TP 4 - Dictionnaires

Bien sûr, on se connecte, on lance spyder et on poursuit sa lecture. Créez un répertoire TP4 dans le répertoire InfoPCSI de votre session, et y sauvegarder le script de l'éditeur de code (fenêtre de gauche) sous le nom `tp4.py`.

1 Objets mutables et immutables

Il y a deux types d'objets en python : ceux qui peuvent être modifiés (une liste par exemple), appelés objets **mutables** et ceux qui ne peuvent l'être (une chaîne de caractères par exemple) appelés objets **immutables**. On rappelle en effet que pour une liste, on peut, par exemple :

- Modifier la valeur d'un élément :

```
>>> L = [2, 3, 'a']; L[0] = 3.1; print(L)
[3.1, 3, 'a']
```

- Ajouter un élément à la fin :

```
>>> L = [3, 1, 4]; L.append('fin'); print(L)
[3, 1, 4, 'fin']
```

- Extraire le dernier élément :

```
>>> L = [2, 4, [5]]; L.pop()
[5]
>>> L
[2, 4]
```

Au contraire, pour une chaîne de caractères toute tentative de modifier celle-ci se solde par une erreur à l'exécution :

```
>>> s = 'une␣chaîne'; s[0]='z'
... TypeError: 'str' object does not support item assignment
```

Parmi les objets immutables figurent aussi :

- Les entiers, nombres flottants et nombres complexes. Par exemple : si on exécute les instructions suivantes : `i = 2**12` puis `i = i + 1` ou `i += 1`, alors l'entier `2**12` est calculé (4096), sa valeur est stockée quelque part et le nom `i` est donné à cette valeur. Que ce soit pour l'instruction `i = i + 1` ou `i += 1`, le fonctionnement est le même : la valeur 4097 est calculée, mais celle-ci ne vient pas remplacer la valeur initialement stockée et vers laquelle pointait initialement `i`, mais elle est stockée ailleurs, et ce qui change, c'est que la variable `i` vient à pointer vers cette nouvelle valeur (la place mémoire occupée par la valeur précédente peut alors être libérée).

- Les tuples, qui comme les listes permettent de regrouper un certain nombre d'objets, mais au lieu de les entourer de crochets `[]`, on les place entre parenthèses. `t = (2, 3, 1)` définit ainsi un tuple de longueur 3 (`len(t)` renvoie 3). Le fonctionnement, en « lecture » est similaire aux listes, mais toute tentative de changer un élément, d'en ajouter un se solde par une erreur à l'exécution.

Il importe de bien comprendre le fonctionnement de l'affectation en python car l'affectation ne consiste guère qu'à donner un nom à un objet, et rien n'interdit alors de lui donner plusieurs noms, chose que l'on fait souvent, de manière consciente ou non, en particulier avec des listes. Bien entendu alors, si cet objet est un objet mutable, comme une liste, et qu'on lui a donné plusieurs noms, tels que `L1` et `L2`, alors toute modification de l'objet que référence `L1` modifie bien sûr l'objet que référence `L2`, puisque c'est le même !

2 Dictionnaires

Un dictionnaire, également appelé tableau associatif, est un objet en python permettant d'associer à des clés des valeurs, mais contrairement aux listes, les clés ne sont pas forcément des entiers naturels (on rappelle que pour une liste `L` de longueur `n`, on accède aux éléments de `L` par les indices de `0` à `n-1`) mais elles peuvent être n'importe quel objet immutable : autrement dit, un nombre (entier ou flottant), un tuple (formé d'objets immutables !), une chaîne de caractères, mais pas, vous l'aurez compris, une liste par exemple…

On va le voir, un dictionnaire en python est de la catégorie des objets mutables. Il n'est pas rare d'ailleurs qu'on parte d'un dictionnaire vide avec l'une ou l'autre des syntaxes : `d = {}` ou `d = dict()`, et qu'on ajoute un par un des couples formés d'une clé et de la valeur associée avec la syntaxe : `d[clé] = valeur`. Par exemple :

```
>>> d = dict(); d['un'] = 1; d[2] = 'deux'; d
{'un': 1, 2:'deux'}
```

Sans surprise, on accède en lecture à la valeur associée à une clé par la syntaxe `d[clé]`, sachant que, si ladite clé ne fait pas partie de notre dictionnaire, alors on aura une erreur (`KeyError`) à l'exécution :

```
>>> d[2]
'deux'
>>> d[1]
KeyError: 1
```

Pour éviter ce genre d'erreur, on peut tester la présence d'une clé dans un dictionnaire avec la syntaxe suivante :

```
if clé in d:
    print(d[clé])
```

On peut itérer sur les clés d'un dictionnaire :

```
>>> for clé in d: # on peut aussi utiliser for clé in d.keys():
        print(clé, d[clé])
    un 1
    2 deux
```

ainsi que sur ses valeurs avec l'itérable `d.values()` mais c'est peu utile (on ne dispose pas de moyen pour associer à une valeur sa ou ses clés), et sur les couples `(clé, valeur)` avec `d.items()` :

```
>>> for clé, valeur in d.items():
        print(clé, valeur)
    un 1
    2 deux
```

Enfin, même si on ne l'utilisera pas dans ce TP, retirer une clé (et la valeur associée) d'un dictionnaire est obtenu avec l'instruction `del d[clé]`.

