Laboratoire 7

- **Durée 3 heures**

EXERCICE 1

Écrivez une fonction nommée ***reverseLookup*** qui trouve toutes les clés d'un dictionnaire qui correspondent à une valeur spécifique. La fonction prendra deux paramètres : le dictionnaire et la valeur à rechercher. Elle renvoie une liste (éventuellement vide) des clés du dictionnaire qui correspondent à la valeur fournie.

```
13  frEn = {
14      "le": "the",
15      "la": "the",
16      "livre": "book",
17      "pomme" : "apple"
18  }
19
20  print(f"The french words for 'the' are {reverseLookup(frEn, 'the')}")
21  # The french words for 'the' are ['le', 'la']
22
23  print(f"The french words for 'apple' are {reverseLookup(frEn, 'apple')}")
24  #The french words for 'apple' are ['pomme']
25
26  print(f"The french words for 'apple' are {reverseLookup(frEn, 'pomme')}")
27  #The french words for 'apple' are []
```

EXERCICE 2

Créez un programme qui détermine et affiche le nombre de caractères uniques dans une chaîne de caractères saisie par l'utilisateur. Par exemple, Hello, World ! a 10 caractères uniques alors que zzz n'a qu'un seul caractère unique. Utilisez un dictionnaire ou un ensemble pour résoudre ce problème.

```
Enter a string: Hello, world !
{'H': True, 'e': True, 'l': True, 'o': True, ',': True, ' ': True, 'w': True, 'r': True, 'd': True,
 '!': True}

(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\Exercices\Labo09\Lab>python ex2.py
Enter a string: zzz
{'z': True}
```

EXERCICE 3

Deux mots sont anagrammes s'ils contiennent les mêmes lettres, mais dans un ordre différent. Par exemple, "evil" et "live" sont des anagrammes car chacun contient un e, un i, un l et un v. Créez un programme qui lit deux chaînes de caractères de l'utilisateur, détermine s'il s'agit ou non d'anagrammes et communique le résultat.

```
(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\Exercices\Labo09\Lab>python ex3.py
Enter the first String: evil
Enter the second String: live
evil and live are anagrams

(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\Exercices\Labo09\Lab>python ex3.py
Enter the first String: evil
Enter the second String: livel
evil and livel are not anagrams
```

EXERCICE 4

Dans le jeu de Scrabble, des points sont associés à chaque lettre. Le score total d'un mot est la somme des scores de ses lettres. Les lettres les plus courantes valent moins de points, tandis que les lettres moins courantes valent plus de points. Les points associés à chaque lettre sont indiqués ci-dessous :

1 point	A, E, I, L, N, O, R, S, T et U
2 points	D et G
3 points	B, C, M et P
4 points	F, H, V, W et Y
5 points	K
8 points	J et X
10 points	Q et Z

Écrire un programme qui calcule et affiche le score Scrabble d'un mot. Créez un dictionnaire qui associe les lettres aux valeurs des points. Utilisez ensuite le dictionnaire pour calculer le score.

```
(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\Exercices\Labo09\Lab>python ex4.py
Enter a word: Bonjour
Bonjour has the score = 10

(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\Exercices\Labo09\Lab>python ex4.py
Enter a word: Hello
Hello has the score = 8
```

EXERCICES SUPPLÉMENTAIRES

EXERCICE 1

Écrivez une fonction **add** qui prend une liste des dictionnaires et qui concatène ces dictionnaires en un seul dictionnaire puis elle retourne ce dictionnaire.

```python
da={"name":"Alex" , "address": "125 Bd X"}
db={"Age": 25, "Gender": "Male"}

lst = [da, db]
addition = add(lst)
print(addition)#{'name': 'Alex', 'address': '125 Bd X', 'Age': 25, 'Gender': 'Male'}
```

EXERCICE 2

Écrivez une fonction **addKey** qui prend trois paramètres : un dictionnaire, une clé et une valeur. Ensuite elle ajoute la clé et sa valeur au dictionnaire si cette clé n'existe pas dans le dictionnaire.

```
da={"name":"Alex" , "address": "125 Bd X"}
addKey(da, "course", ["java"])
print(da) # {'name': 'Alex', 'address': '125 Bd X', 'course': ['java']}
addKey(da, "course", ["python"])
print(da) #{'name': 'Alex', 'address': '125 Bd X', 'course': ['java']}
```

