# TP types construits n°2

Nous allons construire un dictionnaire qui contiendra les mots d'un texte et, pour chaque mot, le nombre d'occurrences de ce mot dans le texte afin de savoir quel est le mot le plus utilisé dans ce texte.

Exercice I :
Commencer par écrire une fonction **occurrences** qui prend en paramètres une liste et qui renvoie un dictionnaire donnant, pour chacune des valeurs apparaissant dans la liste, le nombre de fois qu'elle apparait.

**Exemple :**
Entrer une liste : [1,5,3,8,1,2,4,6,9,2,8,3,3]
{1: 2, 5: 1, 3: 3, 8: 2, 2: 2, 4: 1, 6: 1, 9: 1}

**Indice :** Pour créer un dictionnaire vide : dic={}

Exercice II :
Ecrire une fonction **plusFrequent(dic,n)** qui prend comme paramètres un dictionnaire et un entier. Elle doit renvoyer le mot du dictionnaire de n lettres qui est associé à la plus grande valeur.

**Exemple :**
```
dic={"les":5,"des":8,"mes":2,"aux":6}
print(plusFrequent(dic,3))
```

On obtient en sortie : "des"

Exercice III :
On va appliquer la fonction précédente à un fichier texte: "*Le tour du monde en quatre-vingt jours*" de Jules Verne.

- Placer les fichiers "tour_du_monde.txt" et "tour_du_monde_essai.txt" dans le même répertoire que votre fichier python.

  Nous allons utiliser la fonction **open()** pour ouvrir le fichier txt puis la fonction **split()** pour le transformer en une liste qui contient chaque mot sans les espaces. La ponctuation a déjà été supprimée du texte.

  **Exemple :**
  ```
  f=open("tour_du_monde_essai.txt")
  texte=f.read().split()
  f.close()
  print(texte)
  ```

- Utiliser les fonctions précédentes pour afficher les mots les plus fréquents dans "tour_du_monde.txt" pour des longueurs de 1 à 10 caractères.
  On doit obtenir la sortie suivante :

  1 lettre : **à** qui apparait 1678 fois.
  2 lettres : **de** qui apparait 2826 fois.
  3 lettres : **les** qui apparait 952 fois.
  etc ……. jusqu'à 10 caractères.

Langages et programmation

Exercice IV :

Ecrire une fonction **occurrences_str** qui accepte une chaîne de caractères au lieu d'une liste et qui renvoie un dictionnaire donnant, pour chacun des caractères de la chaîne, le nombre de fois qu'il apparait.

**Exemple :**

```
chaine="ouagadougou"'
print(occurrences_str(chaine))
```

*Sortie* : {'o': 3, 'u': 3, 'a': 2, 'g': 2, 'd': 1}

Exercice V :

Ecrire une fonction **compListes(l1,l2)** qui prend deux listes en paramètres et qui renvoie True si les deux listes contiennent les mêmes éléments, avec le même nombre d'occurrences pour chacun.
Vous devez utiliser la fonction **occurrences.**

**Exemples :**

```
liste=[1,1,1,2,2,3,3,3,3,5]
liste2=[1,2,3,1,2,3,1,3,5,3]
print(compListes(liste,liste2))
```

*Sortie :* True

```
liste=[1,1,1,2,2,3,3,3,3,5]
liste2=[1,2,3,1,2,3,1,3,5,5]
print(compListes(liste,liste2))
```

*Sortie :* False

```
liste=[1,1,1,2,2,3,3,3,3,5]
liste2=[1,2,3,5]
print(compListes(liste,liste2))
```

*Sortie :* False

Exercice VI : Temps de traitement.

On désire créer un traducteur anglais-français. On dispose de deux méthodes pour enregistrer les couples de mots.

```
dic={"oui":"yes","non":"no","bonjour":"hello","au revoir":"goodbye"}
liste= [["oui", "yes"], ["non", "no"], ["bonjour", "hello"], ["au revoir", "goodbye"]]
```

La deuxième méthode utilise une liste de listes.

1. Ecrire une fonction **rechercheDic(dic,m)** qui prend en paramètre un dictionnaire et une variable et qui renvoie la valeur correspondant à la clé "m".

   **Exemple :**

   ```
   dic={"oui":"yes","non":"no","bonjour":"hello","au revoir":"goodbye"}
   print(rechercheDic(dic,"non"))
   ```

   *Sortie :* "no"

2. Ecrire une fonction **rechercheListe(liste,m)** qui prend en paramètre une liste de listes et une variable et qui renvoie le deuxième élément de la sous-liste dont le premier est "m".

   **Exemple :**

   ```
   liste= [["oui", "yes"], ["non", "no"], ["bonjour", "hello"], ["au revoir", "goodbye"]]
   print(rechercheListe(liste,"non"))
   ```

   *Sortie :* "no"

Pour créer une très grande liste et un très grand dictionnaire, utiliser le code suivant:

```
from random import shuffle
liste=[[i,i] for i in range(10**5)]
shuffle(liste)
dic=dict(liste)
```

Pour mesurer le temps d'exécution d'un morceau de programme, utiliser le code suivant :

```
from time import time
debut=time()
# morceau de programme à mesurer
print(time()-debut())
```

3. Ecrire un test qui permet de mesurer les temps de traitement des deux fonctions précédentes pour 100 recherches successives.

Pour aller plus loin :

Rechercher une valeur dans un dictionnaire en connaissant la clé par le code : **dic[clé]** génère une erreur si la clé n'existe pas dans le dictionnaire.
Modifier le code (si ce n'était pas déjà fait) de la fonction **rechercheDic** en utilisant : dic.get(clé).
Tester à nouveau les temps d'exécution.