3 Comptage des éléments d'une liste

D'une grande liste `L` (ce pourrait être aussi n'importe quel itérable, comme par exemple une chaîne de caractères), on aimerait connaître pour chaque élément apparaissant dans `L`, le nombre d'occurrences de celui-ci.

Considérons par exemple la liste `[2, 5, 4, 2, 4, 2]`. On se convainc rapidement qu'il y a trois entiers qui interviennent dans `L` : 2, 5 et 4 (dans l'ordre d'apparition), et on voit que 2 y apparaît 3 fois, 5 une fois et 4 deux. L'objectif va être d'automatiser ce comptage.

On peut penser à une élection de délégués, où au fur et à mesure du dépouillement, on ajoute un candidat dont le nom apparaît pour la première fois, avec une barrette, ou bien on ajoute une barrette au décompte d'un candidat déjà obtenu. En procédant ainsi, il suffit d'ouvrir une et une seule fois chacun des bulletins.

Pour réaliser ce comptage, on va bien sûr s'aider d'un dictionnaire, dont on fera en sorte que les clés soient les éléments qui apparaissent dans `L`, et que la valeur associée soit son nombre d'occurrences dans `L`. Sur l'exemple précédent, on espère donc construire le dictionnaire `{2: 3, 5: 1, 4: 2}`.

Question 1. Compléter le code suivant pour réaliser cette opération :

```
def compte(L):
    d = {}
    for ...:
        if ... in d:
            ...
        else:
            ...
    return d
```

4 Clé associée à la plus grande valeur

Question 2. Etant donné un dictionnaire `d` dont les valeurs sont des entiers positifs, on cherche quelle est la clé dont la valeur associée est la plus grande. Compléter la fonction suivante qui réalise ce travail. On renverra un couple sous la forme `(clé, valeur)` où `valeur` est la plus grande valeur dans `d` et `clé` en est la clé associée.

```
def cléMax(d):
    assert len(d) > 0, 'Le dictionnaire est vide'
    m = -1
    for ...       :
        ...
    return clé, m
```

Question 3. En guise d'application, écrire une fonction d'entête `def valeurMajoritaire(L):` qui prend une liste non vide `L` en argument, et qui renvoie un couple `(objet, occurrences)` constitué de l'élément `objet` qui apparaît le plus grand nombre de fois, accompagné de son nombre d'occurrences. On fera bien sûr appel aux fonctions précédentes.

5 1984

Dans `https://cahier-de-prepa.fr/pcsi-bdb` allez sur Informatique->Documents à télécharger->Informatique->TP04 et téléchargez le fichier `1984.txt` formé des premières pages du roman bien connu de Georges Orwell, et placez celui-ci dans le répertoire `TP4` que vous avez créé (et dans lequel vous avez enregistré votre script). C'est important car, avec Spyder, en exécutant votre script, vous avez fait du répertoire `TP4` le répertoire par défaut, et c'est ce qui permettra d'accéder au fichier `1984.txt` en ne donnant que le nom du fichier, sans avoir à spécifier le chemin complet où le fichier figure.

En utilisant un autre environnement de développement, vous aurez peut-être besoin de changer le répertoire par défaut à l'aide de la fonction `chdir` du module `os`. Une autre fonction utile est `getcwd` (get current working directory) qui permet de vérifier quel est le répertoire de travail par défaut.

Par exemple, si le répertoire du TP est `/Users/jondot/python/TP4` alors voici comment faire de ce répertoire le répertoire par défaut pour python :

```
from os import chdir
chdir("/Users/jondot/python/TP4")
```

(A priori, cette étape n'est donc pas nécessaire avec Spyder, si tant est que le fichier qui nous intéresse est dans le même répertoire que notre script)

Question 4. On va écrire ici une fonction d'entête `def litEtCompteMots(fichier):` qui admet pour unique argument une chaîne de caractères formant le nom d'un fichier texte, qui ouvre et charge celui-ci et crée un dictionnaire dont les clés sont les mots du texte, en caractères minuscules, et les valeurs associées le nombre d'occurrence de chaque mot. On pourra reprendre le code de la question 2.

