Tp7 R1.1

Exercice 1, application directe du cours :
Écrivez des programmes permettant de tester les exemples en python de manipulation de tableaux vus en cours (remplissage d'un tableau de 10 valeurs et son affichage, saisie et affichage d'un tableau à deux dimensions, passage de paramètres).

Exercice 2 :

Écrire un programme (avec un menu) permettant de :

1. Saisir un nom et afficher le nombre de voyelles
2. Saisir un nom et vérifier qu'il s'agit d'un palindrome.
3. Quitter

Rappels :

- le type str est manipulable "comme" un tableau :

```
if __name__ == "__main__":
    chaine : str
    longueur : int
    chaine = input("entrez une chaine : ")
    print("le premier caractère est",chaine[0])
    longueur = len(chaine)
    print("la chaine fait",longueur,"caractere(s)")
    print("le dernier caractère est",chaine[longueur-1])
```

- un palindrome est un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche : « kayak », « elle ». Nous considèrerons également que majuscules et minuscules sont des lettres différentes (« Laval » ne sera pas un palindrome, « LAVAL » en sera un)

Exercice 3 :

On veut effectuer les mêmes opérations que celles de l'exercice 2 sur des noms stockés dans un tableau (et non plus saisis un par un par l'utilisateur).
La saisie des noms, successivement placés dans un tableau, s'effectue tant que l'utilisateur répond 'o' à la question "Encore un nom ?"

Le menu devient donc :

```
1- Saisir un tableau de noms
2- afficher le nombre de voyelles par nom
3- afficher uniquement les noms étant des palindromes
4- quitter
```

Exercice 1

programme:

```python
if __name__ == "__main__":
    tab1: list[int]
    tab2: list[int]
    tab3: list[int]
    tab4: list[list[int]]
    avg:int
    i:int
    j:int

    #remplissage d'un tableau
    tab1 = [1, 2, 3, 4, 5,6,7,8,9,10]
    print("tableau 1 : ", tab1)

    #ajout d'un élément
    tab1.append(11)
    print("tableau 1 après ajout : ", tab1)

    #suppression d'un élément
    tab1.remove(5)
    print("tableau 1 après suppression : ", tab1)

    #copie d'un tableau
    tab2 = tab1.copy()
    print("tableau 2 : ", tab2)

    #concaténation de deux tableaux
    tab3 = tab1 + tab2
    print("tableau 3 : ", tab3)
```

```python
#calcul de la moyenne
avg = 0
for i in range(len(tab3)):
    avg += tab3[i]
avg = avg // len(tab3)
print("moyenne : ", avg)

#tableau a deux dimensions
tab4 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print("tableau 4 : ", tab4)

#affichage de la diagonale
for i in range(len(tab4)):
    print(tab4[i][i])

#affichage de la diagonale inversée
for i in range(len(tab4)):
    print(tab4[i][len(tab4)-1-i])

#affichage de la matrice
for i in range(len(tab4)):
    for j in range(len(tab4[i])):
        print(tab4[i][j], end=" ")
    print()
```

Pour ce programme, j'ai mis en place des choix de conception qui mettent l'accent sur la **simplicité**, la **lisibilité** et la **facilité de manipulation**.
L'utilisation de listes et de boucles permet de réaliser des opérations de base telles que l'**ajout**, la **suppression** et la **copie de données**, rendant le code accessible et compréhensible. J'ai choisi d'utiliser des **tableaux unidimensionnels** (`tab1`, `tab2`, et `tab3`), ce qui simplifie les manipulations élémentaires comme la **concaténation** et le **tri**, facilitant la gestion de données sans complexité inutile.
Pour les besoins de `tab4`, j'ai opté pour une structure de **tableau à deux dimensions**, permettant une gestion plus avancée, comme **l'affichage des éléments le long de diagonales**. Cela offre une **vue matricielle** plus complexe et polyvalente qui permet d'aborder des opérations de manipulation de données sous un autre angle.

