1 Lecture de fichiers

Un fichier est une suite de caractères mémorisés sur le disque durde la machine dans un endroit caractérisé par un « nom de fichier ». Les contraintes techniques font qu'on doit charger du disque dur vers la mémoire ces caractères les uns après les autres pour pouvoir travailler dessus. Ainsi, en programmation, on gère un fichier comme un « flux de caractères » sur lequel on peut « ouvrir un robinet » pour charger un certain nombre de caractères, en partant du début du fichier. Après usage, il faut « fermer le robinet ».

Il y a beaucoup de fichiers sur un disque dur, donc beaucoup de robinets potentiels. L'ensemble de tous ces robinets potentiels constitue le type de données appelé `file`.

Les opérations qui travaillent sur le type `file` sont nombreuses. La première de toutes consiste à « ouvrir un fichier », ce qui met le robinet correspondant à disposition du programme qui a ouvert le fichier. La fonction `open` prend en argument un nom de fichier (qui est une chaîne de caractères) et retourne un objet de type `file` (le robinet).

Par la suite, le programme disposera du « robinet », qu'il pourra ouvrir à chaque instant pour récupérer des quantités déterminées de caractères, ceci avec la « fonction » `read` qui prend en argument le nombre de caractères qu'on veut lire. La première ouverture de robinet « lit » les premiers caractères du fichiers ; l'ouverture suivante, toujours avec `read` reprendra à partir du premier caractère pas encore lu du fichier, et ainsi de suite.

Après utilisation du fichier, il est *très important* de « rendre le robinet au système » pour que d'autres programmes puissent l'utiliser ou le modifier à leur tour. La fonction `close` assure cette tâche en rendant le robinet de nouveau inaccessible au programme. Imaginons un fichier sur le disque, appelé « toto.txt », qui contiendrait la suite de caractères « `TitiTrucMachinChouette` ». On peut par exemple écrire :

```
>>> rob = open("toto.txt")
>>> w = rob.read(4)
>>> x = rob.read(4)
>>> y = rob.read(6)
>>> z = rob.read(8)
>>> rob.close()
>>> w
'Titi'
>>> x
'Truc'
>>> y
'Machin'
>>> z
'Chouette'
```

On observe une syntaxe nouvelle : au lieu d'écrire `read(rob,4)` ou `close(rob)`, on écrit `rob.read(4)` et `rob.close()`. Ceci est dû au fait que le robinet `rob` n'est pas une donnée comme celles qu'on a vues jusqu'à maintenant : c'est un *objet*.

Un objet en informatique est une entité qui peut « agir » dans un programme de plusieurs manières différentes en fonction de l'*état* dans lequel il est. Par exemple, après `open`, l'état de l'objet `rob` est d'être en début de fichier. Ainsi la première lecture `rob.read(4)` vaut `"Titi"` et change *de manière implicite* l'état de `rob` : maintenant le robinet en est au cinquième caractère, et on le voit parce que le prochain appel de `rob.read(4)` ne fournit plus `Titi` mais `Truc`.

Les opérations qui travaillent spécifiquement sur des *objets* en informatique sont appelées des *méthodes* et s'utilisent avec cette notation « pointée » qui indique qu'une méthode ne prend pas seulement sont objet en argument mais peut aussi changer son état interne.

`close` est finalement la méthode « d'apoptose du robinet »... et change radicalement l'état de son objet en le faisant disparaître (le robinet disparaît du programme mais le fichier reste parfaitement visible dans le disque dur).

2 Quelques méthodes utiles de lecture de fichiers

`readline()` : cette méthode lit autant de caractères que nécessaire pour aller jusqu'en fin de ligne (le retour-chariot inclus). Par exemple si `gamin.txt` contient les 4 lignes

```
premier doigt
deuxième doigt
second doigt
troisième doigt
quatrième doigt
```

alors on peut programmer :

```
>>> def affiche (f) :
...     fich = open(f)
...     i = 1
...     ligne = fich.readline()
...     while ligne != "" :
...         print("%i : %s" % (i,ligne))
...         i = i + 1
...         ligne = fich.readline()
...     fich.close()
...
>>> affiche("gamin.txt")
1 : premier doigt

2 : deuxième doigt

3 : second doigt

4 : troisième doigt

5 : quatrième doigt
```

Lorsque l'objet fichier arrive en fin de fichier (plus rien à lire), la méthode `readline` retourne une chaîne vide. La procédure précédente s'arrête donc à la fin du fichier.

