Modularité

Langages et programmation

1 Pourquoi la modularité ?

Lorsqu'un programme dépasse quelques dizaines de lignes, il devient difficile à lire, à maintenir et à faire évoluer si tout le code est dans un seul fichier. La **modularité** consiste à découper un programme en **modules** : des fichiers indépendants, chacun responsable d'une tâche bien définie.

Les avantages sont multiples :

— la **lisibilité** : chaque module a un rôle clair ;

— la **réutilisabilité** : un module bien conçu peut être utilisé dans d'autres projets ;

— le **travail collaboratif** : plusieurs développeurs peuvent travailler sur des modules différents ;

— la **testabilité** : on peut tester chaque module indépendamment.

2 Créer un module en Python

Un module Python est simplement un fichier `.py` contenant des définitions de fonctions, de classes ou de variables.

Exemple. Créons un fichier `geometrie.py` :

```
"""Module de calculs geometriques simples."""

import math

def aire_cercle(rayon):
    """Renvoie l'aire d'un cercle de rayon donne.

    >>> aire_cercle(1)
    3.141592653589793
    >>> round(aire_cercle(5), 2)
    78.54
    """
    return math.pi * rayon ** 2

def perimetre_cercle(rayon):
    """Renvoie le perimetre d'un cercle de rayon donne.

    >>> round(perimetre_cercle(1), 2)
    6.28
    """
    return 2 * math.pi * rayon

def aire_rectangle(longueur, largeur):
    """Renvoie l'aire d'un rectangle.
```

```
    >>> aire_rectangle(3, 4)
    12
    """
    return longueur * largeur
```

Ce fichier est un module. On peut l'utiliser depuis un autre fichier Python situé dans le même répertoire :

```
import geometrie

print(geometrie.aire_cercle(10))
print(geometrie.perimetre_cercle(10))
```

On peut aussi n'importer que certaines fonctions :

```
from geometrie import aire_cercle, aire_rectangle

print(aire_cercle(5))
print(aire_rectangle(3, 7))
```

③ Documenter un module

Un module bien documenté commence par une **docstring** (chaîne de documentation) qui décrit son rôle. Chaque fonction publique possède également sa propre docstring, qui précise ce qu'elle fait, ses paramètres et sa valeur de retour.

3.1 La fonction `help()`

La fonction `help()` de Python permet d'afficher la documentation d'un module ou d'une fonction :

```
import geometrie
help(geometrie.aire_cercle)
```

```
Help on function aire_cercle in module geometrie:

aire_cercle(rayon)
    Renvoie l'aire d'un cercle de rayon donné.

    >>> aire_cercle(1)
    3.141592653589793
```

3.2 Tester avec `doctest`

Les docstrings contenant des exemples au format `>>>` peuvent être testées automatiquement avec le module `doctest` :

```
import doctest
import geometrie

```

```
doctest.testmod(geometrie)
```

Si tous les exemples produisent le résultat attendu, aucune erreur n'est affichée. Sinon, `doctest` signale les écarts.

3.3 Exécuter ou importer un module

Un module peut aussi servir de programme. On place alors à la fin du fichier un bloc qui ne s'exécute que si le fichier est lancé directement :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Python crée pour chaque module une variable spéciale, `__name__`. Quand on lance le fichier lui-même (`python3 geometrie.py`), elle vaut `"__main__"` : la condition est vraie et les exemples sont vérifiés. Quand un autre programme importe le module (`import geometrie`), elle vaut le nom du module, `"geometrie"` : le bloc est ignoré, et le module fournit ses fonctions sans rien lancer de lui-même.

④ Utiliser une API ou une bibliothèque

Une **API** (*application programming interface*, interface de programmation) est l'ensemble des fonctions, classes et constantes qu'un module met à disposition de ses utilisateurs, avec leur spécification. C'est l'interface publique du module : on l'utilise sans connaître son implémentation, et c'est elle que décrivent `help()` et la documentation officielle.

Le code qui réalise ces fonctions s'appelle l'**implémentation** : il peut changer sans gêner les utilisateurs du module, tant que l'interface reste la même. Par convention, un nom qui commence par un tiret bas désigne une fonction interne, qui ne fait pas partie de l'API :

```
def _carre(x):
    return x * x

def aire_cercle(rayon):
    """Renvoie l'aire d'un cercle."""
    return math.pi * _carre(rayon)
```

En Python, de nombreuses bibliothèques fournissent des fonctionnalités prêtes à l'emploi. Pour les utiliser efficacement, il faut savoir **exploiter leur documentation**.

Exemple. Explorons le module `random` de la bibliothèque standard :

```
import random
help(random.choice)
```

```
choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

La documentation indique que `choice` prend en paramètre une séquence non vide et renvoie un élément choisi au hasard.

Exercice 1 *Explorer la documentation* En utilisant uniquement la fonction `help()` et la documentation officielle de Python, écrire un programme qui :

1. génère une liste de 20 entiers aléatoires entre 1 et 100 (utiliser `random.randint`) ;
2. trie cette liste par ordre croissant sans utiliser la méthode `sort` (chercher dans la documentation de `sorted`) ;
3. affiche la médiane de cette liste (chercher dans le module `statistics`).

5 Structurer un projet en modules

Pour un projet plus conséquent, on organise les fichiers en **paquets** (*packages*). Un paquet est un répertoire contenant un fichier `__init__.py` (qui peut être vide) et plusieurs modules.

Exemple. Structure d'un projet de jeu :

```
mon_jeu/
├── __init__.py
├── personnage.py
├── inventaire.py
├── combat.py
└── affichage.py
```

Chaque fichier s'occupe d'un aspect du jeu. On peut alors écrire :

```
from mon_jeu.personnage import Guerrier
from mon_jeu.inventaire import Inventaire

heros = Guerrier("Arthur", points_de_vie=100)
sac = Inventaire()
sac.ajouter("epee")
```

6 À retenir

Notion	Description
Module	Fichier Python contenant des définitions réutilisables
Import	`import module` ou `from module import fonction`
Docstring	Chaîne de documentation décrivant le rôle d'une fonction ou d'un module
`help()`	Affiche la documentation d'un module ou d'une fonction
`doctest`	Teste automatiquement les exemples contenus dans les docstrings
`__name__`	Vaut `"__main__"` si le fichier est lancé directement, le nom du module s'il est importé
API	Interface publique d'un module, utilisée sans connaître son implémentation
Paquet	Répertoire de modules, identifié par `__init__.py`

Exercice 2 *Créer un module*

1. Créer un fichier `conversions.py` contenant trois fonctions documentées :

— `celsius_vers_fahrenheit(c)` qui renvoie la température en Fahrenheit ;

— `fahrenheit_vers_celsius(f)` qui renvoie la température en Celsius ;

— `km_vers_miles(km)` qui convertit des kilomètres en miles (1 km ≈ 0,621 miles).

2. Chaque fonction doit comporter au moins un exemple `>>>` dans sa docstring.
3. Dans un fichier `test_conversions.py`, importer le module et vérifier les docstrings avec `doctest`.