Globalement, cette conception rend le code à la fois **fonctionnel** et **facilement lisible**, permettant une **navigation claire** à travers les différentes étapes du programme et simplifiant ainsi la compréhension des opérations réalisées.

résultat:

```
tableau 1 :  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
tableau 1 après ajout :  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
tableau 1 après suppression :  [1, 2, 3, 4, 6, 7, 8, 9, 10, 11]
tableau 2 :  [1, 2, 3, 4, 6, 7, 8, 9, 10, 11]
tableau 3 :  [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11]
moyenne :  6
tableau 4 :  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
1
5
9
3
5
7
1 2 3
moyenne :  6
tableau 4 :  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
1
5
9
3
5
7
1 2 3
9
3
5
7
1 2 3
3
5
7
1 2 3
7
1 2 3
1 2 3
4 5 6
7 8 9
```

Les résultats de ce programme montrent que les manipulations sur les tableaux sont correctes et reflètent bien les choix de conception établis. Chaque opération sur les données est effectuée avec exactitude, et le code produit les résultats attendus, que ce soit lors des ajouts, suppressions ou lors de l'affichage des matrices.

Exercice 2

programme:

```python
menu_options ={
    "1": "nombre de voyelles",
    "2": "palyndrome ?",
    "3": "quitter",
}

def print_menu():
    for key in menu_options.keys():
        print(key, '- -', menu_options[key])

def compte_voyelles(chaine):
    nb_voyelles = 0
    for lettre in chaine:
        if lettre in voyelles:
            nb_voyelles += 1
    return nb_voyelles

def est_palyndrome(mot):
    est_palyndrome = True
    for i in range(len(mot)//2):
        if mot[i] != mot[len(mot)-1-i]:
            est_palyndrome = False
            break
    return est_palyndrome
```

```python
if __name__ == "__main__":
    chaine :str
    voyelles:set
    n:int
    nb_voyelles:int
    i:int
    mot:str
    palyndrome:bool
    voyelles = {'a', 'e', 'i', 'o', 'u', 'y'}

    while(True):
        print_menu()
        option = ''
        try:
            option = input("choix : ")
        except:
            print("erreur de saisie")

        if option == '1':
            chaine = input("saisir une chaine : ")
            nb_voyelles = compte_voyelles(chaine)
            print("nombre de voyelles : ", nb_voyelles)
        elif option == '2':
            mot = input("saisir un mot : ")
            is_palyndrome = est_palyndrome(mot)
            if is_palyndrome:
                print("le mot est un palyndrome")
            else:
                print("le mot n'est pas un palyndrome")
        elif option == '3':
            break
```

Ce programme propose une **interface interactive** qui permet à l'utilisateur de sélectionner parmi trois options :

- ★ **compter le nombre de voyelles dans une chaîne de caractères**
- ★ **vérifier si un mot est un palindrome**
- ★ **quitter le programme**

J'ai structuré le menu à l'aide d'un **dictionnaire** `menu_options`, qui permet de simplifier l'affichage des options et rend le programme plus modulaire et extensible.

En isolant chaque tâche dans une fonction dédiée, `compte_voyelles` pour le comptage des voyelles et `est_palindrome` pour la vérification des palindromes, j'améliore la **modularité**, rendant le code **réutilisable** et **facilement compréhensible**. Cette organisation rend le programme plus **user-friendly**, permettant à l'utilisateur d'utiliser les fonctionnalités de manière **intuitive** et **rapide**, tout en favorisant une maintenance simplifiée du code.
Cette approche modulaire permet également de **tester chaque fonction indépendamment**, facilitant la détection d'éventuelles **erreurs** ou **incohérences** dans chaque partie du programme.

