Du plaisir du prof…
au travail de l'élève.

Remarques en vrac

Ce que je pense ;-)

⇒ 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/pointal/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/*
- Anaconda : *https://www.anaconda.com/distribution/*
- Numworks (émulateur en ligne) : *https://www.numworks.com/fr/*
- Capytale 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 connecé à Internet)

Éditeurs : Attention ! tabulations / espaces

Remarques en vrac

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

Remarques en vrac

```
import math as m # les fonctions du module <math> devront
    # être précédées de <m.>
import numpy as np # les fonctions du module <numpy>
    # devront être précédées de <np.>

sqrt(48) #fonction <sqrt> (dernière chargée)
m.sqrt(48) #fonction <sqrt> du module <math>
m.sqrt(-48) #fonction <sqrt> du module <math>
m.sqrt([9, -4, 1+3j]) #fonction <sqrt> du module <math>
np.sqrt(-48) #fonction <sqrt> du module <numpy>
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 3,141 57… mais à 2 !!)

```
import keyword as kw
print(kw.kwlist)
len(kw.kwlist) # la longueur de la liste est le nb.
    # de mots protégés en Python3
```

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

Par exemple :

```
def quotient(a, b):
  """ renvoie le quotient entier de <a> et <b> """
  # remarque : a // b renvoie le quotient entier
  k = 0
  while a >= b:
    a = a - b
```

Remarques en vrac

```
7       k = k + 1
8   # print("fct quotient, fin de boucle a=",a)
9     return k
10
11  def reste(a, b):
12    """ renvoie le reste de la div. entiere de <a> par <b> """
13  # print("fct reste : a=",a)
14    return a - b * quotient(a, b)
15
16  def QetR(a, b):
17    """ renvoie le quotient et le reste de la division
18    entiere de <a> par <b> """
19    return quotient(a, b), reste(a, b)
```

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

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 ;-)