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']
```

II Exercices

Exercice 1 - Création d'un fichier
Écrire une fonction `creationFichierEtoile(nomFichier,n)` qui crée un fichier dans votre répertoire *persoServeur/Travail* dont le nom est *nomFichier* qui contient n lignes telles que :

- la première ligne du fichier contient n fois le caractère `*` ;
- la deuxième ligne du fichier contient $n-1$ fois le caractère `*` ;

  ...
- la dernière ligne du fichier contient une fois le caractère `*`.

Exercice 2 - Nombre de lignes dans un fichier
Écrire une fonction `nbLignes(cheminFichier)` qui renvoie le nombre de lignes contenues dans le fichier dont le chemin d'accès est `cheminFichier`.
Déterminer le nombre de lignes du fichier *texte.txt* disponible dans le répertoire *Public* de la classe.

Exercice 3 - Nombre de mots dans un fichier
Écrire une fonction `nbMots(cheminFichier)` qui renvoie le nombre de mots contenues dans le fichier dont le chemin d'accès est `cheminFichier`. On considérera que les mots sont séparés par des espaces.
Déterminer le nombre de mots du fichier *texte.txt* disponible dans le répertoire *Public* de la classe

Exercice 4 - Nombre de lettres dans un texte
Écrire une fonction `nbLettres(texte)` qui indique le nombre de lettres apparaissant dans `texte`. Attention, il ne faut pas compter les caractères espace `" "`, les points, les virgules et les retour à la ligne `'\n'`. Vous pourrez considérer que la chaîne littérale `texte` ne contient que des lettres, des espaces, des points, des virgules et des retours à la ligne comme caractères.
Déterminer le nombre de lettres apparaissant dans le fichier *texte.txt*.

Exercice 5 - Nombre d'occurrences

1. Écrire une fonction `nbOccurrence(texte,lettre)` qui indique le nombre de fois où la lettre `lettre` est présente dans `texte`.

   Afin de gérer correctement les majuscules, il faut utiliser la méthode `lower()` qui transforme une lettre majuscule en minuscule et qui ne change pas les minuscules.

   Déterminer le nombre de fois où la lettre `t` apparaît dans le fichier *texte.txt*. Donnez ensuite ce résultat sous forme du pourcentage de présence de la lettre `t` dans toutes les lettres du texte.

La fonction **`ord`** renvoie le nombre entier représentant le code Unicode du caractère représenté par la chaîne donnée. Par exemple, **`ord('a')`** renvoie le nombre entier 97. La fonction inverse existe : **`chr`**. Les codes des lettres minuscules de l'alphabet latin vont de 97 à 122.

2. À l'aide de la fonction `nbOccurrences`, écrire une fonction `lettrePlusPresente(texte)` qui renvoie la lettre la plus présente dans `texte` ainsi que son nombre d'occurrences. En déduire la lettre la plus présente dans le texte `texte.txt`.
3. En termes de complexité la solution précédente n'est pas optimale. En effet, pour chaque lettre il faut parcourir tout le texte afin de déterminer son nombre d'occurrences. Le texte est ainsi parcouru 26 fois. Pour éviter cela, il suffit de déterminer les occurrences de toutes les lettres en même temps et de stocker dans une liste le nombre de fois ou une lettre donnée est rencontrée en parcourant le texte.

   À l'aide de cette idée, écrire une fonction `occurrences(texte)` qui renvoie une liste des occurrences de chaque lettre. Les occurrences de la lettres `a` se trouveront à l'indice 0, celle de la lettre `b` à l'indice 1 ...

   Utiliser cette fonction afin de déterminer les lettres non présentes dans *texte.txt* ?

Exercice 6 - Recherche d'une sous-chaîne dans un texte

Écrire une fonction `rechercheSousChaine(texte,ssChaine)` qui va rechercher la chaîne `ssChaine` dans `texte` et qui va renvoyer la liste des indices de la première lettre de `ssChaine` quand elle apparaît dans `texte`.

Indice :

Penser à utiliser des tranches du texte dont la longueur est égale à celle de la sous-chaîne.

Combien de fois et à quels endroits apparaît le mot *Interdum* dans le fichier `texte.txt` ?

Exercice 7 - Codage de César

On cherche à crypter un texte t de longueur n composé de caractères en minuscules (soit 26 lettres différentes) et de caractères espace uniquement (pas de virgules, points ...).
Le codage de César est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la droite de 1 ou plusieurs positions. Par exemple, en décalant les lettres de 1 position, le caractère a se transforme en b, le b en c, ...le z en a. Le texte *ave cesar* devient donc *zud bdrzq*.

1. Écrire une fonction `codageCesar(texte,d)` qui prend en arguments la chaîne `texte` et un entier d ; et qui retourne une chaîne de même taille que `texte` contenant le texte `ttexte` décalé de d positions.
2. Écrire de même une fonction `decodageCesar(texte,d)` prenant en argument une chaîne `texte` contenant un texte codé avec le décalage d qui réalise le décalage dans l'autre sens.

Pour les textes courts, on peut essayer toutes les valeurs de décalages possibles et regarder ceux qui donnent un texte lisible. C'est ce qu'on appelle une attaque par force brute, technique de test de toutes les combinaisons possibles.

3. Écrire une fonction `bruteForce(texte)` qui prend une chaîne en entrée et qui renvoie une liste de 26 chaînes chacune correspondant au décodage du texte avec un décalage variant de 0 à 25.

   En déduire les décodages possibles du texte *nyhcl*. Quelles sont les valeurs de décalage associées.

Cette méthode n'est pas efficace sur un texte long. Il est envisageable de ne prendre qu'une portion du texte pour trouver le décalage mais pour les textes courts il peut y avoir plusieurs possibilités de déchiffrement.
Une manière de déterminer automatiquement la valeur de décalage est d'essayer de deviner cette valeur. L'approche la plus couramment employée est de regarder la fréquence d'apparition de chaque lettre dans le texte crypté. En effet, la lettre la plus fréquente dans un texte suffisamment long en français est la lettre e.

4. En utilisant la fonction `occurrences` de l'exercice 5, écrire la fonction `decodageAuto(texte)` qui prend en argument une chaîne représentant le texte crypté ; et qui renvoie la liste des textes décodés. Il peut en effet y avoir plusieurs possibilités de décodage. On calculera le décalage pour que la lettre e soit la plus fréquente dans le texte décrypté.

   Il est conseillé d'écrire une fonction `posMax(l)` qui renvoie la liste des indices des maxima d'une liste `l` d'entiers.
5. Décoder le texte du fichier *texteCode1.txt* et écrire le ou les textes décodés des fichiers intitulés *decodage1-1.txt*, *decodage1-2.txt*, ...
6. Faire de même pour le texte du fichier *texteCode2.txt*.

   Indice :

   Il s'agit d'un extrait du roman «La disparition» de Georges Perec.
   Internet est votre ami.

III Solutions

Solution de l'exercice 1 - Création d'un fichier

