

Feuille d'exercices 4

Les dictionnaires

Exercice 1 Élément le plus fréquent

1. Écrire en Python une fonction prenant en argument une liste de listes de nombres, et qui renvoie l'élément (un élément) le plus fréquent.
2. Même question mais la fonction renverra la liste de tous les éléments de fréquence maximale.

Exercice 2 Décompte de caractères et de mots dans un fichier texte

Pour la manipulation des fichiers textes, on pourra se rapporter au cours concis figurant en annexe.

1. Écrire et exécuter le code suivant :

```
nom = input("Saisir le nom du fichier : ")
texte = input("Saisir le texte du fichier : ")
nom = nom + ".txt"
objet_fichier = open(nom, 'w')
objet_fichier.write(texte)
objet_fichier.close()
```

Vous créerez ainsi votre propre fichier texte (sans caractère accentué!); il se trouvera dans le répertoire courant.

2. a) Écrire une fonction `caracteres` prenant en argument le nom d'un fichier texte présent dans le répertoire courant (de type `str`), et qui renvoie un dictionnaire de champs `c : v` décrivant tous les caractères figurant dans le fichier (ce sont les clés `c`), et pour chacun d'entre eux son nombre d'occurrence (c'est sa valeur `v`).
b) En déduire un script qui affichera le nombre de caractères présents dans le fichier considéré.
c) Écrire une fonction `CaracteresOrdonnes` prenant en argument le nom d'un fichier texte présent dans le répertoire courant (de type `str`), et qui renvoie une liste des couples `(c , v)` décrivant tous les caractères `c` figurant dans le fichier, mais ordonnés par occurrence `v` décroissante.
3. Écrire une fonction `mots` prenant en argument le nom d'un fichier texte présent dans le répertoire courant (de type `str`), et qui renvoie un dictionnaire de champs `m : v` décrivant tous les mots figurant dans le fichier (ce sont les clés `m`), et pour chacun d'entre eux son nombre d'occurrence (c'est sa valeur `v`).

Les mots sont les suites de caractères délimités par des espaces, tabulation `\t`, saut de ligne `\n` et début et fin de fichier. Pour simplifier on distinguera la casse (majuscule et minuscule sont des caractères différents).

Exercice 3 Exemple de table de hachage avec résolution par chaînage

On dispose d'une table de hachage de largeur Ω ; les collisions y sont résolues par chaînage.

Exécuter manuellement l'algorithme d'insertion dans la table, en prenant $\Omega = 9$ et comme fonction de hachage $f : c \mapsto c \bmod \Omega$, pour les clés suivantes : 5, 28, 19, 15, 20, 33, 12, 17, 10.

Exercice 4 Résolution des collisions par adressage ouvert

Une autre méthode de résolution des collisions consiste, en cas de collision, de chercher un emplacement libre où stocker la nouvelle association. Bien sur, pour cela, le nombre de clefs n doit être inférieur à la largeur Ω de la table de hachage. La recherche d'un emplacement libre est dénommée un *sondage*.

- Le *sondage linéaire* consiste en cas de collision à l'emplacement $i = h(c)$ de tester successivement les emplacements $(i + 1) \bmod \Omega$, $(i + 2) \bmod \Omega$, $(i + 3) \bmod \Omega$, etc.

Il est très simple mais présente le désavantage de former des "agrégats", c'est à dire de longues successions de cases occupées, qui nuisent à la répartition uniforme souhaitée, entraînant plus de collisions.

- Le *sondage quadratique* procède de même façon mais en testant successivement les emplacements $(i + 1) \bmod \Omega$, $(i + 1 + 2) \bmod \Omega$, $(i + 1 + 2 + 3) \bmod \Omega$, etc.

On s'intéresse dans cet exercice à la résolution des collisions par adressage ouvert à l'aide d'un sondage linéaire.

1. Exécuter manuellement l'algorithme d'insertion dans la table, en prenant $\Omega = 9$ et comme fonction de hachage $f : c \mapsto c \bmod \Omega$, pour les clés suivantes : 5, 28, 19, 15, 20, 33, 12, 17, 10.
2. On représente la table de hachage par une liste $D = [\text{None}, \text{None}, \dots, \text{None}]$ de longueur Ω et on prend comme fonction de hachage $f : c \mapsto c \bmod \Omega$. Les clefs sont des entiers et les valeurs des chaînes de caractères.
Rédiger les fonctions de lecture `valeur(D, c)`, d'ajout d'une association (c, v) `ajout(D, c, v)` et de recherche d'une clé c `recherche(D, c)`. On supposera que la table de hachage est de capacité suffisante.
3. Quel problème se pose lors de la suppression de certaines associations de la table ? Comment peut-on y remédier ? Écrire le code de la fonction `supprime(D, c)` correspondante. On supposera que la clé c est présente.

Exercice 5 Pré-traitement des chaînes vers les entiers

En général les clés possibles sont d'abord transformées en entiers avant de calculer une valeur de hachage finale grâce à une fonction de $\mathbb{N}$ vers $\mathbb{N}$. C'est ainsi qu'on procède avec les clés de types chaînes de caractères, flottants ou autres tuples...

On propose dans cet exercice un pré-traitement simple des chaînes de caractères, les transformant en entier, à l'aide de la table ASCII.

La table ASCII est une table de caractère –la plus ancienne qui soit. Un jeu de 256 caractères est codé par un entier représenté sur un octet. La plupart des tables actuelles (utf8, latin1, etc...) sont de plus grandes capacités mais contiennent la table ASCII comme sous-table.

