Python : Fichiers

Ouverture

Pour ouvrir un fichier, on utilise la fonction `open`. La syntaxe est `open(nom, mode)`, où `nom` est le nom du fichier à ouvrir (s'il n'existe pas, il sera créé), et `mode` est le mode d'ouverture du fichier.

Attention : il faut donner l'adresse complète du fichier. Par exemple, si on veut ouvrir le fichier `README.txt` qui se trouve dans le dossier `C:\Users\Python`, on écrira :

```
fichier = open('C:\\Users\\Python\\README.txt')
```

Remarquer les `\\` nécessaires pour éviter que Python ne considère les `\U`, `\P`, etc. comme des caractères spéciaux.

Les différents modes d'ouverture d'un fichier sont :

- `'r'` (read - lecture seule),
- `'w'` (write - écriture seule),
- `'a'` (append - ce qui est ajouté au fichier l'est à la fin).

Le mode par défaut est `'r'`. Les fichiers sont ouverts en mode texte, sauf si on ajoute `'b'` au mode, auquel cas on ouvre le fichier en mode binaire.

Pour refermer le fichier on utilise la méthode `close`.

Lecture

Il y a différentes manières de lire le contenu d'un fichier texte.

La méthode `read` renvoie la chaîne de caractères constituant le fichier. On peut lui donner en argument le nombre de caractères à renvoyer.

```
>>> f = open('C:\\Users\\Python\\README.txt')
>>> t = f.read()            # t est la chaîne contenue dans le fichier
>>> f.close()               # on referme le fichier
>>> t[:100]                 # les 100 premiers caractères de t
'This is Python version 3.3.0\n============================\n\nCopyright (c)
2001, 2002, 2003, 2004, 200'
```

La méthode `readlines` renvoie une liste dont les éléments sont les lignes du fichier (qui sont délimitées par les passages à la ligne `\n`).

```
>>> f = open('C:\\Users\\Python\\README.txt')
>>> lignes = f.readlines()     # liste des lignes du fichier
>>> f.close()
>>> lignes[:4]                 # les quatre premières lignes
['This is Python version 3.3.0\n',
 '============================\n',
 '\n',
 'Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,\n']
```

On peut enfin itérer sur un fichier avec une boucle `for`, l'itération se fait ligne par ligne.

```
>>> f = open('C:\\Users\\Python\\README.txt')
>>> for ligne in f:
...     print(ligne, end='')
This is Python version 3.3.0
============================

Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
...
>>> f.close()
```

Écriture

La méthode `write` permet d'écrire dans un fichier. Si celui-ci a été ouvert en mode `'w'`, il est d'abord vidé de son contenu. S'il a été ouvert en mode `'a'`, ce qui est ajouté l'est à la fin du fichier.

Dans l'exemple suivant, on crée un nouveau fichier sur le bureau :

```
>>> f = open('C:\\Users\\Desktop\\unfichier.txt', 'w')
>>> f.write('Ce fichier est un test.')
23       # le nombre de caractères écrits dans le fichier
>>> f.close()
```

Si on veut ajouter ultérieurement quelque chose au fichier il faudra l'ouvrir en mode `'a'`.

Python : Compléments

Types

La fonction `type` renvoie le type d'un objet, qui détermine quelles opérations on peut effectuer sur lui, quelles fonctions on peut lui appliquer, etc.

```
>>> type(12)
<class 'int'>
>>> type(1.5)
<class 'float'>
>>> type([1, 2, 3])
<class 'list'>
```

Quelques types de base :

- `bool` - booléens : `True` et `False`.
- `int` - entiers : `-12`, `6484231317861321`...
- `float` - flottants : `2.0`, `-5.123`, `6.0221418e23`, `-1.602e-19`...
- `str` - chaînes de caractères (*string* en anglais) : `"abcd"`, `'abcd'`, `''`...
- `list` - listes : `[1, 2, 3]`, `[12, 'xyz', [1, 2, 3]]`, `[]`...

Certaines fonctions permettent de convertir :

```
>>> int('123')        # chaîne -> entier
123
>>> int(1.23)         # flottant -> entier
1
>>> float('123')      # chaîne -> flottant
123.0
>>> float(1)          # entier -> flottant
1.0
>>> str(123)          # entier -> chaîne
'123'
>>> str(1.23)         # flottant -> chaîne
'1.23'
>>> str([1, 2, 3])    # liste -> chaîne
'[1, 2, 3]'
>>> list('abcd')      # chaîne -> liste
['a', 'b', 'c', 'd']
```

Affichage et interaction avec l'utilisateur

Par défaut, la fonction `print` insère un espace entre les objets qu'on lui demande d'afficher, et elle passe à la ligne à la fin. Il est possible de modifier les valeurs du séparateur `sep` et du caractère de fin de ligne `end` (qui par défaut valent `' '` et `\n`).

```
>>> print(1, 2, 3, sep='*')
1*2*3
>>> print(1, 2, 3, end='*')
1 2 3*
>>> for x in range(10):
...     print(x, end='')
0123456789
```

La fonction `input` permet d'interagir avec l'utilisateur. Le résultat est une chaîne de caractères (qu'il faut donc éventuellement convertir).

```
>>> x = input('Donnez un entier : ')
Donnez un entier : 42
>>> x
'42'      # x est une chaîne de caractères, pas un entier
```

Compléments sur les boucles

On peut ajouter un pas (éventuellement négatif) comme troisième argument d'un `range`.

```
>>> for i in range(1, 10, 2):
...     print(i, end=' ')
1 3 5 7 9
>>> for i in range(9, 0, -1):
...     print(i, end=' ')
9 8 7 6 5 4 3 2 1
```

Il est possible de créer des boucles `while` infinies si la condition est toujours réalisée :

```
>>> while True:
...     print('Une boucle infinie')
Une boucle infinie
Une boucle infinie
Une boucle infinie
...
```

L'instruction `break` permet d'arrêter l'exécution d'une boucle.

```
>>> for i in range(5):
...     if i == 2:
...         break      # quand i vaut 2 on sort de la boucle
...     print(i)
0
1
```

Divers

Dans le shell, on peut récupérer le dernier résultat calculé avec `_`.

```
>>> 1 + 1
2
>>> _
2
```