Indications : les instructions suivantes permettent de charger le fichier texte dans une seule variable :

```
f = open('1984.txt', 'r') # ouvre en lecture le fichier
s = f.read() # lit le fichier complet, sous la forme d'une longue chaine et nomme celle-ci s
f.close() # ferme le fichier ouvert en lecture.
```

La seconde étape va être de découper la longue chaîne de caractères `s` en une liste de sous-chaînes (les mots qui composent le texte, idéalement). La première idée est de faire appel à la méthode `split()` d'une chaîne, mais le comportement par défaut est de séparer la chaîne autour de chaque espace, mais d'autres séparateurs existent : le caractère `-`, des guillements simples, et on aimerait également se débarrasser des symboles de ponctuation, alors on va faire appel à une fonction du module `re` (qui traite d'expressions régulières) qui propose une fonction `split` un peu plus riche :

```
from re import split
L = split(r"[()-.,';?!\s]+",s)
```

Ici, tout caractère parmi `()-.,';?!` et un espace (ou un saut à la ligne par exemple) que représente `\s`, éventuellement répété est reconnu comme un séparateur, et `L` est une liste formé des mots qui constitue la chaîne `s`.

Il reste à convertir toutes les lettres majuscules en minuscules. Il est une méthode de chaîne de caractères qui réalise exactement cela : `lower`.

Question 5. Combien de fois apparaît le mot « brother » dans le texte proposé ? (sous toutes ses formes : brother, Brother, BROTHER)

Quel est le mot qui y apparaît le plus souvent ?

Question 6. Ecrire une fonction d'entête `motLePlusPrésent(d, n):` qui prend en argument un dictionnaire `d` dont les clés sont les mots d'un texte et les valeurs associés leur nombre d'occurrences, et qui renvoie un couple `(mot, occurrences)` formé du mot de n lettres qui apparaît le plus, et `occurrences` le nombre de fois où ledit mot apparaît. Vérifier avec :

```
>>> motLePlusPrésent(d, 5)
>>> ('which', 35)
```

6 Se passer des dictionnaires en python ? Implémentation naïve

Imaginons réinventer un dictionnaire, mais au lieu de faire appel à la structure de dictionnaire de Python, on décide de présenter un dictionnaire sous la forme d'une liste de listes.

Par exemple, le dictionnaire `{2: 3, 5: 1, 4: 2}` sera représenté par `[[2, 3], [5, 1], [4, 2]]`.

Question 7. On suppose un tableau associatif donné sous la forme d'une liste de listes `L`, où donc chaque élément de `L` est une liste de la forme `[clé, valeur]`. Ecrire une fonction d'entête `valeur(L, clé)` qui renvoie la valeur associée à `clé` si une telle clé figure dans `L` et qui affiche 'clé non présente' si la clé ne figure pas dans `L`.

En adaptant à peine la fonction que vous venez d'écrire, écrire une nouvelle fonction d'entête `cléPrésente(L, clé)` qui renvoie `True` si la clé est présente, et `False` sinon bien sûr.

Question 8. Ecrire également une fonction d'entête `affecte(L, clé, valeur)` qui prend en argument une liste de listes `L` de la forme précédente, et qui ou bien, si `clé` figure parmi les clés de `L` change la valeur qui lui est associée (la remplace par l'argument `valeur`) ou bien ajoute le couple `[clé, valeur]` à la liste `L`.

Question 9. Reprendre la fonction `compte` de la question 1 pour construire non plus un dictionnaire mais une liste de listes à l'aide des fonctions précédentes. On pourra nommer `compte2` cette nouvelle fonction. Vérifier avec :

```
>>> [2, 5, 4, 2, 4, 2]
[[2, 3], [5, 1], [4, 2]]
```

On va désormais tester la complexité temporelle de notre nouvel algorithme. Ajoutez à votre script :

```
from random import randrange
def listeAleatoire(n):
    L = []
    for i in range(n):
        L.append(randrange(10000))
    return L

from time import perf_counter

def compare(n):
    L = listeAleatoire(n)
    t1 = perf_counter()
    d = compte(L)
    t2 = perf_counter()
    D = compte2(L)
    t3 = perf_counter()
    print(t2-t1, t3-t2)
```

Et bien sûr, on teste ! (avec `compare(1000), compare(2000), compare(4000)`…) Vous l'avez compris, la fonction compare renvoie le temps mis dans chacun des deux traitements, avec l'objet dictionnaire de python, et avec notre implémentation avec des listes de listes.

7 Devoir à la maison

Terminer déjà, si ce n'était le cas, les trois questions précédentes, et me les envoyer par mail, et on rajoute :

Question 10. On reste sur l'implémentation naïve d'un dictionnaire : écrire encore une fonction `retire(L, clé)` qui, vous l'aurez compris, retire la clé en argument de `L` (et bien sûr, la valeur associée aussi…), ainsi qu'une fonction `clés(L)` qui renvoie la liste de toutes les clés présentes dans `L`.