TD 2

1 Fichiers

Cette partie sera abordée en cours. En gros, on fait :

```
f = open("monfichier.txt", "r")
for x in f:
     print(x)
f.close()

f = open("monfichier.txt", "w")  # Ràz le fichier
f.write("Nouveau texte")
f.close()
```

Exercice : lecture de fichier

En mode interactif, lire le fichier volume07.txt et afficher son contenu, ligne par ligne.

Attention, on ne veut pas introduire des sauts de lignes qui ne sont pas déjà dans le fichier ! Il y a au moins deux façons de faire, soit en utilisant une méthode du type *string*, soit un paramètre de la fonction *print*. Essayez les deux méthodes.

Passez à un script qui fait la même chose.

Exercice : entités XML

Le fichier contient des entités XML, comme `&#176;` ou `&` . On va les convertir automatiquement en caractères, mais avant ça, on va vérifier à quels caractères elles correspondent, pour ne pas introduire de problèmes.

Écrivez un script qui affiche et écrit dans un fichier toutes les entités de volume07.txt. Pour ce faire, vous créerez une fonction `getEntities` qui prend en entrée une chaîne de caractères (= une ligne du fichier) et retourne la liste des entités qu'elle contient. Une entité commence par un caractère "&" , et se termine par un caractère ";". Tous les caractères "&" du fichier marquent un début d'entité, sans ambiguïté ; par contre, il peut y avoir des ";" un peu partout, pas uniquement à la fin d'une entité.

Début de la sortie attendue :

```
['&amp;', '&amp;', '&amp;']
['&amp;']
['&amp;']
['&amp;']
['&amp;']
['&amp;']
['&amp;']
['&amp;']
['&amp;']
['&amp;']
```

On irait plus vite avec des Regex. Mais pour cet exercice, utilisez **uniquement les fonctionnalités de base des chaînes de caractères**, et un algorithme de type automate à états finis.
Pour aller plus loin : vérifiez que l'entité contient, entre le "&" et le ";", uniquement des caractères présents dans la chaîne "abcdefghijklmnopqrstuvwxyz0123456789#".

2 Sets

Cette partie sera abordée en cours. En gros, un set*, c'est un dictionnaire avec uniquement des clés.*

À présent, on connaît les structures de données suivantes :

- listes : [1, 2, 3, 4, 5]
- dictionnaires : {"a": 1, "b": 2, "c": 3, "d": 4, "e": 5}
- sets : {"a", "b", "c", "d", "e"}

On a aussi vu les séquences :

- listes : ce sont aussi des séquences.
- ranges : range(1, 6) ← pas vraiment une structure de données, puisqu'on ne peut rien écicre dedans !
- Chaînes de caractères.

Exercice

Modifier le programme pour afficher uniquement un *set* des entités XML.

Sortie attendue :

```
{'&uml;', '&#187;', '&#171;', '&amp;', '&#167;', '&#176;',
'&#183;'}
```

Modifier le programme pour afficher chaque entité XML, suivie de sa version décodée. On utilisera la méthode `unescape` de l'objet `html` fourni par le module `html`.

Sortie attendue :

```
&#171; «
&#176; °
&#183; ·
&#167; §
&amp; &
&#187; »
&uml; ¨
```

Enfin, à l'aide de la méthode `unescape`, écrivez un programme qui lit volume07.txt, décode toutes les entités XML, et écrit le résultat dans un fichier volume07.decoded.txt .

3 Récapitulatif

On a vu les structures de données : listes, dictionnaires et sets.

On a vu les séquences range, listes et chaînes :

- `len(s)` pour connaître la longueur d'une séquence.
- `for i in s` pour parcourir une séquence.
- `if i in s` pour savoir si i est dans la séquence s.

On a vu les fonctions sur les chaînes de caractères :

Les chaînes ont aussi des fonctions qui leur sont propres

Voir la liste complète dans la doc python

`lower()` transforme la chaine en minuscules
`upper()` transforme la chaine en majuscules
`replace(old, new)` remplace les occurrences de `old` par `new`
`strip(chars=None)` appelé sans arguments supprime le ou les espaces en tête et en fin de chaîne
`rstrip(chars=None)` fait la même chose en fin de chaîne uniquement
`lstrip(chars=None)` idem en début de chaîne
`split(sep=None)` découpe une chaîne en fonction de `sep` et renvoie une liste. Si `sep` n'est pas donné, coupe sur tous les caractères d'espace
`join(iterable)` est l'inverse de `split`, il permet de joindre les éléments d'une liste de chaînes pour former une seule chaîne de caractères
`format()` depuis python3 (et python2.7) pour effectuer l'interpolation de chaîne

4 Tokenisation

On va tokéniser ce texte. On coupe sur les espaces et les signes de ponctuation usuels ; ces derniers sont des tokens.

Ex.

```
Bonjour
,
comment
ça
va
?
```

→ même les signes de ponctuation sont des tokens.

Exercice

Créez une fonction `lineToToken` qui prend en entrée une ligne et retourne une liste de tokens.

Exercice

Utilisez la fonction `lineToToken` pour afficher le contenu du fichier, un token par ligne.

Exercice

Utilisez la fonction `lineToToken` pour afficher le nombre de tokens, et le token le plus long.

5 Abréviations

Problème : on tokénise aussi les abréviations… Il faudrait les traiter différemment, mais pour ça il nous faudrait en dresser la liste.

Exercice

Écrire une fonction tokenStop qui prend en entrée une liste de tokens et retourne tous les tokens qui précèdent un caractère « . ».

Ex.

```
Cher
M         <--
.
Machin    <--
.
Je
ne
vous
aime
pas
beaucoup  <--
.
```

Exercice

Écrire une fonction `freq` qui prend en entrée une liste de tokens et retourne un dictionnaire.

Il y a plusieurs façons de faire, notamment une qui utilise la méthode *count()* du type *list*. Votre approche est-elle efficiente ?

Cours

→ Listes et dictionnaires en compréhension

6 Métadonnées

Mots grecs

Trouver tous les tokens contenant des caractères grecs « faciles » (par exemple α). Partir de là pour trouver tous les caractères grecs (comme ὴ).

7 Lien dépôt interro

Votre programme doit s'appeler **nom_prénom.py** . Vous pouvez le déposer plusieurs fois.

https://cloud.llf-paris.fr/nextcloud/index.php/s/ZZd5S9g4AiR5gDo

Lien de secours **uniquement** si le précédent ne marche pas :
~~https://kdrive.infomaniak.com/app/collaborate/267819/56f75b26-5522-471e-ac51-e752496d7620~~