14 Création de modules

14.1 Création

Vous pouvez créez votres propres modules très simplement en Python. Il vous suffit d'écrire un ensemble de fonctions (et/ou de variables) dans un fichier, puis d'enregistrer celui-ci avec une extension `.py` (comme n'importe quel script Python). À titre d'exemple, voici le contenu du module `message.py`.

```
"""Module inutile qui affiche des messages :-)
"""

def Bonjour(nom):
    """Affiche bonjour !
    """
    return "bonjour " + nom

def Ciao(nom):
    """Affiche ciao !
    """
    return "ciao " + nom

def Hello(nom):
    """Affiche hello !
    """
    return "hello " + nom + " !"

date=16092008
```

14.2 Utilisation

Pour appeler une fonction ou une variable de ce module, il faut que le fichier `message.py` soit dans le répertoire courant (dans lequel on travaille) ou bien dans un répertoire indiqué par la variable d'environnement Unix `PYTHONPATH`. Ensuite, il suffit d'importer le module et toutes ses fonctions (et variables) vous sont alors accessibles.

La première fois qu'un module est importé, Python crée un fichier avec une extension `.pyc` (ici `message.pyc`) qui contient le bytecode (code précompilé) du module.

L'appel du module se fait avec la commande `import message`. Notez que le fichier est bien enregistré avec une extension `.py` et pourtant on ne la précise pas lorsqu'on importe le module. Ensuite on peut utiliser les fonctions comme avec un module classique.

```
>>> import message
>>> message.Hello("Joe")
'hello Joe !'
>>> message.Ciao("Bill")
'ciao Bill'
>>> message.Bonjour("Monsieur")
'bonjour Monsieur'
>>>  message.date
16092008
```

Les commentaires (entre triple guillemets) situés en début de module et sous chaque fonction permettent de fournir de l'aide invoquée ensuite par la commande `help()` :

```
>>> help(message)
NAME
    message - Module inutile qui affiche des messages :-)

FILE
    /home/cumin/poulain/message.py

FUNCTIONS
    Hello(nom)
        Affiche hello !

    Bonjour(nom)
        Affiche bonjour !

    Ciao(nom)
        Affiche ciao !

DATA
    date = 16092008
```

Remarques :
- Les fonctions dans un module peuvent s'appeler les unes les autres.
- Les fonctions dans un module peuvent appeler des fonctions situées dans un autre module s'il a été préalablement importé avec la commande `import`.

Vous voyez que les modules sont d'une simplicité enfantine à créer. Si vous avez des fonctions que vous serez amenés à utiliser souvent, n'hésitez plus !

14.3 Exercices

Conseil : pour cet exercice, écrivez un script dans un fichier, puis exécutez-le dans un *shell*.

1. Reprenez l'ensemble des fonctions qui gèrent le traitement de séquences nucléiques et incluez-les dans un module `adn.py`. Testez-les au fur et à mesure.

15 Autres modules d'intérêt

Nous allons voir dans cette section deux modules qui peuvent se révéler très pratiques. Le premier (`urllib2`) permet de télécharger n'importe quel fichier depuis internet. Le second (`pickle`) permet d'enregistrer des objets python pour pouvoir les retrouver tels quels plus tard. Ces deux modules sont présents par défaut dans n'importe quelle distribution Python.

15.1 Module urllib2

Les modules urllib et urllib2 permettent de rapatrier, directement avec Python, des fichiers depuis internet. Nous n'aborderons ici que des exemples simples, mais libre à vous d'approfondir si besoin. Il existe des différences subtiles entre `urllib` et `urllib2` que nous n'aborderons pas ici mais qui sont consultables sur la documentation officielle de Python. Dans ce cours, nous avons choisi de vous présenter `urllib2` car une des fonctions principales (`urlopen()`) est considérée comme obsolète dans `urllib` depuis la version 2.6 de Python.

Prenons un exemple simple, supposons que vous souhaitiez rapatrier le fichier PDB de la protéine barstar. En Unix vous utiliseriez probablement la commande `wget`, regardez comment faire en Python :

```
>>> import urllib2
>>> u = urllib2.urlopen("http://www.pdb.org/pdb/files/1BTA.pdb")
>>> pdb_lines = u.readlines()
>>> u.close()
>>> for line in pdb_lines:
...     print line,
...
HEADER    RIBONUCLEASE INHIBITOR                  09-MAY-94   1BTA
TITLE     THREE-DIMENSIONAL SOLUTION STRUCTURE AND 13C ASSIGNMENTS OF
TITLE    2 BARSTAR USING NUCLEAR MAGNETIC RESONANCE SPECTROSCOPY
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: BARSTAR;
COMPND   3 CHAIN: A;
COMPND   4 ENGINEERED: YES
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: BACILLUS AMYLOLIQUEFACIENS;
SOURCE   3 ORGANISM_TAXID: 1390
[...]
>>>
```

Comme vous le voyez, il suffit d'utiliser la fonction `urlopen()` et de lui passer en argument une URL (*i.e.* adresse internet) sous forme de chaîne de caractères. Dès lors, les méthodes associées aux fichiers (cf chapitre 6) sont accessibles (`read()`, `readline()`, `readlines()`, etc.).

