Informatique tronc commun 1re année

TP6 : Dictionnaires, 1re partie

Durée : 1 h

1 Petits exercices d'échauffement

Exercice 1

1. Créez un dictionnaire vide qui s'appelle `quillet`.
2. Ajoutez un élément dont la clef est un nom et la valeur un numéro de téléphone (à inventer comme vous voulez). Le numéro de téléphone devra être une chaîne de caractères.
3. Ajoutez un deuxième élément avec un autre nom au choix et un autre numéro de téléphone au choix.
4. Affichez les noms et numéros de téléphone du dictionnaire.
5. Modifiez le numéro de téléphone fictif.
6. Peut-on modifier le nom associé à ce numéro ?

2 Fréquence d'apparition

2.1 Nombres d'apparitions

On a vu dans la partie cours de ce TP comment on peut compter le nombre d'apparitions de chaque lettre A, C, G, T dans une chaîne de caractères constituée uniquement de ces lettres.

Exercice 2

Écrivez une fonction `initialisation_compteurs` qui renvoie un dictionnaire indexé par les clefs `"A"`, `"C"`, `"G"` `"T"` dont les valeurs sont toutes à 0.
Ainsi, `compteur = initialisation_compteurs()` affecte à la variable `compteur` le dictionnaire `{"A": 0, "C": 0, "G": 0, "T": 0}`.

Exercice 3

Ensuite on veut écrire une fonction qui parcourt le texte et incrémente la valeur associée à chaque lettre (=clef) au fur et à mesure de la lecture du texte.
Complétez le code suivant (lignes 6 et 7).

```
1 def compteur_ACGT(texte):
2     """Entrée : une chaîne composée des caractères A, C, G, T
3        Sortie : le dictionnaire dont les clefs sont les lettres
4                 et les valeurs les compteurs de ces lettres"""
5     compteur = initialisation_compteurs()
6     for ... in ...:
7         compteur[...] += 1
8     return compteur
```

2.2 Plus grande fréquence d'apparition

Exercice 4

On veut maintenant chercher la fréquence maximale sans chercher à savoir à quelle(s) lettre elle est associée. Complétez la fonction `fréquence_maximale` ci-dessous, qui prend en entrée une chaîne de caractère ne contenant que les lettres A, C, G et T et renvoie en sortie un entier égal au nombre d'apparitions de la lettre qui apparaît le plus souvent.

```
def fréquence_maximale(texte):
    compteur = compteur_ACGT(texte)
    maxi = ...
    for lettre in compteur:
        if maxi < ...:
            maxi = compteur[lettre]
    return maxi
```

Exercice 5

Variante de la question précédente. Copiez-collez la fonction `fréquence_maximale`, renommez la copie en `lettres_fréquence_maximale` et modifiez-la de sorte qu'elle renvoie la liste des lettres qui ont le record du nombre d'apparitions. Elle devra utiliser `fréquence_maximale` et `compteur_ACGT`.
Vérifiez son bon fonctionnement avec la fonction de test :

```
def test_lettres_fréquence_maximale():
    assert lettres_fréquence_maximale("ACCTGCAA") == ["A", "C"]
```

Exercice 6

Discutez l'efficacité de la fonction `lettres_fréquence_maximale`. Si vous avez le temps, proposez une version améliorée qui réalise le même travail en moins d'opérations.

2.3 Tricomptage

Question un peu plus difficile. Passez à la suivante si vous n'êtes pas déjà à l'aise.

Exercice 7

On peut enfin classer les lettres dans l'ordre croissant d'apparitions. Écrire une fonction `ordre_apparition` qui prend en entrée une chaîne de caractères ne contenant que les lettre A, C, G et T, et renvoie en sortie une liste `L` de longueur égale au nombre maximal d'apparition +1 telle que `L[i]` est la liste des lettres apparaissant `i` fois.
Par exemple pour la chaîne `"ACCTTGGGGGGGTTACA"` on trouvera :

```
[[], [], [], ["A", "C"], ["T"], [], [], ["G"]]
```

3 Entiers friables

Un entier est 3-friable s'il s'écrit comme un produit de 2 et de 3.

3.1 Écriture décimale

Exercice 8

1. Si a est un entier naturel, l'instruction `str(a)` renvoie la chaîne de caractères contenant la valeur de `a`. Essayez sur quelques exemples.
2. Déduisez-en une façon simple de récupérer le 125^{e} chiffre de l'écriture décimale de $2^{45}3^{234}$ (on convient que, par exemple, le premier chiffre de l'écriture décimale de 8654 est 8).