```
def creationFichierEtoile(nom,n):
    f = open('Bureau/persoServeur/Travail/'+nom,'w')
    for i in range(n):
        f.write('*'*(n-i)+'\n')
    f.close()
```

Solution de l'exercice 2 - Nombre de lignes dans un fichier

```
def nbLignes(cheminFichier):
    f = open(cheminFichier,'r')
    nb = 0
    for ligne in f.readlines():
        nb += 1
    f.close()
    return nb
```

Le fichier *texte.txt* contient 138 lignes.

Solution de l'exercice 3 - Nombre de mots dans un fichier
Il suffit de séparer les mots d'une ligne par la méthode `split` puis d'ajouter la longueur de la liste obtenue au nombre de mots.

```
  def nbMots(cheminFichier):
    f = open(cheminFichier,'r')
    nb = 0
    for ligne in f.readlines():
        nb += len(ligne.split(" "))
    f.close()
    return nb
```

Le fichier *texte.txt* contient 12 342 mots.

Solution de l'exercice 4 - Nombre de lettres dans un texte
Il vaut mieux utiliser la méthode `read` pour mettre tout le contenu du fichier dans une chaîne littérale qui servira de texte :

```
def nbLettres(texte):
    nb = 0
    n = len(texte)
    for i in range(n):
        if texte[i] != " " and texte[i] != "." and texte[i] != "\n
           " and texte[i] != ",":
            nb += 1
    return nb

f = open('texte.txt','r')
txt = f.read() # on continuera d'utiliser cette variable dans la
   suite des exercices
nb = nbLettres(txt)
```

Le nombre de lettres dans le fichier *texte.txt* est 68 906.

Solution de l'exercice 5 - Nombre d'occurrences

1. On utilise la variable `txt` définie dans l'exercice précédent :

