

# Programmation en Python

Polytech Marseille, GII, 3A

Séverine Dubuisson, Simon Vilmin

`severine.dubuisson@univ-amu.fr`,  
`simon.vilmin@univ-amu.fr`

2023 - 2024



# À propos du cours

## Objectifs :

- maîtriser les bases de Python pour implémenter des algorithmes

## Volume horaire :

- 14 séances de Cours-TD
- 10 séances de TP

## Évaluation :

- un TP noté + un examen terminal + un bonus d'investissement


## Ressources :

- supports sur AMeTICE

# Sources

- (principalement) les anciens cours de M. Bulot
- cours d'introduction à Python de l'Université de Grenoble (Caséine)  
[moodle.caseine.org/course/view.php?id=87](https://moodle.caseine.org/course/view.php?id=87)
- Gérard Swinnen - Apprendre à programmer avec Python 3, 2012  
[www.eyrolles.com/Informatique/Livre/apprendre-a-programmer-avec-python-3-9782212134346/](http://www.eyrolles.com/Informatique/Livre/apprendre-a-programmer-avec-python-3-9782212134346/)
- Mark Lutz - Learning Python (5th edition), 2013  
[learning-python.com/about-lp5e.html](http://learning-python.com/about-lp5e.html)
- et bien sur, la [doc](#), [Wikipédia](#) et [stack overflow](#) !

# Programmer dites-vous ?

 **Définition** : (source [Wikipédia](#)) la *programmation* [...] désigne l'ensemble des activités qui permettent l'écriture des programmes informatiques. C'est une étape importante du développement de logiciels (voire de matériel).

 **Définition** : (source [Wikipédia](#)) un *programme informatique* est un ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur.

## Pourquoi faire ?

- automatiser des tâches complexes (très général),
- créer des logiciels, des jeux, des sites webs, faire de la recherche, ...
- mieux comprendre les outils numériques et apprendre à structurer sa pensée
- ~~trouver un travail et gagner du FRIC~~

# Le langage Python


**i Remarque :** pour écrire un programme on utilise un *langage de programmation*, souvent compréhensible par l'humain.


Dans ce cours, on utilise le Python :

- Créé dans les années 80 par Guido Van Rossum
- dernière version : Python 3.11
- langage *interprété* conçu sur le paradigme *orienté-objet* ...
- ... mais avec lequel on fait de l'impératif, de l'orienté-objet, du fonctionnel, etc
- documentation : [docs.python.org](https://docs.python.org)
- bonnes pratiques : [peps.python.org](https://peps.python.org), [clean code](#)




## Langage « interprété » ?

 **Question :** comment l'ordinateur exécute-t-il un programme (Python ou autre) ?

 **Réponse :** il faut d'abord le traduire le langage « humain » en langage machine ! La machine peut ensuite effectuer les opérations.

Deux grands moyens de faire cette traduction :

- *Compilation* : le programme est traduit une fois pour toute (compilé) en langage machine exécutable
- *Interprétation* : le programme est traduit au fur et à mesure et à chaque lancement. On peut exécuter des instructions à la volée sur l'*interpréteur*

 **Remarque :** Python est un langage *interprété*.

# Pourquoi Python ?

- langage TRÈS utilisé
- très haut niveau (proche du pseudo-code)
- des applications variées (web, science des données, embarqué, ...)

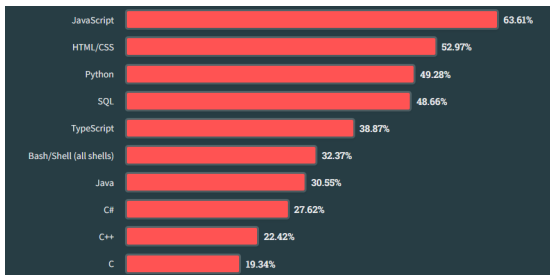


Figure – les 10 langages les plus utilisés/souhaités sur l'année 2022-2023, sondage effectué sur  $\simeq 80k$  personnes (source [survey.stackoverflow.co/2023/](https://survey.stackoverflow.co/2023/))

