1 Lecture de fichiers

Un fichier est une suite de caractères mémorisés sur le disque dur de 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 « réagir » à 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 se veut rappeler qu'une méthode ne prend pas seulement sont objet en argument mais peut aussi changer son état.

`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
```

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
```

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.
À ce propos, 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 retour-chariot final.

`tell` : cette méthode retourne la position du robinet, c'est-à-dire le nombre de caractères déjà lus.

```
>>> r = open("toto.txt")
>>> r.read(4)
'Titi'
>>> r.read(4)
'Truc'
>>> r.tell()
8L
>>> r.close()
```

Le « `L` » signifie « entier `long` » car la taille d'un fichier peut être vraiment très grande et python encode la taille dans ce type, qui se manipule exactement comme les entiers de type `int`.

Ajoutons qu'on peut faire un `for` sur un fichier : cela énumère les lignes du fichier :

```
>>> f = open("gamin.txt")
>>> for ligne in f :
```

```
...     print(len(ligne))
...
14
15
13
15
>>> f.close()
```

On peut par exemple écrire d'une procédure `more(fichier)` qui affiche à l'écran le contenu du fichier, par pages de 24 lignes :

```
>>> def more(f):
...   hauteur=24
...   r=open(f)
...   vues=0
...   for l in r:
...     print(l[:-1])   # enlève le passage à la ligne de l
...     if vues < hauteur:
...       vues=vues+1
...     else:
...       s=input("suite... ")
...       vues=0
...   r.close
```

La dernière ligne, si elle ne se termine pas par un retour chariot, voit son dernier caractère disparaître. Pour bien faire, il faudrait tester si la dernière ligne contient une fin de ligne (`"\n"`) ou non.

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 remis à zéro (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.

On peut par exemple écrire une procédure `merge` qui prend 3 arguments de type chaîne de caractères (`f1`, `f2` et `f3`) contenant des noms de fichiers : en supposant que les fichiers `f1` et `f2` contiennent des lignes triées par ordre alphabétique (selon l'ordre `ascii`), la procédure `merge` crée le fichier `f3` contenant l'union des lignes de `f1` et `f2`, également triées.

```
>>> def merge(f1,f2,f3) :
...   a = open(f1)
...   b = open(f2)
...   c = open(f3,"w")
...   x = a.readline()
...   y = b.readline()
...   while x != "" and y != "" :  # une ligne à voir dans chaque fichier
...     if x < y :
...       c.write(x)               # on écrit la plus petite des 2 lignes
...       x = a.readline()         # et on lit la suite du fichier utilisé
...     else :
...       c.write(y)               # idem
...       y = b.readline()
...   while x != "" :              # on termine le fichier non épuisé,
...     c.write(x)
...     x = a.readline()
...   while y != "" :              # un seul des 2 derniers while est exécuté
...     c.write(y)
...     y = b.readline()
...   a.close()
...   b.close()
...   c.close()
...   print("Le fichier " + f3 + " est rempli.")
```

En supposant que le fichier `toto.txt` contienne

```
alain
marc
pierre
robert
yves
zorro
```

et que `titi.txt` contienne

```
bernard
joe
luc
thomas
```

la commande `merge("toto.txt","titi.txt","tutu.txt")` fabrique le fichier `tutu.txt` contenant :

```
alain
bernard
joe
luc
marc
pierre
robert
thomas
yves
zorro
```