tests

```
1 - - nombre de voyelles
2 - - palyndrome ?
3 - - quitter
choix : 1
saisir une chaine : j'aime l'algorithmique
nombre de voyelles :  9
1 - - nombre de voyelles
2 - - palyndrome ?
3 - - quitter
choix : 1
saisir une chaine : jjjjj
nombre de voyelles :  0
1 - - nombre de voyelles
2 - - palyndrome ?
3 - - quitter
choix : 2
saisir un mot : lol
le mot est un palyndrome
1 - - nombre de voyelles
2 - - palyndrome ?
3 - - quitter
choix : 2
saisir un mot : ile
le mot n'est pas un palyndrome
1 - - nombre de voyelles
2 - - palyndrome ?
3 - - quitter
choix : 2
saisir un mot : kayak
le mot est un palyndrome
1 - - nombre de voyelles
2 - - palyndrome ?
3 - - quitter
choix : 3
```

Pour garantir la fiabilité du programme, j'ai conçu des tests spécifiques pour chaque option. Les tests impliquent notamment d'entrer différentes chaînes de caractères pour vérifier que le comptage des voyelles est exact et que les mots identifiés comme palindromes sont corrects. Ces tests valident que chaque fonctionnalité produit les résultats escomptés et que le programme réagit de manière appropriée à divers types d'entrées.

exercice 3

pseudo code

```
fonction compte_voyelle(chaine:chaine) retourne entier

fonction qui permet de retourner le nombre de voyelles présentent
dans une chaine entrer dans le code principale
entrée: chaine
sortie: entier

voyelles:chaine
voyelles <- {'a', 'e', 'i', 'o',
'u','y','é','è','ê','à','ù','â','î','ô','û'}
nb_voyelles:int
```

```
lettre:int
nb_voyelles ← 0
lettre ← 0

Pour lettre 0 à chaine-1
      si chaine = voyelles faire:
            nb_voyelles ← nb_voyelles + 1
retourne nb_voyelles
```

```
fonction est_palindrome(mot:chaine)
fonction qui permet de savoir si une chaine de caractères est un
palindrome
entrée: chaine de caractère
sortie: booléen

compteur_i:int
réponse: booléen
réponse← false
pour compteur_i de 0 à mot/2 faire
      si mot[compteur_i] != mot[taille(mot)-1-compteur_i] alors
            retourne réponse
      réponse← vrai
retourne réponse
```

```
programme tableau_de_noms
début
      avec: noms[100]:chaine, option:entier, continuer: booléen ,
oui_non:car , compteur_i:entier, nb_voyelles:entier,
palindromes[100]:chaine
      option←0

      tant que option !=4 faire
            afficher "1- entrer un tableau de noms
            a la ligne
            afficher "2- afficher le nombre de voyelles par nom"
            a la ligne
            afficher "3- afficher uniquement les noms étant des
                                                      palindromes"
            à la ligne
            afficher "4- Quitter"
            à la ligne
            afficher "Votre choix: "
            saisir option

            Tant que option<1 ou option>4 faire
                  afficher "1- entrer un tableau de noms
                  a la ligne
                  afficher "2- afficher le nombre de voyelles par nom"
                  a la ligne
```

```
      afficher "3- afficher uniquement les noms étant des
                                                palindromes"
      à la ligne
      afficher "4- Quitter"
      à la ligne
      afficher "Votre choix: "
      saisir option
fin faire

si option=1 alors:
      continuer← vrai
      tant que continuer = vrai faire:
            afficher "saisir un nom:"
            saisir nom
            afficher " encore un nom? (o/n):"
            saisir oui_non
            si oui_non = 'o' alors
                  continuer← vrai
            sinon
                  si oui_non = 'n' alors
                        continuer ← faux
                  sinon
                  tant que oui_non!= 'o' ou oui_non != 'n'
                                                      faire:
                        afficher " encore un nom? (o/n):"
                        saisir oui_non
                  fin faire
                  fin si
            fin si
      fin faire
sinon:
      si option=2
      compteur_i ← 0
      nb_voyelles ← 0
      Pour noms[compteur_i] a noms[noms-1] faire
            nb_voyelles = compte_voyelles(noms)
            afficher "nombre de voyelles dans ", noms,
                                        ":",nb_voyelles
            à la ligne
      fin faire
      sinon
            si option=3 alors:
                  afficher " Noms étant des palindromes:"
                  à la ligne
                  palindromes<- noms pour noms si
                                              est_palindrome
                                        (noms)
                  si palindromes!=0
                        pour palindromes[0] à
                                    palindromes[palindromes]
                                    faire:
```

```
                                        afficher palindromes
                                    fin faire
                                fin si
                            fin si
                        fin si
                fin si
                fin faire
                afficher “Au revoir”

fin tableau_noms
```

