Fiche : Modularité en Python

Les modules et packages permettent de découper un programme en plusieurs fichiers et de regrouper ceux-ci de manière logique.

1 Python en dehors des notebooks

Python interactif : La commande `python` dans un terminal transforme celui-ci en une console Python interactive. Chaque instruction Python saisie est directement exécutée.

Fichiers de code Python : Il est possible d'écrire du code Python dans un fichier (extension `.py` et codage en utf-8) que l'on peut exécuter dans un terminal avec la commande : `python nom_du_fichier_python` (si le fichier est exécutable et commence par la ligne `#!/usr/bin/env python`, il peut être exécuté en tapant directement `./nom_du_fichier_python`).

2 Modules

Un module est un fichier de code Python (le nom de fichier d'un module est le nom du module suffixé de `.py`) qui peut ensuite être inclus dans un autre fichier Python ou dans un notebook. Ces derniers peuvent alors appeler les fonctions définies dans le module.

Pour utiliser les fonctions d'un module, il faut auparavant importer ce module. Il existe deux manières d'importer un module.

1- On inclut le module avec l'instruction `import ... as ...` où ce qui suit le mot clé `as` est le nom donné au module dans le code. Les fonctions sont appelés selon le nom donné au module suivi du `.` puis du nom de la fonction.

[ ]:
```
import module as nom_module

mom_module.fonction(val1, val2, ..., valk) # fonction est définie dans module
```

2 - On inclut le module avec l'instruction `from ... import ...`. Dans ce cas, les fonctions définies dans le module sont appelées directement dans le code.

[ ]:
```
from module import fonction, autre_fonction # etc
fonction(val1, val2, ..., valk)
```

Remarque : seules les fonctions du module dont les noms sont après le mot clé `import` peuvent être utilisées dans le script. Pour importer toutes les fonctions du module, on peut écrire directement :

[ ]:
```
from nom_du_module import * # importe toutes les fonctions du module
```

Attention : les fonctions importées de cette manière écrasent les fonctions ayant le même nom dans le script.

Utiliser des modules comme des scripts Un module est utilisé dans d'autres scripts Python. Il est néanmoins possible d'y inclure du code qui ne sera exécuté que si ce module est le fichier exécuté (autrement dit, l'instruction `python nom_du_module.py` est exécutée). Ceci se fait en utilisant :

[ ]:
```
if __name__ == '__main__':
    # Code qui sera exécuté uniquement si le module est exécuté
    # directement (pas inclus dans un autre script)
```

3 Packages en Python

Les packages en Python permettent de regrouper logiquement des modules ensemble. La création de packages est simple puisqu'un package correspond à un répertoire : les fichiers (et répertoires) à l'intérieur correspondent aux modules (et sous-packages) contenus dans le package.

On importe un module d'un package selon l'une des deux méthodes :

[ ]:
```
import package.module as nom_module
from package.module import nom_fonction1, nom_fonction2 # etc
```

Utiliser des modules et packages qui ne sont pas dans le répertoire courant Si le module ou le package n'est pas dans le même répertoire que le script (ou notebook) qui les utilise, alors Python ne le trouve pas et cela engendre une erreur. Pour corriger ce problème, il faut ajouter le répertoire contenant le module (ou package) dans le chemin de Python. Pour cela on utilise le package `sys` qui permet d'ajouter des nouveaux chemins :

```
# chemin doit être une chaîne de caractères correspondant au chemin (relatif ou absolu)
# du répertoire contenant le module ou package que l'on souhaite importer
import sys
sys.path.append(chemin)
```

4 Tests unitaires avec pytest

Il est possible d'exécuter tous les tests unitaires dans un package ou un module à l'aide du programme `pytest`. La commande `pytest fichier.py` exécute toutes les fonctions commençant par `test_` qui sont définies à l'intérieur du fichier `fichier.py`. Ces fonctions doivent être des fonctions de tests unitaires utilisant la fonction `assert` (selon le modèle vu en cours).

La commande `pytest` considère tous les fichiers à l'intérieur du répertoire courant (et des sous-répertoires) commençant par `test_`. Pour chacun de ces fichiers, le programme exécute toutes les fonctions commençant par `test_`.