W3-S5-C1-packages

December 14, 2014

1 La notion de package

1.1 Complément - niveau basique

Dans ce complément, nous approfondissons la notion de module, qui a été introduite dans la vidéo, et nous décrivons la notion de *package* qui permet de créer des librairies plus structurées qu'avec un simple module.

Pour ce notebook nous aurons besoin de deux utilitaires pour voir le code correspondant aux modules et packages que nous manipulons

```
In []: from modtools import show_module, show_package
```

1.1.1 Rappel sur les modules

Nous avons vu dans la vidéo qu'on peut charger une librairie, lorsqu'elle se présente sous la forme d'un seul fichier source, au travers d'un objet python de type **module**.

Chargeons un module "jouet"

```
In []: import module_simple
```

Voyons à quoi ressemble ce module :

```
In []: show_module(module_simple)
```

Il est important de bien voir que le module joue le rôle d'**espace de nom**, dans le sens où :

```
In []: # on peut définir sans rique un variable globale 'spam'
       spam = 'eggs'
       # qui est indépendante de celle définie dans le module
       print "spam globale", spam
       print "spam du module", module_simple.spam
```

Pour résumer, un module est donc un objet python qui correspond à la fois à * un (seul) **fichier** sur le disque * un **espace de nom** pour les variables du programme

1.1.2 La notion de package

Lorsqu'il s'agit d'implémenter une très grosse librairie, il n'est pas concevable de tout concentrer en un seul fichier. C'est là qu'intervient la notion de **package**, qui est un peu aux **répertoires** ce que que le **module** est aux **fichiers**.

On importe un package exactement comme un module

```
In []: import package_jouet
```

Le package porte **le même nom** que le répertoire, c'est-à-dire que, de même que le module `module_jouet` correspond au fichier `module_jouet.py`, le package python `package_jouet` corrrespond au répertoire `package_jouet`.

Pour définir un package, il faut **obligatoirement** créer dans le répertoire (celui, donc, que l'on veut exposer à python), un fichier nommé `__init__.py`. Voilà comment a été implémenté le package que nous venons d'importer :

```
In []: show_package(package_jouet)
```

Comme on le voit, importer un package revient essentiellement à charger le fichier `__init__.py` correspondant. Le package se présente aussi comme un espace de nom, à présent on a une troisième variable `spam` qui est encore différente des deux autres :

```
In []: package_jouet.spam
```

L'avantage principal du package par rapport au module est qu'il peut contenir d'autres packages ou modules. Dans notre cas, package "jouet" vient avec un module qu'on peut importer en utilisant un attribut du package, c'est-à-dire comme ceci :

```
In []: import package_jouet.module_jouet
```

À nouveau regardons comment cela est implémenté; le fichier correspondant au module se trouve naturellement à l'intérieur du répertoire correspondant au package, c'était le but du jeu au départ :

```
In []: show_module(package_jouet.module_jouet)
```

1.1.3 À quoi sert `__init__.py` ?

Le code placé dans `__init__.py` est chargé d'initialiser la librairie. Le fichier **peut être vide** mais **doit absolument exister**. Nous vous mettons en garde car c'est une erreur fréquente de l'oublier. Sans lui vous ne pourrez importer ni le package, ni les modules ou sous-packages qu'il contient.

1.2 Complément - niveau avancé

1.2.1 Variables spéciales

Comme on le voit dans les exemples, certaines variables *spéciales* peuvent être lues ou écrites dans les modules ou packages. Voici les plus utilisées.

`__name__`

```
In []: print package_jouet.__name__, package_jouet.module_jouet.__name__
```

Remarquons à cet égard que le **point d'entrée** du programme (c'est-à-dire, on le rappelle, le fichier qui est passé directement à l'interpréteur python) est considéré comme un module dont l'attribut `__name__` vaut la chaîne `"__main__"`

C'est pourquoi (et c'est également expliqué ici) les scripts python se terminent généralement par une phrase du genre de

```
if __name__ == "__main__":
    <faire vraiment quelque chose>
```

`__file__`

```
In []: print package_jouet.__file__
       print package_jouet.module_jouet.__file__
```

`__all__` Il est possible de redéfinir dans un package la variable `__all__`, de façon à ne pas exposer certaines parties du package qui ont vocation à rester privées, comme c'est décrit ici.

1.2.2 Pour en savoir plus

Voir la section sur les modules dans la documentation python, et notamment la section sur les packages.