programme

```python
menu_options = {
    "1": "Saisir un tableau de noms",
    "2": "Afficher le nombre de voyelles par nom",
    "3": "Afficher uniquement les noms étant des palindromes",
    "4": "Quitter",
}

def print_menu():
    for key in menu_options.keys():
        print(key, '- -', menu_options[key])

def compte_voyelles(chaine):
    voyelles = {'a', 'e', 'i', 'o', 'u', 'y'}
    nb_voyelles = 0
    for lettre in chaine:
        if lettre in voyelles:
            nb_voyelles += 1
    return nb_voyelles

def est_palindrome(mot):
    for i in range(len(mot) // 2):
        if mot[i] != mot[len(mot) - 1 - i]:
            return False
    return True
```

```python
if __name__ == "__main__":
    noms = []

    while True:
        print_menu()
        option = input("Choix : ")

        if option == '1':
            # Saisie d'un tableau de noms
            while True:
                nom = input("Entrez un nom : ")
                noms.append(nom)
                continuer = input("Encore un nom ? (o/n) : ")
                if continuer.lower() != 'o':
                    break

        elif option == '2':
            # Afficher le nombre de voyelles par nom
            for nom in noms:
                nb_voyelles = compte_voyelles(nom)
                print(f"Nombre de voyelles dans '{nom}' : {nb_voyelles}")

        elif option == '3':
            # Afficher les noms qui sont des palindromes
            print("Noms étant des palindromes :")
            palindromes = [nom for nom in noms if est_palindrome(nom)]
            if palindromes:
                for palindrome in palindromes:
                    print(palindrome)
            else:
                print("Aucun palindrome trouvé.")

        elif option == '4':
            print("Au revoir!")
            break
```

```python
else:
    print("Option invalide, veuillez réessayer.")
```

Ce programme commence par définir **un menu interactif**, permettant à l'utilisateur de:

- ★ **saisir des noms,**
- ★ **de calculer le nombre de voyelles dans chaque nom**
- ★ **d'afficher uniquement les noms palindromes**

J'ai utilisé une **boucle infinie** pour **afficher en continu les options du menu** jusqu'à ce que l'utilisateur choisisse de **quitter**.

Au début, l'utilisateur peut choisir l'option de **saisie d'un tableau de noms**. Dans cette section, une boucle `while` est utilisée pour recueillir **plusieurs noms** et vérifier si l'utilisateur souhaite en **ajouter davantage**. Chaque nom est stocké dans une **liste** `noms` afin de pouvoir les traiter ultérieurement.

Pour l'option de **calcul des voyelles**, j'ai intégré une fonction `compte_voyelles` qui parcourt **chaque nom** et **vérifie si chaque caractère est une voyelle**. Si c'est le cas, un **compteur incrémente**, et le nombre de voyelles est affiché pour chaque nom. Cette approche utilise une boucle `for` pour examiner chaque nom et une logique conditionnelle pour compter les voyelles.

L'option de **recherche des palindromes** utilise une fonction `est_palindrome`, où j'ai choisi de comparer les caractères à des **indices symétriques du mot**. Si tous les caractères correspondent, le nom est identifié comme un **palindrome** et **affiché**. J'ai opté pour une liste de compréhension afin d'identifier les palindromes en une ligne concise.

