Travaux pratiques d'introduction 9

Chaînes littérales et fichiers

Informatique tronc commun MPSI et PCSI

I Rappels de cours

I.1 Les chaînes littérales

Les chaînes littérales ou chaînes de caractères sont des données textuelles qui sont manipulées avec le type `str` ou `string` en Python.

Écriture des chaînes littérales. Les chaînes littérales peuvent être écrites de différentes manières :

Définition 1 : Écriture d'une chaîne littérale

Il est possible d'écrire une chaîne littérale à l'aide de :

- guillemets simples : `'autorisent les "guillemets"'` ;
- guillemets : `"autorisent l'utilisation des guillemets simples"` ;
- guillemets triples : `'''Trois guillemets simples'''` ou `"""Trois guillemets"""`.

Les chaînes entre triples guillemets peuvent faire plusieurs lignes.
Il est aussi possible de convertir une entier ou un flottant en chaîne littérale à l'aide de l'instruction `str`.

Quelques exemples :

```
>>> s = 'Attention à l'apostrophe'
  File "<stdin>", line 1
    s = 'Attention à l'apostrophe'
                      ^
SyntaxError: invalid syntax
>>> s = "Attention à l'apostrophe"
>>> print(s)
Attention à l'apostrophe
>>> s = """Les triples guillemets permettent de définir des textes
    avec des retours à la ligne.
... Voici un tel texte."""
>>> print(s)
Les triples guillemets permettent de définir des textes avec des
   retours à la ligne.
Voici un tel texte.
```

Les chaînes littérales sont des séquences. À ce titre, elles disposent des opérations classiques qui existent pour les autres types séquentiels (listes et tuples essentiellement) :

longueur : `len s` renvoie la longueur (= nombre de caractères) de la chaîne littérale ;

concaténation : `s + t` renvoie la concaténation de `s` et de `t` ;

```
>>> s = "Bonjour "
>>> t="le monde !"
>>> u=s+t
>>> len(u)
18
>>> print(u)
Bonjour le monde !
```

accès aux éléments par leur indice : `s[i]` est le i^e caractère de `s` en commençant par 0 ;

```
>>> u[9]
'e'
```

tranche : `s[i:j]` sous-chaîne allant du caractère `i` inclus à `j` exclu, si `i` n'est pas spécifié alors il vaut 0 et si `j` n'est pas spécifié alors la tranche va jusqu'au dernier caractère de la chaîne ;

```
>>> u[2:10]
'njour le'
>>> u[:3]
'Bon'
>>> u[4:]
'our le monde !'
```

répétition : `t = s*3` génère la chaîne composée de la concaténation de `s` trois fois ;

```
>>> v = u*2
>>> print(v)
Bonjour le monde !Bonjour le monde !
```

itération : `for c in s:` permet d'itérer sur tous les caractères de la chaîne.

```
>>> total = 0
>>> for c in u:
...     if c == "e":
...         total += 1
...
>>> print(total)
2
```

Les chaînes littérales sont immuables. Il est impossible de modifier un caractère ou la longueur de la chaîne. La seule possibilité est de redéfinir la chaîne ou d'en créer une nouvelle. Remplacement d'un caractère :

```
>>> u
'Bonjour le monde !'
>>> u[4] = "4"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> v = u[:4]+ "4" + u[5:]
>>> print(v)
Bonj4ur le monde !
```

Ajout d'un caractère en fin de chaîne :

```
>>> u.append("l")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'append'
>>> u = u + 'l'
>>> print(u)
Bonj4ur le monde !l
```

I.2 Les fichiers

Pour ouvrir un fichier, il suffit d'utiliser l'instruction **open** qui ouvre un fichier donné et crée un **file object** que l'on peut ensuite manipuler. La syntaxe est la suivante :

```
open(chemin du fichier,mode)
```

où

- `chemin du fichier` est une chaîne littérale indiquant le chemin d'accès au fichier ;
- `mode` est une chaîne littérale indiquant le mode d'ouverture du fichier qui peut-être
  - `'r'` pour une ouverture en lecture seule,
  - `'w'` pour une ouverture en écriture, créant le fichier s'il n'existe pas et le tronquant s'il existe,
  - `'a'` pour une ouverture en écriture, créant le fichier s'il n'existe pas et ajoutant à la fin du fichier s'il existe.

Le module `os`. Ce module définit des instructions qui permettent de retrouver facilement le chemin d'un fichier donné. Les instructions principales de ce module sont :

- `os.getcwd()` renvoie une chaîne de caractères représentant le répertoire de travail actuel ;
- `os.listdir()` renvoie une liste contenant le nom des entrées dans le répertoire de travail actuel ;
- `os.chdir(chemin)` change le répertoire de travail actuel par celui donné par `chemin` ; le chemin peut être donné de manière relative ou absolue.

Remplacement du répertoire de travail actuel afin de pouvoir ouvrir le fichier `resultats.txt` se trouvant dans le répertoire */home/johann/Bureau/persoServeur/* :

```
>>> f = open('resultats.txt','r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  FileNotFoundError: [Errno 2] No such file or directory: '
     resultats.txt'
>>> os.getcwd()
'/home/johann'
>>> os.chdir('Bureau/persoServeur')
>>> os.getcwd()
'/home/johann/Bureau/persoServeur'
>>> os.listdir()
['resultats.txt', 'autresResultats.tx']
>>> f = open('resultats.txt','r')
>>> f
<_io.TextIOWrapper name='resultats.txt' mode='r' encoding='UTF-8'>
```

Travailler avec un objet fichier. Une fois qu'un objet fichier a été créé avec l'instruction `open`, il est possible de lire et écrire dans le fichier avec les méthodes suivantes :

- `close` permet de fermer l'objet fichier ;
- `write` permet d'écrire dans le fichier la chaîne littérale passée en entrée et renvoie le nombre de caractère écrits ;

```
>>> f = open('resultats.txt','w')
>>> for i in range(1,3):
...     f.write('*'*i + '\n')
...
2
3
4
>>> f.close()
```

Le caractère `\n` dans une chaîne littérale permet de provoquer un retour à la ligne. Ainsi le fichier *resultats.txt* contient les lignes suivantes :

```
*
**
***
```

- `read` permet de stocker le contenu complet d'un fichier dans une chaîne littérale ;

```
>>> f = open('resultats.txt','r')
>>> contenu = f.read()
>>> f.close()
>>> contenu
'*\n**\n***\n'
```

- `readlines` permet de stocker toutes les lignes du fichier dans une liste ;

```
>>> f = open('resultats.txt','r')
>>> listContenu = f.readlines()
>>> f.close()
>>> listContenu
['*\n', '**\n', '***\n']
```

- `readline` permet de stocker en mémoire uniquement la prochaine ligne du fichier ;

```
>>> f = open('resultats.txt','r')
>>> ligne = f.readline()
>>> while ligne != '':
...     print(ligne)
...     ligne = f.readline()
...
*

**

***
>>> f.close()
```

cette méthode est utile lorsque le fichier est très volumineux et qu'il est possible d'avoir des problèmes de mémoire en manipulant un fichier trop volumineux.

- `split('separateur')` est une méthode qui s'applique aux chaînes littérales et pas aux fichiers, elle renvoie une liste des mots de la chaîne en utilisant `separateur` comme séparateur de mots.

```
>>> chaine = "1,2,3"
>>> chaine.split(',')
['1', '2', '3']
```