**! Attention :** la popularité d'un langage dépend aussi du domaine d'application !

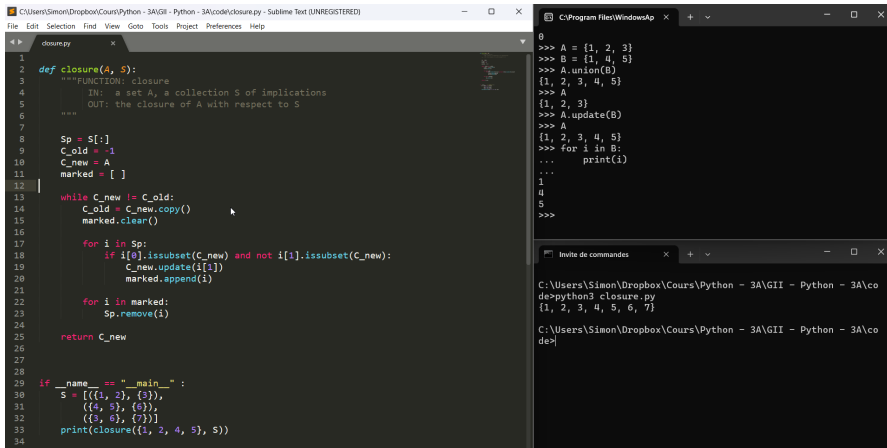
# Pourquoi pas Python ?

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Figure – Résultats d'une étude sur la consommation des langages de programmation (2017, source : [energy-efficiency-languages/results](https://energy-efficiency-languages/results))



# Comment faire du Python ?

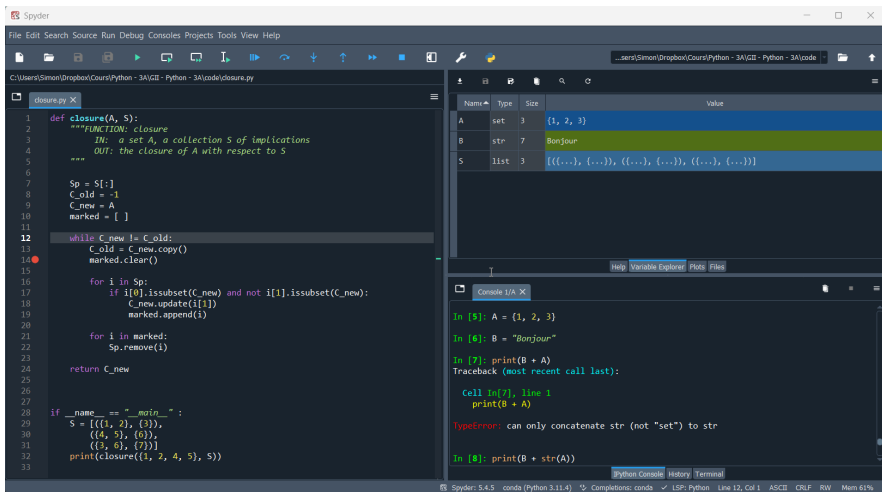


The screenshot displays a Python development environment with three windows:

- Code Editor (Sublime Text):** Shows the implementation of a `closure` function. The function takes a set `A` and a collection `S` of implications. It iteratively builds the closure `C_new` by adding elements from `S` that are subsets of the current closure. The main block tests the function with a specific set and collection.
- Interactive Shell (Python 3.6):** Shows the execution of the `closure` function. It defines `A = {1, 2, 3}` and `B = {1, 4, 5}`, then calls `A.union(B)` and `A.update(B)` to demonstrate set operations.
- Command Prompt (Shell):** Shows the execution of the `python3 closure.py` command, which outputs the result of the closure function: `{1, 2, 3, 4, 5, 6, 7}`.

Du mix interpréteur, éditeur de texte, invite de commande (ou shell) ...