Enfin, le programme offre une **option pour quitter le menu**. Les messages d'erreur sont gérés pour assurer que l'utilisateur entre des choix valides.

jeux de test

```
1 - - Saisir un tableau de noms
2 - - Afficher le nombre de voyelles par nom
3 - - Afficher uniquement les noms étant des palindromes
4 - - Quitter
Choix : 1
Entrez un nom : tom
Encore un nom ? (o/n) : o
Entrez un nom : Kim
Encore un nom ? (o/n) : o
Entrez un nom : oxo
Encore un nom ? (o/n) : o
Entrez un nom : pierre
Entrez un nom : tom
Encore un nom ? (o/n) : o
Entrez un nom : Kim
Encore un nom ? (o/n) : o
Entrez un nom : oxo
Encore un nom ? (o/n) : o
Entrez un nom : pierre
Encore un nom ? (o/n) : o
Entrez un nom : Kim
Encore un nom ? (o/n) : o
Entrez un nom : oxo
Encore un nom ? (o/n) : o
Entrez un nom : pierre
Entrez un nom : oxo
Encore un nom ? (o/n) : o
Entrez un nom : pierre
Entrez un nom : pierre
Encore un nom ? (o/n) : o
Entrez un nom : QUUQ
Encore un nom ? (o/n) : n
1 - - Saisir un tableau de noms
2 - - Afficher le nombre de voyelles par nom
3 - - Afficher uniquement les noms étant des palindromes
4 - - Quitter
Choix : 2
```

Ici je choisi de tester option par option mon programme pour commencer je remplis mon tableau de noms pour que les autres options

```
Nombre de voyelles dans 'tom' : 1
Nombre de voyelles dans 'Kim' : 1
Nombre de voyelles dans 'oxo' : 2
Nombre de voyelles dans 'pierre' : 3
Nombre de voyelles dans 'QUUQ' : 0
1 - - Saisir un tableau de noms
2 - - Afficher le nombre de voyelles par nom
3 - - Afficher uniquement les noms étant des palindromes
4 - - Quitter
Choix : 3
Noms étant des palindromes :
oxo
QUUQ
```

Ex4

programme

Pour enrichir le programme, j'ai décidé d'ajouter la gestion de **tableaux d'entiers**, en choisissant de ne pas réutiliser les matrices déjà étudiées dans l'exercice 1. Le programme inclut plusieurs fonctionnalités avancées qui démontrent une manipulation plus poussée des tableaux :

- Une **rotation** d'un tableau vers la **droite** ou vers la **gauche**, permettant de réorganiser les éléments de manière circulaire.
- Un tri par méthode de **tri à bulle**, qui trie les éléments dans un ordre spécifique en permutant successivement les éléments adjacents.
- **Ajout** et **suppression** d'un élément à un **indice spécifique**, permettant d'insérer ou de retirer des valeurs selon les besoins de l'utilisateur.
- Calcul du **nombre d'occurrences** d'un entier dans un tableau, fournissant un outil statistique pour analyser la fréquence des valeurs.
- **Union** et **intersection de deux tableaux**, **avec et sans doublons**, permettant de comparer les contenus de deux tableaux et d'extraire les valeurs uniques ou répétées.

