



## Ce que je pense ;-)

- ⇒ Python permet de résoudre des problèmes de seuil, MAIS le tableur est parfois plus naturel à utiliser ;
- ⇒ Python permet de tracer des histogrammes, gérer des stats MAIS GeoGebra aussi et est souvent plus convivial (ou la calculatrice) ;
- ⇒ Python permet de faire du calcul formel, MAIS Xcas ou les modules de résolution de système de la calculatrice sont plus adaptés ;
- ⇒ Faire attention à la valeur ajoutée / temps d'apprentissage !
- ⇒ On apprend un langage de programmation par mimétisme et par besoin : choix d'écrire des programmes un peu « ambitieux », le travail élève étant réduit compléter des boucles et/ou des tests.

## Documents

- ceux du site EduPython : <https://edupython.tuxfamily.org/> MAIS je ne pense pas que ce soit une bonne idée d'utiliser le module `lycee`.
- le livre de Vincent MAILLE : *Python, les bases de l'algorithmique et de la programmation*.
- le livre de Gérard SWINNEN : *Apprendre à programmer avec Python 3*. Existe en version papier et en version .pdf gratuite : <https://inforef.be/swi/python.htm>
- des fiches de résumé : <https://perso.limsi.fr/poital/python:memento>

## Les différentes distributions / différents éditeurs

- Edupython : <http://edupython.tuxfamily.org/> (installable en version portable)
- Portable Python : <https://portablepython.com/wiki/PortablePython3.2.5.1/>
- Numworks (émulateur en ligne) : <https://www.numworks.com/fr/>
- Cappytale via l'ENT mon lycee.net (mais il faut être connecté à Internet et il faut avoir un compte sur l'ENT)
- Jupyter <https://jupyter.org/> (il faut être connecté à Internet)
- Basthon <https://basthon.fr/> (il faut être connecté à Internet)

Éditeurs : Attention ! tabulations / espaces



## Importance du code

Le langage Python (en hommage aux Monty Python) a été créé par Guido van Rossum.

Le langage évolue grâce à sa communauté : chaque proposition d'amélioration est publique et est publiée sur le site de Python, ce sont les PEP (Python Enhancement Proposal : proposition d'amélioration de Python). <https://www.python.org/dev/peps>

### Recommandations de la PEP 8 :

Un code est plus souvent lu qu'écrit ! Commenter les lignes par # et décrire les fonctions à l'aide de `""" ... """` (trois double quotes).

Voici quelques recommandations :

- Une ligne doit contenir 80 caractères maximum.
- L'indentation doit être de 4 espaces.
- Chaque fonction est séparée de la suivante par une ligne vide.
- Les noms de variable, fonction, ... sont composés uniquement de lettres non accentuées ou de chiffres et éventuellement de `_`. Les noms de constante seront en majuscule, les autres en minuscule. Les noms de fichiers ne peuvent pas commencer par un chiffre.

Pour conclure, le poème « Zen of Python » de Tim Peters résume cette *philosophie pythonique*. Taper dans la console `import this`

## Modules, Librairies, Bibliothèques...

---

```
1 from math import sqrt
2     # permet l'accès à la fonction <sqrt> du module <math>
3 from math import *
4     # permet l'accès à toute les fonctions du module <math>
5 from numpy import sqrt
6     # permet l'accès à la fonction <sqrt> du module <numpy>.
7     # Attention : quand deux fonctions ont le même nom,
8     # c'est la dernière chargée qui sera utilisée !
```

---



---

```
1 import math as m
2     # les fonctions du module <math> seront précédées de <m.
   ↔>
3 import numpy as np
4     # les fonctions du module <numpy> seront précédées de <
   ↔np.>
5
6 sqrt(48)     # fonction <sqrt> (dernière chargée)
7 m.sqrt(48)  # fonction <sqrt> du module <math>
8 m.sqrt(-48) # fonction <sqrt> du module <math>
9 m.sqrt([9, -4, 1+3j]) # fonction <sqrt> du module <math>
10 np.sqrt(-48) # fonction <sqrt> du module <numpy>
11 np.sqrt([9, -4, 1+3j]) # fonction <sqrt> du module <numpy>
```

---

## Variables

Python ne protège que 33 mots clés : tous les autres peuvent être redéfinis sans avertissement ! (par exemple si on tape  $\pi=2$ , alors le nombre  $\pi$  n'est plus affecté à 3,14157... mais à 2!!)

---

```
1 import keyword as kw
2
3 # affiche les mots protégés
4 print(kw.kwlist)
5 # la longueur de la liste donne le nb. de mots protégés
6 len(kw.kwlist)
```

---

Les variables sont locales à la fonction, si on utilise une variable globale il faut le préciser.

On peut définir des « fonctions filles » dans des « fonctions mères » : les « fonctions filles » seront locales à la « fonction mère ».

Par exemple :



---

```
1 def quotient(a, b):
2     """ renvoie le quotient entier de <a> et <b> """
3     # remarque : a // b renvoie le quotient entier
4     k = 0
5     while a >= b:
6         a = a - b
7         k = k + 1
8     print(f"fct quotient, fin de la boucle while : a = {a}")
9     return k
10
11 def QetR(a, b):
12     """ renvoie le quotient et le reste de la division
13     entière de <a> par <b>
14     """
15     # utilisation des f-string
16     print(f"appel du programme : a={a} et b={b}")
17     q = quotient(a, b)
18     # l'opérateur % renvoie le reste de la division
19     ↔ euclidienne
20     print(f"fct QetR, après appel à la fct quotient : a = {a
21     ↔}")
22     return q, a % b
```

---

La variable `a` est modifiée dans la fonction `quotient` mais pas dans la fonction d'appel `QetR`.



Une fonction peut être passée en paramètre d'une autre :  
dans la console : `image(3, f)`  
revoie l'image de 3 par la fonction  $f$ .

---

```
1 def f(x):  
2     return x**2 + 2  
3  
4 def g(x):  
5     return x + 1  
6  
7 def image(x, fct):  
8     return fct(x)
```

---

## Obtenir de l'aide

- dans la console : `help(nom)` ou `dir(nom)` (nom est un nom de fonction ou de bibliothèque)
- dans certains éditeur : les *docstring*
- sur le web : <https://docs.python.org/fr/3>
- les manuels scolaires
- un collègue sympa;-)