20
b=
13
```



Les sorties : fonction `print()`

- On a déjà utilisé les chaînes de caractères, notamment dans les fonctions `print()` et `input()`.
- En Python, il existe 3 syntaxes pour les chaînes de caractères :
 - avec des **guillemets** :
`print("toto")`
 - avec des **apostrophes** :
`print('toto')`
 - avec des guillemets **triples** :
`print("""toto""")!`



Les sorties : fonction print()

- On peut utiliser " dans une chaîne délimitée par ' ... '
- On peut utiliser ' dans une chaîne délimitée par "..."
- On peut utiliser " et ' dans une chaîne délimitée par """" ... """"
- """"..."""" permet aussi d'écrire des chaînes de caractères sur plusieurs lignes (on y reviendra plus tard)



Exemples

```
>>> print("C'est toto")
C'est toto
>>> print('C'est toto')
SyntaxError : invalid syntax
>>> print("Il a dit "hello" !")
SyntaxError : invalid syntax
>>> print('Il a dit "hello" !')
Il a dit "hello"
>>> print("""C'est toto qui a dit "hello" !""")
C'est toto qui a dit "hello" !
>>> print("""C'est toto qui a dit "hello""")
SyntaxError : ...
```



Les sorties : Autres façons

La méthode *.format()* permet une meilleure organisation de l'affichage des variables.

```
1 >>> x = 32
2 >>> nom = "John"
3 >>> print("{} a {} ans".format(nom, x))
4 John a 32 ans
```

```
1 >>> x = 32
2 >>> nom = "John"
3 >>> print("{0} a {1} ans".format(nom, x))
4 John a 32 ans
5 >>> print("{1} a {0} ans".format(nom, x))
6 32 a John ans
```

```
>>> prop_GC = (4500 + 2575) / 14800
>>> print("La proportion de GC est", prop_GC)
La proportion de GC est 0.4780405405405405
```

```
1 >>> print("La proportion de GC est {:.2f}".format(prop_GC))
2 Le proportion de GC est 0.48
3 >>> print("La proportion de GC est {:.3f}".format(prop_GC))
4 La proportion de GC est 0.478
```


Les sorties : Autres façons

- Un peu comme le C

```
1 >>> x = 32
2 >>> nom = "John"
3 >>> print("%s a %d ans" % (nom, x))
```

- **Fstring** : Une syntaxe simple est similaire à celle qu'on a utilisée avec **str.format ()** mais moins verbeuse. Regardez à quel point c'est facile à lire:

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

```
>>> f"{2 * 37}"
'74'
```



Exercices

- **Exercice 1 :**

Ecrire un programme qui permet d'échanger le contenu de deux variables a et b

Exécution du programme

Donnez la valeur de a : 1

Donnez la valeur de b : 5

Avant l'échange a = 1 et b = 5

Après l'échange a = 5 et b = 1

```
a = eval(input(" Donnez le premier nombre : "))
b = eval(input(" Donnez le deuxième nombre : "))
print(" avant l'échange a=",a,"b=",b)
a,b=b,a
print(" après l'échange a=",a,"b=",b)
```

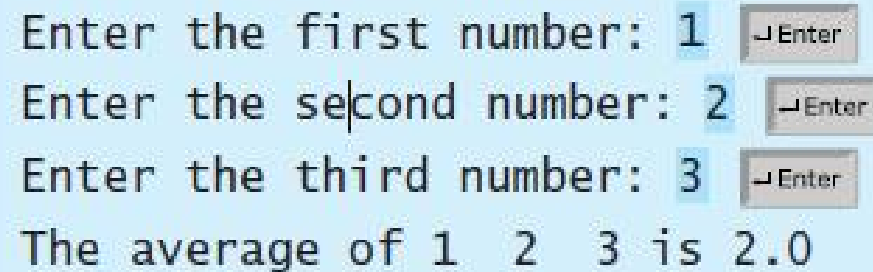


Exercices

- **Exercice 2 :**

Ecrire un programme qui calcule la moyenne de trois valeurs saisie par un utilisateur et affiche le résultat.

voici l'exécution :



Enter the first number: 1 ↵ Enter
Enter the second number: 2 ↵ Enter
Enter the third number: 3 ↵ Enter
The average of 1 2 3 is 2.0

```
a = eval(input(" Donnez le premier nombre : "))  
b = eval(input(" Donnez le deuxième nombre : "))  
c = eval(input(" Donnez le troisième nombre : "))  
moy=(a+b+c)/3  
# Affichage du résultat  
print(" la moyenne de ",a,b,"et",c,"est",moy)
```

Exercices

- **Exercice 3**: Ecrire un programme qui permet d'afficher le nombre d'heure, de minutes et de secondes restantes à partir d'un nombre de seconde saisie par l'utilisateur
- Exécution du programme

Donnez le nombre des secondes : 500

500 secondes vaut : 0 h 8 min 20 s

```
nbs = eval(input(" Donnez le nombre des secondes : "))  
h=nbs//3600  
m=(nbs%3600)//60  
s=(nbs%3600)%60  
print(nbs,"secondes vaut : ",h,"h",m,"min",s,"s")
```