Ces opérations, intégrées de manière fonctionnelle et **optimisées**, élargissent les possibilités d'utilisation du programme et démontrent une compréhension des opérations de manipulation de données dans des structures de tableaux.

```python
def exPassage(li : list[str], val : int) -> None :
    print("modifications dans la procedure :")
    li[0] = "zero"
    val = val*2
    print(li," valeur = ",val)

def exPassage2(li : list[str], val : int) -> None :
    print("modifications dans la procedure :")
    li[1] = "un"
    val = val*2
    print(li," valeur = ",val)

def décaler_tableau_vers_la_gauche(li : list[int]) -> None :
    last : str
    last=li[0]
    for i in range(len(li)-1):
        li[i] = li[i+1]
    li[len(li)-1] = last

def décaler_tableau_vers_la_droite(li : list[int]) -> None :
    first : str
    first=li[len(li)-1]
    for i in range(len(li)-1,0,-1):
        li[i] = li[i-1]
    li[0] = first

def tri_bulle(tab : list[int]) -> None :
    for i in range(len(tab)):
        for j in range(len(tab)-1):
            if tab[j] > tab[j+1]:
                tab[j], tab[j+1] = tab[j+1], tab[j]
```

```python
def ajouter_element_a_lindice_p(li : list[int], p:int, autre:int) -> None :
    li.append(0)  # Add a dummy element to increase the size of the list
    for i in range(len(li)-1, p, -1):
        li[i] = li[i-1]
    li[p] = autre

def supprimer_element_a_lindice_p(li : list[int], p:int) -> None :
    for i in range(p,len(li)-1):
        li[i] = li[i+1]
    li.pop()

def compter_occurence(li : list[int], p:int) -> int :
    occurence = 0
    for i in range(len(li)):
        if li[i] == p:
            occurence += 1
    return occurence

def intersection(li1 : list[int], li2 : list[int]) -> list[int] :
    li3 = []
    for i in range(len(li1)):
        for j in range(len(li2)):
            if li1[i] == li2[j]:
                li3.append(li1[i])
    return li3

def intersection_sans_doublons(li1 : list[int], li2 : list[int]) -> list[int] :
    li3 = []
    for i in range(len(li1)):
        for j in range(len(li2)):
            if li1[i] == li2[j] and li1[i] not in li3:
                li3.append(li1[i])
    return li3
```

```python
if __name__ == "__main__":
    maliste : list[str]
    entier  : int
    autre : int
    tab : list[int]
    tab2 : list[int]
    tab3 : list[int]
    p:int
    element:int
    occurence:int
    maliste = ["un", "deux", "trois", "quatre"]
    entier = 10
    tab = [5,6,0,6,9,1,4]
    tab2 = [5,6,0,6,9,1,4,18,72,56,20,63]


    print("avant l'appel :")
    print(maliste," entier = ", entier)
    exPassage(maliste, entier)
    print("apres l'appel :")
    print(maliste," entier = ", entier)
    print("deuxième appel :")
    exPassage2(maliste, entier)
    print("apres l'appel :")
    print(maliste," entier = ", entier)
    print("on constate que la liste a été modifiée mais pas l'entier")

    #décalage vers la gauche
    print("décalage vers la gauche")
    print(maliste)
    décaler_tableau_vers_la_gauche(maliste)
    print(maliste)
```

