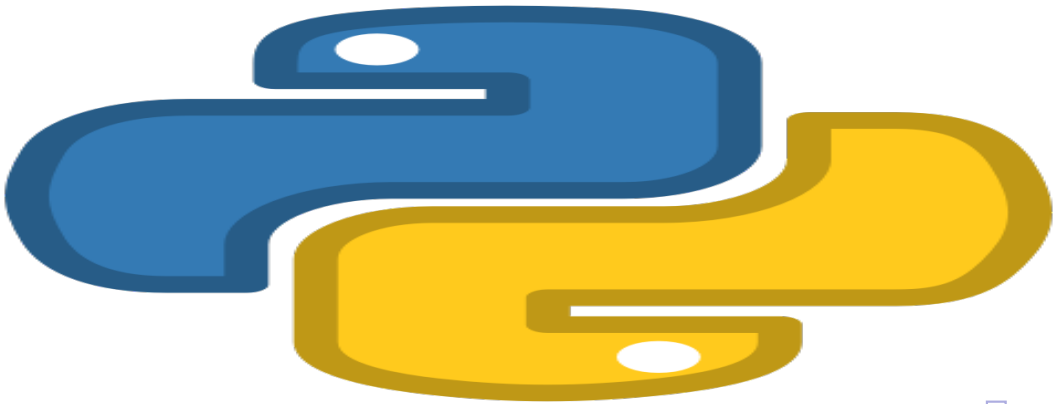
Python : modules et packages

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

Python

Dans une application **Python**

- On peut utiliser des éléments définis dans un autre fichier : une variable, une fonction, une classe, une interface...
- Pour cela, il faut l'importer là où on a besoin de l'utiliser
- Un fichier contenant du code **Python** = `module`

Quatre types de modules

- **Built-in modules** :
  - Définis dans la librairie standard de **Python** : **STDLIB**.
  - Pour les utiliser, ils ne nécessitent aucune importation.
- **Core modules** :
  - Définis dans **STDLIB**.
  - Pour les utiliser, il faut les importer.
- **Community modules** :
  - Proposés par la communauté **Python**.
  - Pour les utiliser, il faut les installer et ensuite les importer.
- **Custom modules** :
  - Définis par le développeur.
  - Pour les utiliser, il faut les importer.

Python

Avant de commencer

- Créez un nouveau projet `python_modules` dans votre espace de travail
- Créez un fichier `main.py`
- Validez

Python

Étant donné le fichier `fonctions.py` ayant le contenu suivant

```
somme = lambda a, b: a + b

produit = lambda a, b: a * b
```

Python

Étant donné le fichier `fonctions.py` ayant le contenu suivant

```
somme = lambda a, b: a + b

produit = lambda a, b: a * b
```

Pour importer le contenu de `fonctions.py` dans `main.py`

```
import fonctions

print(fonctions.somme (2, 3))
# affiche 5

print(fonctions.produit (2, 3))
# affiche 6
```

Python

On peut aussi utiliser des alias

```
import fonctions as f

print(f.somme (2, 3))
# affiche 5

print(f.produit (2, 3))
# affiche 6
```

Python

On peut aussi indiquer ce que l'on souhaite importer et simplifier l'utilisation

```
from fonctions import somme, produit

print(somme (2, 3))
# affiche 5

print(produit (2, 3))
# affiche 6
```

Python

On peut aussi utiliser les alias pour les fonctions

```
from fonctions import somme as s, produit as p

print(s (2, 3))
# affiche 5

print(p (2, 3))
# affiche 6
```

Python

Pour tout importer, on peut utiliser *

```
from fonctions import *

print(somme (2, 3))
# affiche 5

print(produit (2, 3))
# affiche 6
```

Python

Python : interprété, compilé ou les deux ?