EXERCICE 3

Écrivez une fonction **existKey** qui prend un dictionnaire et une clé. Ensuite elle retourne True et la valeur de la clé si cette clé est dans le dictionnaire. Sinon, elle retourne False et None.

```
da={"name":"Alex" , "address": "125 Bd X"}
flag, kValue = existKey(da, "name")
print(f"flag={flag} value = {kValue}") # flag=True value = Alex
flag, kValue = existKey(da, "age")
print(f"flag={flag} value = {kValue}") # flag=False value = None
```

EXERCICE 4

Écrivez une fonction **getByIndex** qui prend un dictionnaire et un entier (index) comme paramètre. Puis el le retourne la clé et la valeur qui se trouve sur cet index dans le dictionnaire. Par exemple, si index = 0, ge tByIndex retourne la première clé et sa valeur dans le dictionnaire. Sinon elle retourne None, None.

```
da={"name":"Alex" , "address": "125 Bd X"}
key, value = getByIndex(da, 1)
print(f"key={key} and value={value}") #key=address and value=125 Bd X
key, value = getByIndex(da, 2)
print(f"key={key} and value={value}") # key=None and value=None
```

EXERCICE 5

Écrivez une fonction **cleanDic** qui permet de nettoyer un dictionnaire en éliminant toutes les clés ayant des valeurs None.

```
da={"name":"Alex" , "address": "125 Bd X", "age": None}
print(da) #{'name': 'Alex', 'address': '125 Bd X', 'age': None}
cleanDic(da)
print(da)#{'name': 'Alex', 'address': '125 Bd X'}
```

EXERCICE 6

Écrivez une fonction **rename** qui prend trois paramètres : un dictionnaire et deux clés (source et cible). Ensuite, si la clé source existe dans le dictionnaire, elle la remplace par la clé cible.

```
da={"name":"Alex" , "address": "125 Bd X"}
rename(da, "name", "first Name")
print(da) # {'address': '125 Bd X', 'first Name': 'Alex'}
rename(da, "Age", "doesnt Exist")
print(da) # {"name":"Alex" , "address": "125 Bd X"}
```

EXERCICE 7

Écrivez une fonction **maxMin** qui prend un dictionnaire comme paramètre puis elle retourne les valeurs maximum et minimum dans le dictionnaire.

```
ages = {"pierre": 15, "anne":2, "steph": 18}
minV, maxV = minMax(ages)
print(f"minV={minV}, maxV={maxV}") #minV=2, maxV=18
```

EXERCICE 8

Écrivez une fonction **randomValue** qui prend un dictionnaire comme paramètre et retourne une valeur aléatoire à partir des valeurs existantes dans le dictionnaire.

```
ages = {"pierre": 15, "anne":2, "steph": 18}
rValue = randomValue(ages)
print(rValue) #18
```

EXERCICE 9

Écrivez une fonction **sortDic** qui prend un dictionnaire dValue dont les valeurs des clés sont des listes. Et puis elle transforme le dictionnaire dValue en triant les listes.

```
d = {'k1': [22, 18, 23, 5], 'k2': [12, 15, 8, 14],
    'k3': [9, 18, 4, 12] , 'k4':[21,17,8,9]}

sortDic(d)
print(d)#{'k1': [5, 18, 22, 23], 'k2': [8, 12, 14, 15], 'k3': [4, 9, 12, 18], 'k4': [8, 9, 17, 21]}
```

EXERCICE 10

Écrivez une fonction **updateVote** qui prend comme parametre un dictionnaire dont les clés sont les noms des personnes et les valeurs sont leurs **â**ges associés. Puis elle remplace les âges des personnes par 'You cannot vote' si l'age < 18 et 'You can vote' si non.

```
pers_age = {"Pierre": 15 , "Alia" : 21 , "Ribal": 27 , "Alain" : 35 }
updateVote(pers_age)
print(pers_age)
#{'Pierre': 'You cannot vote', 'Alia': 'You cannot vote', 'Ribal': 'You cannot vote', 'Alain': 'You cannot vote'}
```

EXERCICE 11

Écrivez une fonction dicOcc qui prend une chaine de caractères comme paramètre et retourne un dictionnaire dont les clés sont les caractères de la chaine et les valeurs sont les nombres d'occurrences des caractères dans cette chaine.

```
d = dicOcc("Java")
print(d) #{'J': 1, 'a': 2, 'v': 1}
```