# Comment faire du Python « confortablement » ?



The screenshot displays the Spyder Python IDE interface. The left pane shows a code editor with a Python script named `closure.py`. The script defines a `closure` function that takes a set `A` and a collection `S` of implications, and returns a new collection `C_new` of implications. The right pane shows the Variable Explorer, which displays the current state of the program's variables. The bottom pane shows the IPython Console, which displays the output of the code execution.

**Code Editor (closure.py):**

```
1 def closure(A, S):
2     """FUNCTION: closure
3     IN: a set A, a collection S of implications
4     OUT: the closure of A with respect to S
5     """
6
7     Sp = S[:]
8     C_old = -1
9     C_new = A
10    marked = [ ]
11
12    while C_new != C_old:
13        C_old = C_new.copy()
14        marked.clear()
15
16        for i in Sp:
17            if i[0].issubset(C_new) and not i[1].issubset(C_new):
18                C_new.update(i[1])
19                marked.append(i)
20
21        for i in marked:
22            Sp.remove(i)
23
24    return C_new
25
26
27
28 if __name__ == "__main__":
29     S = [({1, 2}, {3}),
30          ({4, 5}, {6}),
31          ({3, 6}, {7})]
32     print(closure({1, 2, 4, 5}, S))
33
```

**Variable Explorer:**

Name	Type	Size	Value
A	set	3	{1, 2, 3}
B	str	7	Bonjour
S	list	3	[({1, 2}, {3}), ({4, 5}, {6}), ({3, 6}, {7})]

**IPython Console:**

```
In [5]: A = {1, 2, 3}
In [6]: B = "Bonjour"
In [7]: print(B + A)
Traceback (most recent call last):
  Cell In[7], line 1
    print(B + A)
TypeError: can only concatenate str (not "set") to str

In [8]: print(B + str(A))
```

... aux environnements de développements intégrés (ici Spyder)

# Les TP sont sur Spyder

exécuter debugger

vue des variables

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Oct 4 09:02:44 2023
4
5 @author: Simon
6 """
7
8 """ factorielle
9 entree: un entier n
10 """
11 calcule n!
12
13 def factorielle(n):
14     if n <= 1:
15         return 1
16     else:
17         return n * factorielle(n - 1)
18
19
20 if __name__ == "__main__":
21     n = int(input("entrez un entier :"))
22     res = factorielle(n)
23     print(f"n! = {res}")
24
```

Var	Type	Size	Value
n	int	1	24
res	int	1	620448401732239439360000

Console 2/6

```
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:47:18) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python -- An enhanced Interactive Python.

In [1]: n=factorielle(24)
entrez un entier :24
24! = 620448401732239439360000

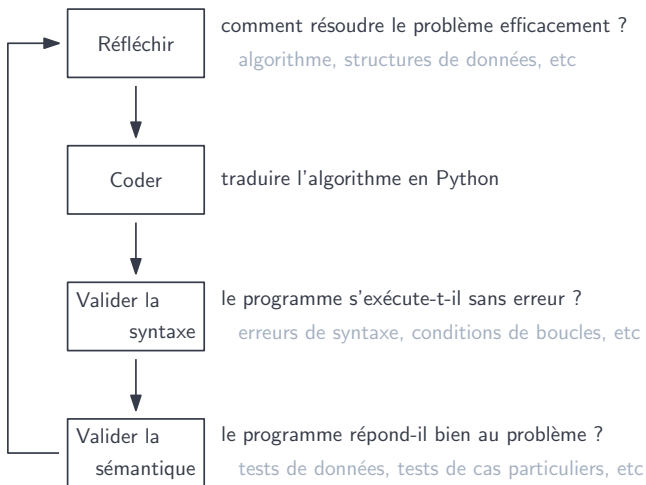
In [2]:
```

programme (.py)

interpréteur

# Réfléchir et coder

**! Attention :** programmer ça n'est pas écrire du code au hasard !



# Plan

- Premiers pas en Python
- Structures conditionnelles et itératives
- Fonctions
- Types structurés
- Fichiers et Erreurs
- Quelques modules
- Introduction à l'objet