

1 Les modules en Python

En ce point du cours, nous avons étudié la majorité des structures de données classiques (manquent les *arbres* et les *graphes* qui relèvent de cours plus avancés) et la quasi-totalité des structures de contrôle (le *traitement d'exceptions* et la *récurtivité* relèvent de cours plus avancés). On peut même affirmer que quel que soit le problème posé, s'il existe un programme qui le résoud alors nous avons vu tous les éléments pour le faire.

Il existe cependant des questions « standard » qu'il serait peu productif de résoudre chacun pour soi : ces questions sont tellement courantes que les programmes qui les résolvent sont stockés dans des *bibliothèques* (*libraries* en anglais). En Python, une telle bibliothèque est constituée de nombreux *modules* et un module est constitué de plusieurs fonctions (ou procédures) déjà programmées. Dans un module, on prend soin de regrouper les fonctions qui permettent de gérer un type de problème donné, bien souvent ce sont simplement les opérations associées à une structure de données adaptée au problème.

On peut même écrire soi-même des modules. Lorsque l'on veut écrire un gros logiciel, c'est généralement une bonne idée de le découper en modules (qui seront du même coup réutilisables par ailleurs), en suivant généralement le principe « au moins un module par nouvelle structure de donnée ». Il suffit alors d'importer les modules pour utiliser les fonctions qui y ont été programmées.

2 Gestion des fichiers : le module « os »

Pour utiliser un module, il faut d'abord le charger avec la commande `import`. Parmi les modules vraiment utiles, il y a celui qui permet de gérer l'arborescence des fichiers de l'ordinateur. Il se nomme `os`, comme « operating system », et nous nous servons de ce module comme exemple.

```
>>> import os
```

Remarquez que, exceptionnellement, `os` n'est pas écrit comme une chaîne de caractère (donc entre guillemets) et pourtant `import` ne considère pas `os` comme un nom de variable. . . Avant d'utiliser `os` pour gérer le système de fichiers, il faut savoir :

- qu'il peut y avoir plusieurs disques dur sur un ordinateur, ou qu'un disque dur peut être partagé en plusieurs partitions. Sous Mac ou un système Unix comme Linux cette séparation des disques et des partitions est gérée pour qu'on ait l'impression de n'avoir qu'un seul disque. Sous Windows, ces partitions sont vues comme plusieurs « lecteurs » différents numérotés A, B, C, D, etc. En général A et B sont présents pour des raisons historiques et sont réservés à deux lecteurs de disquettes. Le disque C est généralement celui sur lequel le système Windows est installé et les suivants sont des disques de données ou des lecteurs/graveurs de CD, DVD, etc.
- Un disque est organisé en arbre avec des répertoires (*directories* en anglais ou « dossiers » pour les utilisateurs naïfs) qui peuvent contenir des sous-arbres et des fichiers (*files*) qui contiennent les « vraies » données et sont nécessairement des feuilles de l'arbre.
- Un nœud de l'arbre est repéré par son chemin depuis la racine, qui est souvent appelé son adresse.
- Compte tenu de la taille de cet arbre et de la longueur des chemins, et du fait qu'on travaille longtemps sous un même répertoire, il est pratique d'avoir un répertoire dit « courant ». Cela permet de ne donner que les adresses à partir du répertoire courant.
- On note généralement « .. » le répertoire père du répertoire de travail, et « . » ce répertoire lui-même.

Pour utiliser une fonction ou une procédure d'un module, il faut la préfixer par le nom du module (une fois qu'il a été importé). Par exemple `getcwd` est une fonction qui retourne l'adresse du répertoire dans lequel on est en train de travailler (« `cwd` » pour *current working directory*).

```
>>> os.getcwd()  
'/home/bernot'
```

Le module `os` offre aussi une procédure qui crée un répertoire, appelée `mkdir` (*make directory*) :

```
>>> os.mkdir("test")
```

Pour se déplacer dans les répertoires, et donc changer l'adresse du répertoire de travail, on utilise la procédure `chdir` (*change directory*) et l'on peut lister le contenu d'un répertoire avec la fonction `listdir`, qui retourne la liste des éléments contenus dans le répertoire :

```

>>> os.chdir("test")
>>> os.listdir("/home/bernot/test")
[]
>>> f=open("toto.txt")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'toto.txt'
>>> f=open("toto.txt","w")
>>> f.close()
>>> os.listdir("/home/bernot/test")
['toto.txt']

```

Il existe des abréviations : la chaîne "." remplace le répertoire de travail courant et la chaîne ".." désigne son répertoire parent :

```

>>> os.listdir(".")
['toto.txt']
>>> os.mkdir("repfiles")
>>> os.listdir(".")
['repfiles', 'toto.txt']
>>> os.chdir("repfiles")
>>> os.getcwd()
'/home/bernot/test/repfiles'
>>> os.listdir(".")
[]
>>> os.listdir("..")
['repfiles', 'toto.txt']

```

D'autres fonctions ou procédures utiles pour la gestion du système de fichiers sont :

- `rename` prend en argument les chemins source et cible : peut déplacer dans l'arbre, aussi bien un fichier qu'un sous-arbre complet
- `remove` prend en argument le chemin d'un fichier et le supprime
- `rmdir` comme `remove`, mais pour un répertoire vide

On remarque que la liste obtenue par `listdir` mélange des répertoires et les « vrais » fichiers sans moyen de les distinguer. Le module `os` contient un « sous-module » `os.path` qui est automatiquement importé avec `os` et fournit d'autres fonctions bien utiles :

- `isfile` dit si le chemin pointe sur un fichier standard
- `isdir` ... si c'est un répertoire
- `islink` ... si c'est un lien symbolique (« raccourci » sous windows)
- `exists` dit si le chemin donné en argument existe (fonction booléenne)
- `getsize` taille en octets du fichier désigné par le chemin donné en argument
- `abspath` prend un chemin en entrée et retourne sa version en adresse absolue.
- `basename` fournit le dernier nom du chemin
- `dirname` le contraire

```

>>> os.isfile("toto.txt")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute 'isfile'
>>> os.path.isfile("toto.txt")
True
>>> os.path.isdir("toto.txt")
False

```