Modules et documentations

1. Introduction

Que l'on distribue du code ou que l'on utilise l'offre pléthorique de modules Python, la manipulation des modules et de la documentation fait partie intégrante du développement logiciel.

2. Programmation modulaire

Le **principe de la modularité** est particulièrement important dans tout développement logiciel. Il simplifie les tests, permet de réutiliser du code, et facilite la maintenance. Ce principe peut opérer à plusieurs niveaux :

- découper le code en fonctions ;
- grouper les fonctions dans une classe (on les appelle alors méthodes) ;
- grouper des fonctions par thèmes, dans un fichier séparé alors appelé **modules** ;
- grouper des fichiers (**modules**) en **bibliothèques** (**packages**).

La **programmation modulaire** intervient :

- lors d'écriture de partions de code pour les réutiliser plus tard, ou les distribuer ;
- lors de l'utilisation, pour répondre à un besoin, de code écrit par quelqu'un d'autre.

3. Importation de modules

Certains modules Python sont installés par défaut (bibliothèques standard) et d'autres peuvent être ajoutés en utilisant des outils comme **`pip`**.

Dans les exemples qui suivent, nous allons utiliser le module **`random`**.

- Import du module **`random`**. Les fonctions du module doivent être préfixées du nom du module.

```
import random
a = random.randint(1, 10)
```

- Import du module **`random`** et renommage en **`rd`** avec le mot clé **`as`** pour rendre le code plus lisible (le nouveau nom est généralement plus court).

```
import random as rd
a = rd.randint(1, 10)
```

- Import uniquement des fonctions nécessaires du module **`random`** (plus de préfixe, mais seules les fonctions importées sont disponibles).

```
from random import randint
a = randint(1, 10)
```

On peut aussi importer plusieurs fonctions :

```
from random import randint, choice
```

- Import du module **`random`** sans utilisation du préfixe. Cette utilisation est **déconseillée**.

```
from random import *
a = randint(1, 10)
```

Avec **`import *`**, toutes les fonctions sont disponibles sans utilisation de préfixe, ce qui peut paraître efficace. En réalité, il n'en est rien : on ne maitrise plus la liste exacte de ce qui est importé. De plus, plusieurs fonctions, provenant de différents modules, peuvent avoir le même nom, ce qui sera source de problèmes lors de l'utilisation des fonctions.

4. Documentation (docstrings)

Le docstring permet de décrire une fonction, une méthode, une classe, un module ou un package.

Cette documentation est ensuite consultable à l'aide de la commande **help** :

```
>>> import random
>>> help(random)           # affiche la docstring du module
>>> help(random.randint)   # affiche la docstring de la fonction
```

Pour documenter un module, il suffit de créer la docstring au début du fichier.

Exemple : module de tri (fichier tri.py)

```
"""Module de tri

ce module contient deux méthodes de tri :
- le tri par insertion
- le tri par sélection
"""


def tri_insertion(liste):
    """tri une liste par insertion"""
    . . .
    . . .
    return liste


def tri_selection(liste):
    """tri une liste par sélection"""
    . . .
    . . .
    return liste
```

5. Création d'un module

✍ **Implémenter** et **documenter** un module **aires** contenant des fonctions permettant le calcul de l'aire d'un :

- carré ;
- rectangle ;
- triangle ;
- disque ;
- parallélogramme ;
- losange.

✍ Implémenter un petit programme afin de tester le fonctionnement de ce module.

Source : https://lyceum.fr/tg/nsi/4-langages-et-programmation/3-modularite
livre prépabac NSI éditions Hatier