Fiche 3.0 : scripts et modules

BUT Informatique
IUT de Vélizy

1 Notion de script

Jusqu'à présent, vous avez écrit du python dans l'environnement interactif *jupyter*. Mais en général les programmes python sont indépendants de ce genre d'environnement et peuvent être exécutés "directement".
Un *script* python est simplement un fichier texte, dont l'extension est en général `.py` et qui contient du code. Un programme python est en général un ensemble de scripts.
En python, un script s'appelle un *module*, il y a quelques nuances mais ça ira pour l'instant. Plus de détails dans `https://docs.python.org/fr/3/tutorial/modules.html`
Pour écrire un tel script, on peut utiliser un simple *éditeur de texte* (gedit, vim, emacs, geany, sublimetext,...) ou un *environnement de développement intégré* (IDE) (eclipse, pycharm, jetbrains,vscode,...), qui est un éditeur de texte avec de nombreuses fonctions supplémentaires pour écrire du code.
Dans cette séquence, afin de ne pas tout mélanger, nous utiliserons l'éditeur **geany** qui est très simple. Par la suite, vous ferez comme vous voudrez.

2 Prise en main

☐ 0. Ouvrez l'éditeur de texte *geany*. Tapez quelques lignes de code python dans l'éditeur, par exemple effectuez une boucle pour écrire 10 fois "bonjour".

☐ 1. Enregistrez dans votre dossier de la sequence3 ce programme sous le nom `first_script.py`

☐ 2. Maintenant, ouvrez un terminal et naviguez en ligne de commande jusqu'à vous trouver dans le répertoire où se trouve le script.

☐ 3. Tapez dans la ligne de commande

```
python3 first_script.py
```

N'oubliez pas que la touche TAB permet de compléter automatiquement. En principe, vous devriez voir dans le terminal la sortie de votre programme. Sinon, c'est peut-être que vous avez un message d'erreur. Corrigez le script et recommencez.

☐ 4. Retenez la procédure ci-dessus, il faudra toujours la faire pour exécuter les programmes python en ligne de commande.

3 Ecriture d'une librairie *todolist* pour gérer les trucs à faire

Quand vous exécutez le script, python va tout lire et exécuter, dans l'ordre du script, de haut en bas. Quand python tombe sur la définition d'une fonction,(`def ...`) il n'exécute pas le code de la fonction, il prend juste note de son existence. Pour vraiment exécuter la fonction, il faut l'appeler !

Vous allez maintenant réaliser un petit module avec quelques fonctions ayant pour but de gérer une liste de tâches à réaliser ou `todolist`. Plus précisément, on veut avoir en mémoire un ensemble de tâches à réaliser, avec trois priorités possibles, comme :

```
Tâches priorité haute :
0. réviser cours dev
1. appeler Mamie
Tâches priorité moyenne :
2. vaisselle
Tâches priorité basse :
3. finir de regarder F&F 42
4. réviser encore cours dev
```

Vous pouvez imaginer autre chose et faire des variations plus avancées sur ce principe (possibilité de prévoir des tâches à des dates fixées, plus de priorités, etc.)

☐ 5. Créez deux scripts, `todolist.py` et `test_todo.py`.
   — Le premier contiendra l'ensemble des fonctions pour manipuler la todolist, et uniquement des fonctions destinées à être utilisées par un programme. Il s'agit d'une librairie, un ensemble de fonctions destinées à gérer les todolist. Si on exécute ce script directement en ligne de commande, rien ne se passera car aucune des fonctions n'est appelée, il n'y a pas de code principal.
   — le second script contient des tests destinés à vérifier si nos fonctions de l'autre script sont bien écrites. Après les fonctions de test, le code principal de ce script lance les tests.

☐ 6. A vous de décider comment vous allez gérer la sauvegarde des tâches et leur priorité : plusieurs listes, dictionnaire de listes, liste de listes ... A vous de faire au mieux, au plus simple et surtout au plus efficace ! Toutes les fonctions que vous allez écrire devront avoir ces objets en paramètre.

☐ 7. On va créeer, dans le script `todolist.py`, une fonction destinée à afficher la liste des tâches comme ci-dessus. Ecrivez le *prototype* de cette fonction : le mot clé `def`, le nom de la fonction, et pour l'instant ne mettez pas de code (juste `pass`).

☐ 8. Ecrivez maintenant la spécification de la fonction en docstring : que sont les paramètres, que fait la fonction, qu'est-ce qu'elle renvoie ? Respectez le format utilisé dans la séquence précédente.

☐ 9. Dans le script `test_todo.py`, qui doit se trouver dans le même répertoire, écrivez `import todolist` avant votre code. Ceci permettra d'utiliser les fonctions de l'autre script dans celui-ci. Ecrivez maintenant une fonction de test de votre affichage : il faut créer (« à la main »pour l'instant) une liste de tâches au format que vous avez choisi, appeler la fonction d'affichage en lui donnant cette liste en paramètre. Faites aussi un test pour un cas où il n'y a pas de tâches à afficher. En principe, cette fonction de test ne prends pas de paramètres.

   note : on ne peut pas faire un `assert` pour vérifier l'affichage. Il faudra regarder avec nos yeux si ça correspond bien à ce qu'on veut.

☐ 10. Il est temps d'écrire maintenant la fonction d'affichage ! Supprimez `pass` et écrivez le code.

☐ 11. Dans le script de tests, en dessous des fonction de tests, écrivez le « code principal »de ce script (ou *main*) : il s'agit juste d'un appel à votre fonction de test.

☐ 12. Maintenant, en ligne de commande, lancez votre script de tests. La dernière ligne est exécutée : elle appelle la fonction de test, qui elle même appelle la fonction de d'affichage du script `todolist.py`. Si tout se passe bien, l'affichage a lieu. Sinon, il doit y avoir un bug quelque part !

☐ 13. Respectez scrupuleusement les mêmes étapes (spécification, test, développement) pour l'écriture d'une fonction qui permet d'ajouter une nouvelle tâche dans la todolist. Attention, cette fonction ne lit pas de saisie au clavier.

☐ 14. Idem pour supprimer une tâche, pour changer la priorité d'une tâche, et toute autre fonction que vous pourrez imaginer qui pourrait servir dans votre librairie.

☐ 15. Ecrivez un test complet de votre librairie : cela crée une todolist, ajoute quelques tâches, les affiche, supprime et ajoute d'autres tâches, affiche, etc.