La gestion des modules

Alexandre Roulois (Université Paris Cité, LLF, CNRS)

Table of contents

Module ou *package* ?

Module : fichier qui regroupe plusieurs fonctions.

Package : répertoire qui rassemble plusieurs modules.

Dans le précédent TD, vous avez déplacé une fonction dans un fichier séparé intitulé *utils.py*. Par cette simple opération, vous avez créé un module qu'il est possible d'importer dans un script avec la commande `import` plus le nom du fichier sans l'extension. Afin de tester l'opération :

1. créez un nouveau script ;
2. importez le module `utils` avec l'instruction `import utils` ;
3. récupérez le répertoire *scrape* sur Moodle ;
4. exécutez le script.

Comment ça ?! `ModuleNotFoundError` ?

Python a raison, il s'attendait à trouver un fichier *utils.py* au même niveau que votre script, c'est-à-dire dans le même répertoire, alors qu'il est rangé dans un sous-dossier *scrape*.

Ce sous-dossier *scrape* est ce que l'on appelle un *package*, et un *package* peut renfermer plusieurs modules.

Importations et espaces de noms

Avant tout, un *package* dispose d'un nom. Et c'est par ce nom qu'il sera appelé par python :

```
import scrape
```

On pourrait supposer que cette commande donne accès à tous les modules qui dépendent de `scrape` mais, en l'état, aucune possibilité d'accéder à la fonction `get_html_from_url()` définie dans `utils`. Toutes les commandes ci-dessous échouent :

```
html = get_html_from_url('http://www.llf.cnrs.fr/')
html = utils.get_html_from_url('http://www.llf.cnrs.fr/')
html = scrape.utils.get_html_from_url('http://www.llf.cnrs.fr/')
```

Une solution consiste à importer directement le module `utils` du *package* `scrape` :

```
import scrape.utils
```

Puis à appeler la fonction en passant par tous les espaces de noms :

```
html = scrape.utils.get_html_from_url('http://www.llf.cnrs.fr/')
```

Alias

Plutôt fastidieux d'appeler une fonction d'aussi loin, non ? Heureusement, il existe les alias :

```
import scrape.utils as sc

html = sc.get_html_from_url('http://www.llf.cnrs.fr/')
```

Importation complète

On peut aussi accéder directement aux fonctions sans les espaces de noms :

```
from scrape.utils import *

html = get_html_from_url('http://www.llf.cnrs.fr/')
```

Importation spécifique

L'importation totale des fonctions d'un module présente deux risques :

- charger des fonctions inutiles pour le script en cours ;
- l'étoile * masque le nom des fonctions.

Une autre méthode consiste à puiser une fonction spécifique dans un module :

```
from scrape.utils import get_html_from_url

html = get_html_from_url('http://www.llf.cnrs.fr/')
```

Organiser son *package*

Dans la pratique, on préférera toujours conserver dans la syntaxe d'appel aux fonctions utilisateur le lien de dépendance à un espace de nommage (module) :

```
from scrape import utils

html = utils.get_html_from_url('http://www.llf.cnrs.fr/')
```

De cette manière, si l'on définissait dans un module une fonction qui porte le même nom qu'une autre fonction d'un autre module, on éviterait tout risque de collision.

Des fichiers thématiques

Un *package* s'organise autour de fichiers regroupant des fonctions qui servent elles-mêmes un objectif commun. On peut imaginer des modules avec des noms comme : `core`, `utils`, `tests`, `data`…

Un sommaire pour le *package*

Il existe un fichier un peu particulier que l'on a coutume de placer en-tête de tout *package*, et ce même s'il est devenu optionnel avec les mises à jour successives de python : ***init.py***

Ce fichier est utilisé à l'initialisation du paquet et peut se concevoir comme un sommaire du contenu. Il est chargé de lister l'ensemble des modules qu'il contient.

Créez un fichier `__init__.py` à la racine du répertoire `scrape` avec juste ces quelques lignes :

```
__name__ = 'Scrape: the Web-corpus maker'
__version__ = 'v1.0b'

from .utils import get_html_from_url
```

Ce qui permet ensuite de raccourcir les appels aux fonctions de tous les modules dépendants, comme s'il s'agissait de méthodes du *package* :

```
import scrape

html = scrape.get_html_from_url('http://www.llf.cnrs.fr/')
```

Les variables spéciales : l'exemple de `__name__`

Vous avez sans doute remarqué l'utilisation de variables encadrées par des doubles underscores : - `__name__` - `__version__`

Ce sont des variables spéciales qui permettent d'indiquer certaines informations à Python. Elles sont facultatives et ne servent que dans des cas particuliers.

Prenons l'exemple de la variable `__name__` que l'on trouve dans l'expression :

```
if __name__ == '__main__':
  # Main program
```

Lorsque Python lit un fichier *.py*, il réalise deux opérations :

- exécuter une instruction similaire à : `__name__ = '__main__'` ;
- exécuter le code du script.

D'un point du vue pratique, observez le fichier *utils.py* qui dispose d'un morceau de code :

```
if __name__ == '__main__':
  """Usage example.
  Works only when used as standalone.
  """
  url = "http://www.llf.cnrs.fr/ita/"
  html = get_html_from_url(url)
  anchors = parse_html_by_class(html, '.liste-membres .field-content a')

  for anchor in anchors:
    print(anchor.get_text())
```

Ce morceau de code n'est exécuté que lorsque le script *utils.py* est appelé tout seul (*standalone*). Téléchargez-le et lancez-le dans une console :

```
$ python utils.py
```

Vous verrez une liste de noms s'afficher.

En revanche, lorsque vous importez le module depuis un script, la liste de noms ne s'affiche pas !

```
import scrape.utils
```

L'explication est simple : lorsque le script est importé en tant que module, son `__name__` prend la valeur du fichier. En l'occurrence, le script *utils.py* a pour nom `utils` et pas `__main__`.

Le nom `__main__` est donc réservé au script principal.

Faites un test :

1. éditez le fichier *utils.py* ;
2. retirez l'instruction `if __name__ = '__main__':` ;
3. exécutez de nouveau le script.

Cette fois-ci, alors que l'importation du module s'effectue, le code qui sert d'exemple à l'utilisation du module s'exécute et la liste de noms apparaît…

La condition `if __name__ = '__main__':` est par conséquent impérative pour éviter tout effet de bord dans un script !