```
def nbOccurrences(texte,lettre):
    n = len(texte)
    nb = 0
    for i in range(n):
        if texte[i].lower() == lettre:
            nb += 1
    return nb

nb = nbOccurrencesLettre(txt,'t')
p = nb/nbLettres(txt) * 100
```

   Le nombre de fois où la lettre `t` apparaît est 5706 soit $\approx 8,2$ % des lettres du texte.

2. Il suffit de chercher la lettre dont le nombre d'occurrences est le plus élevé :

```
def lettrePlusPresente(texte):
    maxi = 0
    for i in range(97,123):
        nb = nbOccurrences(texte,chr(i))
        if nb > maxi:
            l=chr(i)
            maxi = nb
    return l,maxi
```

   La lettre la plus présente dans *texte.txt* est la lettre `e` qui apparaît 7825 fois.

3. En utilisant l'indication de l'énoncé, sans oublier de ne pas prendre en compte les caractères spéciaux :

```
def occurrences(texte):
    oc = [0]*26
    for c in texte:
        if c != " " and c != "." and c != "\n" and c != ",":
            oc[ord(c.lower())-97] += 1
    return oc
```

   Les lettres `k`, `w`, `y` et `z` ne sont pas présentes dans *texte.txt*.

Solution de l'exercice 6 - Recherche d'une sous-chaîne dans un texte

En utilisant des tranches de texte de la longueur du mot, il est facile de voir quand il apparaît dans le texte :

```
def rechercheSousChaine(texte,ssChaine):
    indices = []
    n = len(texte)
    l = len(ssChaine)
    i = 0
    while i <= n-l:
        if texte[i:i+l] == ssChaine:
            indices.append(i)
        i += 1
    return indices

    rechercheSousChaine(txt,'Interdum')
```

Le mot *Interdum* apparaît 13 fois dans le texte, aux indices `[2467, 3725, 8551, 13219, 17824, 25757, 25940, 30557, 35729, 52730, 60848, 65363, 79571]`

Solution de l'exercice 7 - Codage de César

1. Il suffit de penser à utiliser le modulo pour éviter les valeurs supérieures à 25 :

```
def codageCesar(texte,d):
    n = len(texte)
    texteCode = ""
    for c in texte:
        if c != " ":
            nb = ord(c) - 97 #Nombre entre 0 : a et 25 : z
            nb = (nb +d)% 26 #On ajoute le décalage et on
               prend le modulo pour garder un nombre entre 0
               et 25
            nb += 97 #On ajoute 97 pour retomber sur un nombre
                 entre 97 et 122
            c = chr(nb)
        texteCode += c #En dehors du if, cela permet de gérer
           les caractères espace
    return texteCode
```

2. De le manière la plus simple, un décodage est simplement un codage avec un décalage vers la gauche, il suffit donc de coder avec une valeur de décalage négative :

```
def decodageCesar(texte,d):
    return codageCesar(texte,-d)
```

3. Pas de subtilités ici :

```
def bruteForce(texte):
    l = []
    for i in range(26):
        l.append(decodageCesar(texte,i))
    return l
```

nyhcl peut-être le codage de *grave* avec un décalage de 7 ou de *tenir* avec un décalage de 20.

4. Écriture de la fonction qui détermine les indices des maxima d'une liste :

```
def maximaListe(l):
    indices = [0]
    for i in range(1,len(l)):
        if l[i] > l[indices[0]]:
            indices = [i]
        elif l[i] == l[indices[0]]:
            indices.append(i)
    return indices
```

Ensuite il suffit de trouver la ou les lettres du texte codé la plus présente et d'en déduire leurs décalages par rapport à la lettre e d'indice 4 :

```
def decodageAuto(texte):
    frequences = occurrences(texte)
    indices = maximaListe(frequences)
    texteDecode=[]
    for i in indices:
        decalage= (i - 4)%26
        texteDecode.append(decodageCesar(texte,decalage))
    return texteDecode
```

5. Il faut réinvestir ce qui a été fait précédemment :

```
f = open('texteCode1.txt','r')
txt = f.read()
f.close()
textesDecodes = decodageAuto(txt)
for i in range(len(textesDecodes)):
    s = open('decodage1-'+str(i+1)+'.txt','w')
    s.write(textesDecodes[i])
    s.close()
```

6. Ce roman est écrit sans utiliser la lettre e. Il faut donc changer la fonction de décodage automatique pour prendre en compte ce fait. Suivant les textes, la deuxième qui apparaît le plus dans un texte en français est le a, s, i, t ou n. Il suffit de tester ces différentes possibilités. La première est la bonne, c'est la lettre a qui est la plus présente dans le texte d'origine.