Remarquons aussi que les trois premières lignes sont suivies d'une ligne vide : c'est dû au fait que `readline` inclût le retour-chariot(c'est-à-dire le passage à la ligne : la touche « Entrée » du clavier) *et* que `print` en écrit également un de sorte qu'il y a 2 passages à la ligne successifs. Pour éviter ce double interligne, il suffit de ne pas écrire la ligne toute entière mais seulement jusqu'à son avant-dernier caractère :

```
...         print("%i : %s" % (i,ligne[0:len(l)-1]))
```

et comme, d'une part, on peut rendre implicite le début ou la fin d'une chaîne de caractères dans la définition d'une sous-chaîne, et d'autre part, un indice négatif signifie « en partant de la fin », on peut simplifier encore :

```
...         print("%i : %s" % (i,ligne[:-1]))
```

Remarquons également que, dans la fonction `affiche`, si le fichier `f` contient une ligne vide au milieu des autres lignes, on pourrait croîre que cette ligne vide fasse sortir du `while` avant la fin du fichier. Il n'en est rien car pour une « ligne vide » la chaîne `s` n'est pas vide : elle contient le retour-chariot.

Enfin, précisons aussi que si le fichier ne se termine pas par un retour-chariot, alors le dernier `readline` fournit les caractères qui restent jusqu'à la fin de fichier, sans ajouter de retour-chariot final.

3 Ecriture de fichiers

Il est possible « d'entrer un flux contraire dans un robinet » c'est-à-dire d'écrire dans un fichier. Il faut alors ouvrir le fichier non plus en lecture mais en écriture :
`open(nom,'w')` est une fonction qui crée un fichier de ce `nom`. Si un fichier du même `nom` existait, il est écrasé (vidé) par `open`.

Ensuite, il suffit d'utiliser la méthode `write` qui prend en argument une chaine de caractères et a pour effet de l'écrire dans le fichier.

```
>>> f = open("nouveau.txt","w")
>>> f.write("Toto")
>>> f.write("Tutu")
>>> f.write("Glop\n")
>>> f.write("ligne numero 2 seulement\n")
>>> f.close()
```

Le fichier contient alors :

```
TotoTutuGlop
ligne numero 2 seulement
```

et l'on remarque qu'il faut explicitement écrire, avec `write`, les retour-chariots sous la forme « `\n` ».

Ainsi la fonction `open` peut prendre 2 arguments. Le deuxième argument peut être
- `"r"` (comme *read*) : c'est l'argument par défaut, voir les sections précédentes.
- `"w"` (comme *write*) : si le fichier à l'adresse du premier argument n'existe pas alors il est créé ; s'il existe déjà alors il est entièrement écrasé.
- `"a"` (comme *append*) : si le fichier à l'adresse du premier argument n'existe pas alors il est créé ; s'il existe déjà alors les caractères sont ajoutés à la fin du fichier.

4 Un point technique : « Operating System »

En TD, il est parfois difficile de retrouver un fichier qu'on a créé en Python car on ne sait pas dans quel répertoire par défaut Python crée ses fichiers. Comme on le verra plus complètement au cours suivant, on peut lancer les commandes successives suivantes pour connaître ce répertoire (directory en anglais) ou pour en changer :

```
>>> import os
>>> os.getcwd()
'/home/bernot/python'
>>> os.listdir(".")
['chromosome.txt','test.py']
>>> os.chdir("/home/bernot")
>>> os.getcwd()
'/home/bernot'
```

« os » signifie « operating system » et « cwd » signifie « current working directory »
- `os.getcwd()` fournit le répertoire où se trouve le programme python actuellement,
- `os.listdir()` fournit la liste des fichiers et répertoires contenus dans le répertoire courant (`dir` comme *directory*),
- `os.chdir("...")` change le répertoire où se trouve le programme python.