Exercice 1 :

1) Soit d un dictionnaire. Si la clé k n'est pas dans le dictionnaire, l'instruction $d[k] = 5$ provoque-t-elle une erreur ?
2) Soit d un dictionnaire. L'instruction $d[k] = [10,5]$ provoque-t-elle une erreur ?
3) Soit d un dictionnaire. L'instruction $d[[10,5]] = 5$ provoque-t-elle une erreur ?

1)Non, si k n'est pas encore une clé connue, cette instruction rajoute le couple $\{k: 5\}$ au dictionnaire existant

2)Non, les valeurs peuvent être des objets quelconques mutables.

3)Oui, une clé ne peut pas être un objet de type liste car mutable.
$d[(10,5)]$ aurait été possible.

Exercice 2 :

1) Créer le dictionnaire ci-dessous :

```
jours={"lundi":1,"mardi":2,"mercredi":3,"jeudi":4,"vendredi":
5,"samedi":6,"dimanche":8}
```

2) Corriger l'erreur `"dimanche":8` en `"dimanche":7`
3) Créer un programme qui affiche la liste des clés
4) Créer un programme qui affiche la liste des valeurs
5) Créer un programme qui affiche la liste des pairs (clé,valeur)

```
#question 1
jours={"lundi":1,"mardi":2,"mercredi":3,"jeudi":4,"vendredi":
5,"samedi":6,"dimanche":8}
jours["dimanche"]=7
print(jours)
#question 2
liste_cle=[]
for cle in jours:
    liste_cle.append(cle)

print(liste_cle)
print(jours.keys())
for cle in jours.keys():
    liste_cle.append(cle)

print(liste_cle)

#question 3

liste_valeur=[]
for cle in jours :
    liste_valeur.append(jours[cle])
print(liste_valeur)

#question 4

liste=[]

for couple in jours.items():
    liste.append(couple)
print(liste)
```

Exercice 3 :

On considère les deux dictionnaires suivants :

```
dico1={"paris":75,"La Rochelle":17}
dico2={"Rennes":35,"Orléans":45}
```

Ecrire un programme permettant de concaténer en seul dictionnaire les clés et valeurs des 2 dictionnaires précédents

```
dico1={"paris":75,"La Rochelle":17}
dico2={"Rennes":35,"Orléans":45}

dico={}
for cle,valeur in [dico1.items(),dico2.items()]:
    dico[cle]=valeur
print(dico)
```

Exercice 4 :

On considère un dictionnaire donnant la moyenne de quelques étudiants.

```
dico={"etudiant1":15,"etudiant2":9,"etudiant3":12,"étudiant4":5}
```

Partitionner ce dictionnaire en deux sous dictionnaires : un dictionnaire des élèves ayant la moyenne et un dictionnaire des élèves n'ayant pas la moyenne

```
dico_sup={}
dico_inf={}
for clé in dico_etudiant:
    if dico_etudiant[clé]>=10:
        dico_sup[clé]=dico_etudiant[clé]
    else:
        dico_inf[clé]=dico_etudiant[clé]
print(dico_sup,dico_inf)
```

Exercice 5 :

Écrire une fonction en Python qui prend en paramètre une liste de nombres entiers et qui renvoie un dictionnaire dont les clés sont les entiers de la liste et dont les valeurs sont 'pair' ou 'impair' selon la parité du nombre.

```
L=[0,1,2,3,4,5,6,7,8,9]
dico={}
for i in L:
    if i%2:
        dico[i]="impair"
    else :
        dico[i]="pair"
```

Exercice 6 :

Ecrire une fonction *comptage* qui prend en paramètre un mot et renvoie un dictionnaire qui pour chaque caractère du mot associe le nombre d'occurrence de chaque lettre. On suppose le texte écrit en lettres capitales non accentuées.

```
def comptage(chaine):
    dico={}
    for i in chaine:
        if i in dico:
            dico[i]=dico[i]+1
        else :
            dico[i]=1
    return dico
```

Une instruction comme *if dico*[*i*] == 1 provoquerait une erreur car aucune clé n'existe au départ (donc problème dès la 1e lettre de la chaîne de caractère)

Exercice 7 :

Ecrire une fonction *comptage2* qui prend en paramètre un mot et renvoie un dictionnaire qui pour chaque caractère du mot associe le nombre d'occurrence de chaque lettre. On suppose le texte écrit en lettres capitales non accentuées. On précisera que les autres caractères ., ;:!? sont exclus.

```
def comptage2(chaine):
    dico={}
    for i in chaine :
        if i not in ",;:?! ":
            if i in dico:
                dico[i]=dico[i]+1
            else :
                dico[i]=1
    return dico
print(comptage2("ABC, DEFG"))
print(sorted(comptage2("TESTS").items()))#permet de faire un tri alphabétique
```

Exercice 8 :