3.2 Comptage

Exercice 9

Écrivez une fonction `init_chiffres` qui renvoie un dictionnaire contenant 10 entrées dont les clefs sont les entiers de 0 à 9 et les valeurs sont nulles.

Exercice 10

Écrivez une fonction `nombres_chiffres` prenant en entrée deux entiers `n` et `p`, et qui renvoie un dictionnaire contenant 10 entrées dont les clefs sont les entiers de 0 à 9 et les valeurs sont le nombre d'apparitions de chaque clef parmi les chiffres de l'écriture décimale de $2^n 3^p$.
Vous pourrez utiliser la fonction `int` qui convertit (quand c'est possible) une chaîne de caractères en un entier.
Vérifiez votre fonction avec :

```
def test_nombres_chiffres():
    assert nombres_chiffres(12, 7) == {0: 0, 1: 0, 2: 1, 3: 0, 4: 0,
                                       5: 2, 6: 0, 7: 1, 8: 1, 9: 2}
```

Exercice 11

Écrivez une fonction `somme_dictionnaire` qui prend en entrée un dictionnaire (dont les valeurs sont des nombres) et renvoie en sortie la somme de ses valeurs.

Exercice 12

Écrivez la liste des probabilités d'apparition des chiffres de 0 à 9 dans l'écriture de $2^{4567}3^{1234}$. Le résultat sera donné sous la forme d'un dictionnaire dont les clef seront les chiffres et les valeurs les probabilités.
Vous utiliserez bien sûr autant que possible les fonctions précédemment écrites.

4 Anagrammes

On considère une liste de mots dans laquelle on veut chercher s'il y a des anagrammes.
Ainsi, voici une liste exemple qui contient un triplet d'anagrammes :

```
mots = ["ceinh", "chien", "roue", "il", "chien", "il", "pleut", "niche"]
```

Exercice 13

Nous avons mis à votre disposition (dans le dossier Public) un fichier `TP06_liste_anagrammes.py` qui contient une variable `liste_mots` dont la valeur est une longue liste de mots. Ouvrez ce fichier et copiez-collez la liste dans votre programme.

Pour un mot donné, si on ordonne ses lettres dans l'ordre alphabétique, on obtient sa *signature*. Par exemple *cehin* est la signature de *chien* mais aussi de son anagramme *niche*.
Vous admettrez que `sig = "".join(sorted(mot))` affecte à `sig` la signature de la chaîne de caractères `mot`.[1]

Exercice 14

Écrivez une fonction `liste_anagrammes` qui prend une liste de mots en entrée et qui renvoie un dictionnaire dont les clefs sont les signatures des mots de la liste et les valeurs sont les listes des mots associés.
Par exemple, vérifiez que `liste_anagrammes(mots)` renvoie le dictionnaire :

```
{"cehin": ["ceinh", "chien", "chien", "niche"],
 "eoru": ["roue"],
 "il": ["il", "il"],
 "elptu": ["pleut"]
}
```

[1] En résumé, `sorted` renvoie une liste dont les éléments sont les lettres triées dans l'ordre alphabétique, puis `join` regroupe la chaîne de caractères vide `""` avec les éléments de la liste.

Exercice 15

Écrivez une fonction `liste_anagrammes2` qui prend en entrée la liste de mots et renvoie en sortie un dictionnaire avec la même structure qu'à l'exercice précédent, mais contenant seulement les clefs associées à plusieurs anagrammes.
Par exemple, vérifiez que `liste_anagrammes2(mots)` renvoie :

```
{"cehin": ["ceinh", "chien", "chien", "niche"],
 "il": ["il", "il"]
}
```

Exercice 16

Écrivez une fonction `liste_anagrammes_sans_doublons` qui prend en entrée une liste de mots et renvoie en sortie un dictionnaire avec la même structure que précédemment, mais :

- sans doublons dans les listes d'anagrammes,
- avec seulement les clefs associées à plusieurs anagrammes.

