Mémo Python : les bases d'un programme

Conseil
Toujours avoir la documentation Python à portée de main. Vous trouverez celle-ci sur le site : `https://docs.python.org` (En prenant soin de sélectionner la bonne version de Python.)

Rappel
Il n'y a pas qu'une façon d'écrire un programme ! Le·a développeur·se est responsable de faire des choix en fonction du et des .

- Le programme est-il destiné à être lu par quelqu'un d'autre ?
- Le programme appartient-il à un projet ? Ce projet a-t-il des règles de nommage et de présentation particulières ?
- Le programme doit-il être fait dans l'urgence ?

Convention
En Python, on a pour convention d'écrire les noms de variables, de fonctions, etc. en « ». Il est courant de voir des programmes dans d'autres langages écrits en « ». Attention de rester cohérent au sein d'un même programme !

SNAKE CASE 🐍	CAMEL CASE 🐫
ma_variable	maVariable
ma_fonction()	maFonction()
une_autre_fonction(ma_variable)	uneAutreFonction(maVariable)
~~uneAutreFonction(ma_variable)~~	

1 Mode interactif vs mode programme

1.1 Le mode interactif

Il s'agit d'un dialogue direct avec l' Python via une console. On peut ouvrir un terminal ou une invite de commande et taper `python3` pour lancer le mode interactif. Certains IDE ont une console prévue pour ce mode (pas besoin de taper `python`) : c'est le cas de Thonny que nous utilisons en cours. Les trois chevrons (»>) indiquent qu'on est dans une console Python et qu'on peut taper une commande.
Sous Windows, il existe un shell Python nommé IDLE. Pour l'ouvrir, aller dans la barre des tâches de Windows et taper « IDLE ». Une fois lancé, constater les trois chevrons.

Vous pouvez essayer d'exécuter des instructions Python dans le shell, voici quelques exemples de commandes et leurs sorties :

```
>>> print("Hello world!")
Hello world!
>>> 2 * 3
6
>>> from math import pi
```

1.2 Le mode programme

Il consiste à écrire un programme dans un fichier puis à le faire par l'interpréteur.

Attention aux différences entre le mode console et le mode programme : si on appelle une fonction dans la console, la valeur renvoyée sera directement affichée. En revanche, dans un script qu'on exécute, l'appel à une fonction n'affichera pas directement sa valeur de retour, il faudra veiller à utiliser la fonction `print`.

Le #
Le symbole # en début de ligne n'est pas un hashtag. Il signifie qu'on écrit un , c'est-à-dire une ligne qui . On peut écrire ce qu'on veut pour aider la lecture du code ou pour mieux structurer le programme. Il ne faut pas non plus en abuser, ça ne doit pas se subsituer à la lisibilité du code !

Les """
Les triples guillements anglais servent à écrire de la de code. Ces chaînes de caractères s'appellent les *docstrings*. On trouve les recommandations pour les écrire et les utiliser ici : `https://peps.python.org/pep-0257/#what-is-a-docstring`

Il ne faut **pas confondre les commentaires et la documentation** de code : un commentaire sert au développeur, de manière à donner des indications pour ne pas oublier certaines détails. On remarque dans bien des cas qu'un bon nommage des variables évite des commentaires. Un code clair et lisible aussi. Il faut être vigilant à ceci.

La documentation a pour but de servir à l'extérieur du programme lui-même. On peut générer une page web contenant la documentation d'un programme qu'on a écrit de manière à en rendre l'utilisation claires à des personnes qui voudraient s'en servir. Elle permet de lister les fonctionnalités et de faciliter le travail en équipe. Il n'y a pas que des développeurs dans un équipe et il faut que les autres puissent savoir où on en est du travail et ce qu'on propose.

Nous verrons dans ce qui suit une bonne façon de structurer un programme tenant dans un fichier.

2 Structure d'un programme dans un fichier simple

2.1 Les modules importés

Les Python disponibles via la sont autant de boîtes à outils qu'on peut utiliser pour écrire nos propres programmes.

Exemple : le module `turtle` est issu de la bibliothèque standard.

On peut les fonctions d'un module de différentes façons :

- Ici, on peut s'imaginer qu'on amène une boîte à outils et que, à chaque fois qu'on a besoin d'outil, on précise de quelle boîte à outils il sort pour pouvoir l'utiliser :

```
import turtle
turtle.forward(50)
```

- Là, on sait qu'on n'aura besoin que de deux outils précis, donc on les sort de leur boîte pour les laisser près de nous et on peut les utiliser directement :

```
from turtle import forward, backward
forward(100)
```

- Enfin, on sait qu'on va utiliser de nombreux outils et qu'on ne veut pas les prendre dans leur boîte puis les ranger à chaque fois donc on sort tout (le symbole * représente «tout»). C'est comme si on renversait tout le contenu de la boîte à outils sur une table pour pouvoir utiliser ces derniers :

```
from turtle import *
forward(40)
right(90)
```

Ces trois façons de faire ne sont pas équivalentes, il faut choisir en fonction du besoin. Un problème peut se poser si on utilise deux modules dans lesquels des fonctions ont le même nom !

2.2 Les définitions de fonctions

Nous reviendrons dans un prochain cours sur les fonctions plus précisément. Ici, il faut juste retenir qu'on peut définir ses propres fonctions pour les utiliser ensuite dans la partie `script`.

Il est important de documenter ses fonctions à l'aide des *docstrings*, à savoir ce qui est dans les blocs de commentaires entre triples guillemets anglais `"""`. Cette documentation est ensuite visible grâce à la fonction Python `help`. L'idéal est de se référer aux bonnes

pratiques de Python appelées PEP8 lisibles ici : `https://peps.python.org/pep-0008/` et plus précisément pour les docstrings là : `https://peps.python.org/pep-0257/`

```
def ma_fonction(ma_variable):
    """ ma_fonction(<type de ma_variable>) -> <type de la valeur renvoyée, 'None' pas défaut>
    Description succinte de ce que fait la fonction.
    """
    ... # mon code ici
```

2.3 Le script

C'est la partie où on dit au programme quoi faire. Si on a nos propres fonctions plus haut, on peut les appeler ici, on peut faire des calculs, afficher des choses, etc.

2.4 Exemple de programme simple

```
# IMPORTS
from turtle import *

# FONCTIONS
def square(c):
    """ square(int) -> None
    Trace un carré.
    """
    for i in range(4):
        forward(c)
        right(90)

# SCRIPT
# taille de la fenêtre
setup(800, 600)
speed(10)
sides = [100, 80, 60, 40, 20]

for i in range(4):
    for c in sides:
        square(c)
        forward(c)
    backward(sum(sides))
    right(90)

# boucle maintenant la fenêtre ouverte
mainloop()
```