Ecrire une fonction qui prend en argument un mot (type str) et qui renvoie un dictionnaire ayant pour clés les caractères du mot et pour valeurs les distances (les plus courtes si des caractères apparaissent plusieurs fois) entre chaque caractère du mot et le dernier caractère de ce mot.

```
def comptage3(chaine):
    dico={}
    n=len(chaine)
    for i in range(len(chaine)):
        dico[chaine[i]]=n-i-1
    return dico
```

Exercice 9 :

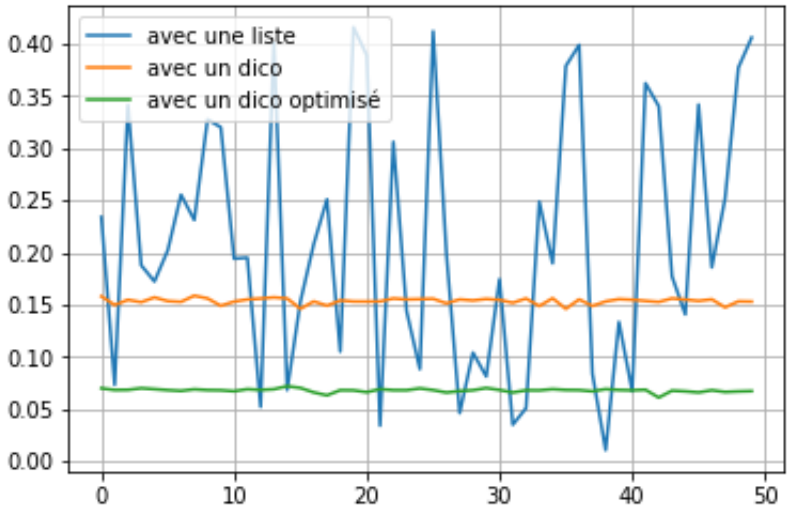
La question est de mesurer l'intérêt d'une représentation par un dictionnaire plutôt que par une liste de listes. Nous allons vérifier que le temps de recherche d'une valeur dans un dictionnaire est le même quel que soit le nombre d'éléments de cette structure (ce qui n'est pas le cas avec une liste).

1) Construire une liste, appelée *liste*, dont les éléments sont de la forme $[i,i]$ pour i allant de 0 à 10^6-1. Mélanger cette liste avec la fonction *shuffle* du module *random*. Il suffit d'importer la fonction et d'écrire *shuffle*(*liste*). Créer ensuite le dictionnaire correspondant qui contient les couples *cle*: *valeur* de la forme i: i à l'aide de la fonction *dict*.
2) Ecrire une fonction *recherche*1 qui prend en paramètres une liste de listes et une variable k et renvoie le 2e élément de la sous liste dont le premier élément a la valeur de k.
3) Ecrire une fonction *recherche*2 qui prend en paramètres un dictionnaire et une variable k et renvoie la valeur correspondant à la clé k.
4) Tester les fonctions de recherche sur la liste et le dictionnaire en utilisant pour le paramètre k des valeurs allant de 0 à 49.
   Pour les mesures, utiliser la fonction *time* du module *time* ainsi que le module *matplotlib*

```
from random import shuffle

#1e méthode pour créer la liste
liste=[[i,i] for i in range(10**6)]
#2e méthode pour créer la liste
liste=[]
for i in range(10**6):
    liste.append([i,i])
shuffle(liste)
dico=dict(liste)

def recherche1(liste,k):
    for sous_liste in liste:
        if sous_liste[0]==k:
            return sous_liste[1]


def recherche2(dico,k):
    for cle in dico :#méthode pas vraiment optimale
        if cle==k:
            return dico[k]
def recherche3(dico,k):
    if k in dico :#on profite ici des améliorations offertes
par les dictionnaires
        return dico[k]
```

```
import time
import matplotlib.pyplot as plt
temps_liste=[]
temps_dico1=[]
temps_dico2=[]
for k in range(50):
    t0=time.time()
    recherche1(liste,k)
    temps_liste.append(time.time()-t0)
    t0=time.time()
    recherche2(liste,k)
    temps_dico1.append(time.time()-t0)
    t0=time.time()
    recherche3(liste,k)
    temps_dico2.append(time.time()-t0)


plt.plot([k for k in range(50)],temps_liste,label="avec une
liste")
```

```
plt.plot([k for k in range(50)],temps_dico1,label="avec un
dico")
plt.plot([k for k in range(50)],temps_dico2,label="avec un
dico optimisé")
plt.legend()
plt.grid()
plt.show()
```



On remarque bien un temps de recherche constant. Le mot clé *in* utilisé pour un dictionnaire fait référence un algorithme appelé table hachage qui utilise une fonction f de hachage $clé \overset{f}{\Rightarrow} nvelle\ valeur$. Avec cette fonction f de hachage, la recherche $f(clé)$ se fait à coût constant.

Ce qui n'est pas le cas avec une liste qui présente un temps de calcul qui est fonction de la position de la clé dans la liste.