- **Python** est un langage interprété : le code source est généralement exécuté ligne par ligne par l'interpréteur **Python**
- Cependant, lors de l'exécution d'un script contenant des importations de modules, **Python** compile le code source en bytecode, qui est ensuite exécuté par la machine virtuelle **Python** (**PVM**).
- Le bytecode est stocké dans des fichiers `.pyc` dans le dossier `__pycache__` pour optimiser les exécutions ultérieures.
- Dans certains cas, comme l'exécution directe de scripts simples, **Python** peut choisir de ne pas générer ces fichiers `.pyc` pour éviter l'encombrement du système de fichiers

Python

Packages

- Un répertoire contenant des fichiers et/ou répertoires est un package.
- Avant **Python 3.3**, un package contenant des modules **Python** devait contenir un fichier `__init__.py`, indiquant à **Python** qu'il s'agissait d'un package.
- Ce fichier est désormais facultatif, mais reste utile pour initialiser des variables de package.

Python

Packages

- Un répertoire contenant des fichiers et/ou répertoires est un package.
- Avant **Python 3.3**, un package contenant des modules **Python** devait contenir un fichier `__init__.py`, indiquant à **Python** qu'il s'agissait d'un package.
- Ce fichier est désormais facultatif, mais reste utile pour initialiser des variables de package.

Pour la suite

- créons un répertoire appelé `package` à la racine du projet.
- déplaçons le fichier `fonctions.py` dans `package`.

Python

Dans `main.py`, pour importer et utiliser les fonctions définies dans `fonctions.py`

```
from package.fonctions import produit, somme

print(somme (2, 3))
# affiche 5

print(produit (2, 3))
# affiche 6
```

Python

Et si `fonctions.py` **était dans** `subpackage` **qui est défini dans** `package`

```
from package.subpackage.fonctions import produit, somme

print(somme (2, 3))
# affiche 5

print(produit (2, 3))
# affiche 6
```

Python

Constantes pour modules et packages

- `__name__` contient
  - soit la valeur `__main__` dans le fichier d'entrée
  - soit le chemin complet depuis la racine du projet vers le fichier source
- `__package__` contient le package et les sous-packages du fichier source
- `__file__` contient le chemin complet depuis la racine du disque vers le fichier source

Python

En plaçant les constantes précédentes dans `main.py`, le résultat est

```
print(__name__)
# affiche __main__

print(__package__)
# affiche None

print(__file__)
# affiche C:/Users/user/cours-modules/main.py
```

Python

En les plaçant dans `fonctions.py`, le résultat est

```
print(__name__)
# affiche package.subpackage.fonctions

print(__package__)
# affiche package.subpackage

print(__file__)
# affiche C:\Users\user\cours-modules\package\subpackage\fonctions.py
```

Python

Dans `fonctions.py`, ajoutons la fonction `somme_carre`

```
somme = lambda a, b: a + b

produit = lambda a, b: a * b

somme_carre = lambda a, b: somme(a * a, b * b)

print(somme_carre(3, 4))
```

Python

Dans `fonctions.py`, ajoutons la fonction `somme_carre`

```
somme = lambda a, b: a + b

produit = lambda a, b: a * b

somme_carre = lambda a, b: somme(a * a, b * b)

print(somme_carre(3, 4))
```


En exécutant `fonctions.py`, le résultat est :

25

Python

En exécutant `main.py`

```
from package.subpackage.fonctions import *

print(somme (2, 3))

print(produit (2, 3))
```

Python

En exécutant `main.py`

```
from package.subpackage.fonctions import *

print(somme (2, 3))

print(produit (2, 3))
```


Le résultat est :

25
5
6

Python

Modifions `fonctions.py` **pour ne plus exécuter** `somme_carre` **que si on exécute** `fonctions.py`

```
somme = lambda a, b: a + b

produit = lambda a, b: a * b

somme_carre = lambda a, b: somme(a * a, b * b)

if __name__ == '__main__':
    print(somme_carre(3, 4))
```

Python

En exécutant `fonctions.py`, le résultat est :

25

Python

En exécutant `fonctions.py`, le résultat est :

25

En exécutant `main.py`, le résultat est :

5
6

Python

Import relatif en **Python**

- Commence par `.` (répertoire actuel) ou `..` (répertoire parent) pour naviguer dans la structure du package.
- Autorisé uniquement si le module n'est pas exécuté comme script principal : ayant un `__name__` différent de `__main__`

Python

Dans `package/subpackage`**, considérons le module** `avancees.py`

```
def somme_carree(a, b):
    return somme(a*a, b*b)
```