En Python pour utiliser la table ASCII, la fonction `chr(n)` renvoie le caractère de code n , la fonction `ord(c)` renvoie le code ASCII du caractère c :

```
In [1]: chr(100)
```

```
Out[1]: 'd'
```

```
In [2]: ord('d')
```

```
Out[2]: 100
```

Voici une partie de la table obtenu à l'aide du code suivant (les premiers caractères de 0 à 32 sont des caractères spéciaux, comme le saut de ligne ou la tabulation) :

```
for k in range(33,127):
    print(chr(k), end=':')
```

produit :

```
! : " : # : $ : % : & : ' : ( : ) : * : + : , : - : . : / : 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 : 9 : : ; : < : =
: > : ? : { : } : A : B : C : D : E : F : G : H : I : J : K : L : M : N : O : P : Q : R : S : T : U : V : W : X : Y : Z
: [ : \ : ] : ^ : _ : ' : a : b : c : d : e : f : g : h : i : j : k : l : m : n : o : p : q : r : s : t : u : v : w
: x : y : z : { : | : } : ~ : - : :
```

1. Écrire une fonction `srt2int(ch)` prenant en argument une chaîne de caractère `ch` de longueur n et qui renvoie la valeur entière :

$$\sum_{k=0}^{n-1} \text{ord}(\text{ch}[k]) \times 256^k$$

Afin d'en réduire la complexité on appliquera l'algorithme de Horner (déjà rencontré) pour le calcul.

2. Soit Ω un entier positif fixé ; on souhaite en déduire une fonction de hachage des chaînes de caractères en renvoyant le reste modulo Ω de l'entier obtenu par pré-traitement. Écrire le code de la fonction `hachage(ch)` ; on prendra soin à ce que le calcul se fasse à moindre coût.

Annexe : Manipulation de fichiers texte

1 Accès à un fichier

1.1 Création d'un objet-fichier

En `python`, création et manipulation d'un fichier se font par l'intermédiaire d'un objet-fichier, généré par la fonction :

```
objet_fichier = open(nom du fichier, mode d'accès).
```

Les paramètres sont des chaînes de caractère :

- `nom du fichier` est le nom du fichier, avec son extension.

- `mode d'accès` peut être :

- `'w'` : (write) ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; lorsque le fichier existe il est écrasé.

- `'a'` : (append) ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; lorsque le fichier existe les données écrites le seront à la suite.

- `'r'` : (read) ouverture pour lecture seule. Le fichier doit exister dans le répertoire courant.

1.2 Où se trouve le fichier ?

Pour créer un objet-fichier :

- Le fichier doit être présent dans le dossier courant (*current working directory* ou *répertoire de travail*) : en général c'est le dossier utilisateur (celui de même nom que le compte utilisateur).
- Sinon on peut faire précéder le nom du fichier par le **chemin d'accès** :

```
obj_fich = open('Users/Moi/Documents/Fichier.ext', 'r')
```

1.3 Obtention et changement du répertoire courant

Pour obtenir le répertoire de travail, utiliser la fonction `getcwd()` du module `os` :

```
>>> import os
>>> os.getcwd()
'/Users/Moi'
```

- Pour modifier le répertoire de travail, utiliser la fonction `chdir` du module `os` :

```
>>> os.chdir('/Users/Moi/Documents')
>>> os.getcwd()
'/Users/Moi/Documents'
```

1.4 Fermeture

Une fois la lecture et l'écriture dans le fichier terminés il faut refermer l'objet-fichier avec l'instruction :

```
objet_fichier.close()
```

- Les modifications apportées au fichier ne seront prises en compte qu'après la fermeture.
- Tant qu'un fichier n'a pas été fermé, son ouverture provoquera une erreur.



Ne jamais oublier de finir par refermer un objet-fichier par `objet_fichier.close()`

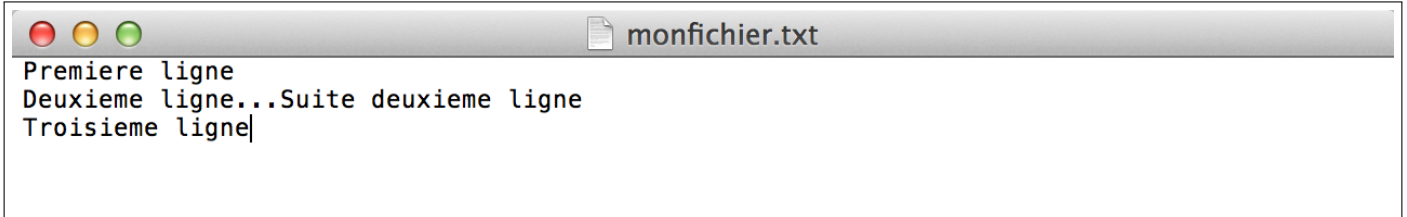
1.5 Méthodes des objets-fichiers

Un objet-fichier admet les méthodes :

- Lorsque **ouvert en écriture** :
`write()` écrit une chaîne de caractère en fin de fichier ouvert en écriture.
- lorsque **ouvert en lecture** :
`read()` lit l'intégralité du fichier. `read(n)` lit `n` caractères.

1.6 Exemples

On suppose que le fichier texte suivant se trouve dans le répertoire courant :



```
>>> objet = open('monfichier.txt','r')
>>> lect = objet.read()
>>> print(lect)
Premiere ligne
Deuxieme ligne...Suite deuxieme ligne
Troisieme ligne
>>> objet.close()
```

On ajoute du texte en fin de fichier grâce au code suivant :

```
>>> objet = open('monfichier.txt','a')
>>> objet.write("Quatrieme ligne rajoutee")
>>> objet.close()
```