Par exemple, vérifiez que `liste_anagrammes_sans_doublons(mots)` renvoie :

```
{"cehin": ["ceinh", "chien", "chien", "niche"]}
```

Corrigés des exercices

Solution de l'exercice 1 :

```
quillet = {}
quillet["Amchin"] = "0325555864"
quillet["Tcur"] = "0655222552"
for x in quillet:
    print(x, quillet[x])
quillet["Tcur"] = "4444444444"
```

Solution de l'exercice 2 :

```
def initialisation_compteurs():
    compteur = {}
    compteur["A"] = 0
    compteur["C"] = 0
    compteur["G"] = 0
    compteur["T"] = 0
    return compteur
```

Solution de l'exercice 3 :

```
def compteur_ACGT(texte):
    compteur = initialisation_compteurs()
    for lettre in texte:
        compteur[lettre] += 1
    return compteur
```

Solution de l'exercice 4 :

```
def fréquence_maximale(texte):
    compteur = compteur_ACGT(texte)
    maxi = 0
    for lettre in compteur:
        if maxi < compteur[lettre]:
            maxi = compteur[lettre]
    return maxi
```

Solution de l'exercice 5 :

Une première version peut être facilement obtenue en réutilisant la fonction de l'exercice précédent :

```
def lettres_fréquence_maximale(texte):
    maxi = fréquence_maximale(texte)
    compteur = compteur_ACGT(texte)
    lettres = []
    for lettre in compteur:
        if compteur[lettre] == maxi:
            lettres.append(lettre)
    return lettres
```

Solution de l'exercice 6 :

La fonction `fréquence_maximale` crée le dictionnaire de compteurs et parcourt le texte pour trouver le nombre maximal d'apparitions, puis la fonction `lettres_fréquence_maximale` recrée ce dictionnaire et reparcourt le texte. Il est donc préférable de fusionner ces opérations, mais il faut alors s'abstenir d'utiliser `fréquence_maximale`.

```
1  def lettres_fréquence_maximale(texte):
2      compteur = compteur_ACGT(texte)
3      maxi = 0
4      lettres = []
5      for lettre in compteur:
6          if maxi < compteur[lettre]:
7              maxi = compteur[lettre]
8              lettres = [lettre]
9          elif maxi == compteur[lettre]:
10             lettres.append(lettre)
11     return lettres
```

Solution de l'exercice 7 :

```
1  def ordre_apparition(texte):
2      compteur = compteur_ACGT(texte)
3      maxi = fréquence_maximale(texte)
4      ordre_croissant = [[] for _ in range(maxi+1)]
5      for lettre in compteur:
6          ordre_croissant[compteur[lettre]].append(lettre)
7      return ordre_croissant
```

Solution de l'exercice 8 :

```
1  a = 2**45*3**234
2  b = str(a)
3  chiffre = b[124]
```

Solution de l'exercice 9 :

```
1  def init_chiffres():
2      dico = {}
3      for i in range(10):
4          dico[i] = 0
5      return dico
```

Solution de l'exercice 10 :

```
1  def nombres_chiffres(n, p):
2      compteur = init_chiffres()
3      nombre = 2**n*3**p
4      chaîne = str(nombre)
5      for chiffre in chaîne:
6          compteur[int(chiffre)] += 1
7      return compteur
```

Solution de l'exercice 11 :

```
def somme_dictionnaire(dico):
    somme = 0
    for x in dico:
        somme += dico[x]
    return somme
```

Solution de l'exercice 12 :

```
compteur = nombres_chiffres(4567, 1234)
total = somme_dictionnaire(compteur)
probas = {}
for i in range(10):
    probas[i] = compteur[i]/total
```

Solution de l'exercice 14 :

```
def liste_anagrammes(liste_mots):
    signatures = {}
    for mot in liste_mots:
        sig = "".join(sorted(mot))
        if sig in signatures:
            signatures[sig].append(mot)
        else:
            signatures[sig] = [mot]
    return signatures
```

Solution de l'exercice 15 :

```
def liste_anagrammes2(liste_mots):
    signatures = liste_anagrammes(liste_mots)
    signatures2 = {}
    for s in signatures:
        if len(signatures[s]) > 1:
        signatures2[s] = signatures[s]
    return signatures2
```

Solution de l'exercice 16 :

```
def liste_anagrammes_sans_doublons(liste_mots):
    signatures = liste_anagrammes2(liste_mots)
    sans_doublons = {}
    for s in signatures:
        liste = []
        for mot in signatures[s]:
            if mot not in liste:
                liste.append(mot)
        if len(liste) > 1:
            sans_doublons[s] = liste
    return sans_doublons
```