Il est également possible d'enregistrer le fichier téléchargé :

```
>>> import urllib2
>>> u = urllib2.urlopen("http://www.pdb.org/pdb/files/1BTA.pdb")
>>> pdb_file_content = u.read()
>>> u.close()
>>> f = open("1BTA.pdb","w")
>>> f.write(pdb_file_content)
>>> f.close()
```

Dans ce dernier exemple, nous récupérons le fichier complet sous la forme d'une chaîne de caractères unique (avec la fonction `read()`). Le fichier est ensuite enregistré dans le répertoire courant (duquel vous avez lancé l'interpréteur Python). Nous vous laissons imaginer comment faire sur une série de plusieurs fichiers (cf. exercices).

15.2 Module pickle

Le module `pickle` permet d'effectuer une sérialisation et désérialisation des données. Il s'agit en fait d'encoder (d'une manière particulière) les objets que l'on peut créer en Python (variables, listes, dictionnaires, fonctions, classes, etc.) afin de les stocker dans un fichier, puis de les récupérer, sans devoir effectuer une étape de *parsing*.

15.2.1 Codage des données

Pour encoder des données avec `pickle` et les envoyer dans un fichier, on peut utiliser la fonction `dump(obj,file)` (où `file` est un fichier déjà ouvert) :

```
>>> import pickle
>>> fileout = open("mydata.dat","w")
>>> maliste = range(10)
>>> mondico = {'year': 2000, 'name': 'Far beyond driven', 'author': 'Pantera',
'style': 'Trash Metal'}
>>> pickle.dump(maliste,fileout)
>>> pickle.dump(mondico,fileout)
>>> pickle.dump(3.14,fileout)
>>> fileout.close()
```

Si on observe le fichier créé, on s'aperçoit que les données sont codées :

```
>>> import os
>>> os.system("cat mydata.dat")
(lp0
I0
aI1
aI2
aI3
aI4
aI5
aI6
aI7
aI8
aI9
a.(dp0
S'style'
p1
S'Trash Metal'
p2
sS'author'
p3
S'Pantera'
p4
sS'name'
p5
```

```
S'Far beyond driven'
p6
sS'year'
p7
I2000
s.F3.1400000000000001
.0
>>>
```

15.2.2 Décodage des données

Pour décoder les données, rien de plus simple, il suffit d'utiliser la fonction `load(file)` où `file` est un fichier ouvert en lecture :

```
>>> filein = open("mydata.dat")
>>> pickle.load(filein)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> pickle.load(filein)
{'style': 'Trash Metal', 'year': 2000, 'name': 'Far beyond driven',
'author': 'Pantera'}
>>> pickle.load(filein)
3.1400000000000001
```

Attention à ne pas utiliser la fonction `load` une fois de trop, sinon une erreur est générée :

```
>>> pickle.load(filein)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.5/pickle.py", line 1370, in load
    return Unpickler(file).load()
  File "/usr/lib/python2.5/pickle.py", line 858, in load
    dispatch[key](self)
  File "/usr/lib/python2.5/pickle.py", line 880, in load_eof
    raise EOFError
EOFError
>>>
```

Il est à noter qu'il existe un module équivalent mais beaucoup plus rapide, le module cpickle.

15.3 Exercices

1. Concevez un script qui rapatrie de manière automatique le fichier `1MSE.pdb` depuis la *Protein Data Bank* (à l'aide du module `urllib2`).
2. On souhaite récupérer les structure 3D de la partie N-terminale de la protéine HSP90 en présence de différents co-cristaux, les codes PDB sont `3K97`, `3K98` et `3K99`. Écrivez un script Python qui télécharge de manière automatique les fichiers PDB (à l'aide du module `urllib2` et d'une boucle) et les enregistre dans le répertoire courant.
3. Reprenez le dictionnaire des mots de 4 lettres du génome du chromosome I de la levure S. cerevisiae (*exercice 7 sur les dictionnaires et tuples*). Encodez le dictionnaire dans un fichier avec le module `pickle`. Quittez Python et regardez attentivement le fichier généré avec un éditeur de texte. Relancez Python, et décodez le dictionnaire.