Python

Dans `package/subpackage`, considérons le module `avancees.py`

```
def somme_carree(a, b):
    return somme(a*a, b*b)
```

La fonction `somme` peut être importer avec un chemin absolu

```
from package.subpackage.fonctions import somme


def somme_carree(a, b):
    return somme(a*a, b*b)
```

Python

Dans `package/subpackage`**, considérons le module** `avancees.py`

```
def somme_carree(a, b):
    return somme(a*a, b*b)
```

La fonction `somme` **peut être importer avec un chemin absolu**

```
from package.subpackage.fonctions import somme


def somme_carree(a, b):
    return somme(a*a, b*b)
```

Pour tester dans `main.py`

```
from package.subpackage.avancees import somme_carree

print(somme_carree(2, 3))
# affiche 13
```

Python

La fonction `somme` peut être importer avec un chemin relatif

```
from .fonctions import somme


def somme_carree(a, b):
    return somme(a*a, b*b)
```

Python

La fonction `somme` peut être importer avec un chemin relatif

```
from .fonctions import somme


def somme_carree(a, b):
    return somme(a*a, b*b)
```


`main.py` **reste inchangé**

```
from package.subpackage.avancees import somme_carree

print(somme_carree(2, 3))
# affiche 13
```

Python

Deuxième import relatif possible

```
from . import fonctions


def somme_carree(a, b):
    return fonctions.somme(a*a, b*b)
```

Python

Deuxième import relatif possible

```
from . import fonctions


def somme_carree(a, b):
    return fonctions.somme(a*a, b*b)
```

`main.py` **reste inchangé**

```
from package.subpackage.avancees import somme_carree

print(somme_carree(2, 3))
# affiche 13
```

Python

Remarque

Déplaçons `fonctions.py` dans un package `src/utils`.

Python

Pour utiliser `somme` avec un import relatif

```
from ..utils import fonctions


def somme_carree(a, b):
    return fonctions.somme(a*a, b*b)
```

Python

Pour utiliser `somme` avec un import relatif

```
from ..utils import fonctions


def somme_carree(a, b):
    return fonctions.somme(a*a, b*b)
```

`main.py` **reste inchangé**

```
from package.subpackage.avancees import somme_carree

print(somme_carree(2, 3))
# affiche 13
```

Python

Dans `package/subpackage/__init__.py`**, ajoutons l'**`import` **suivant**

```
from .avancees import somme_carree
```

Python

Dans `package/subpackage/__init__.py`**, ajoutons l'**`import` **suivant**

```
from .avancees import somme_carree
```

Ainsi, l'`import` **dans** `main.py` **peut être simplifié**

```
from package.subpackage import somme_carree

print(somme_carree(2, 3))
# affiche 13
```

Python

Modifions `package/subpackage/__init__.py` **en ajoutant** `__all__`

```
from .avancees import somme_carree

__all__ = ["somme_carree"]
```

Python

Modifions `package/subpackage/__init__.py` **en ajoutant** `__all__`

```
from .avancees import somme_carree

__all__ = ["somme_carree"]
```

Ainsi, nous pouvons utiliser l'`import *`

```
from package.subpackage import *

print(somme_carree(2, 3))
# affiche 13
```

Python

Dans `package/__init__.py`, ajoutons le code suivant

```
from .subpackage import somme_carree

__all__ = ["somme_carree"]
```

Python

Dans `package/__init__.py`**, ajoutons le code suivant**

```
from .subpackage import somme_carree

__all__ = ["somme_carree"]
```

Ainsi, l'`import` **dans** `main.py` **peut être simplifié encore plus**

```
from package import *

print(somme_carree(2, 3))
# affiche 13
```

Python

Bonnes pratiques d'organisation d'un projet **Python**

- Un seul module d'entrée à la racine du projet, souvent nommé `main.py`, pour centraliser le lancement du projet.
- Un package, par exemple `src`, à la racine du projet pour contenir le code source, organisé en sous-packages.
- Un fichier `requirements.txt` pour les dépendances du projet.
- Un fichier `README.md` pour documenter le projet et ses instructions d'utilisation.
- Un répertoire `tests` pour regrouper les tests unitaires.