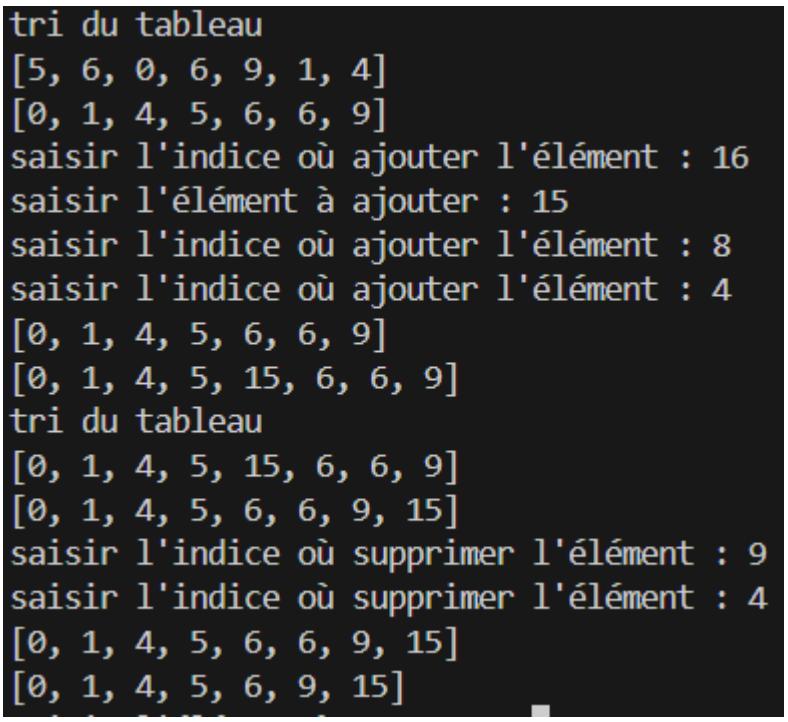
```
#décalage vers la droite
print("décalage vers la droite")
print(maliste)
décaler_tableau_vers_la_droite(maliste)
print(maliste)

#ajouter un élément à la fin
print("ajout d'un élément à la fin")
print(maliste)
maliste.append("cinq")
print(maliste)

#tri du tableau
print("tri du tableau")
print(tab)
tri_bulle(tab)
print(tab)

#ajouter un élément à l'indice p
p =int(input("saisir l'indice où ajouter l'élément : "))
element =int(input("saisir l'élément à ajouter : "))
while p<0 or p>len(tab):
    p =int(input("saisir l'indice où ajouter l'élément : "))
print(tab)
ajouter_element_a_lindice_p(tab,p,element)
print(tab)

#tri du tableau
print("tri du tableau")
print(tab)
tri_bulle(tab)
print(tab)
```

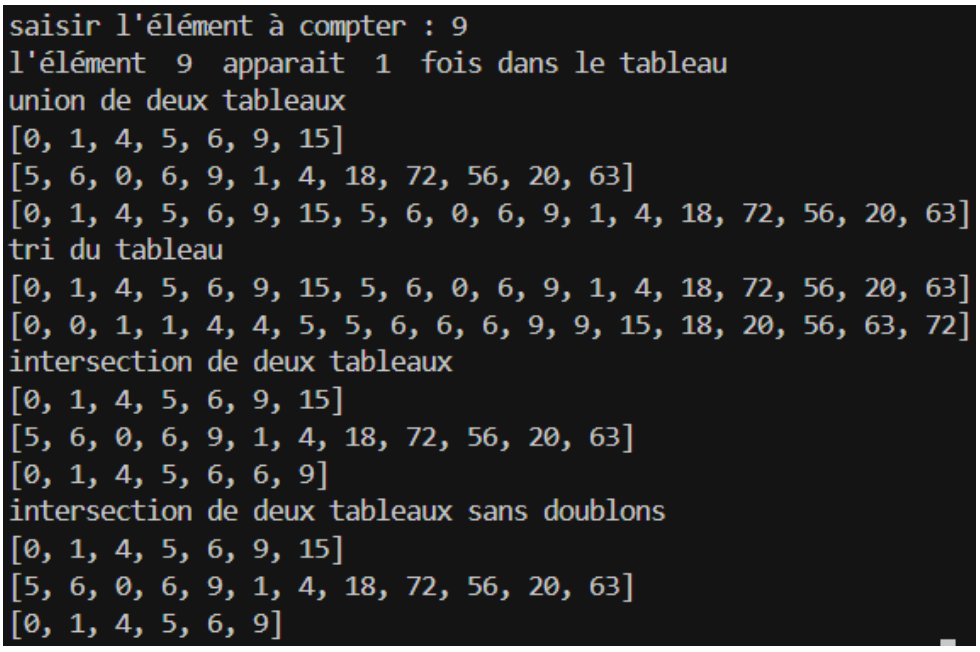
```
#suppresion d'un élément a l'indice p
p =int(input("saisir l'indice où supprimer l'élément : "))
while p<0 or p>len(tab):
    p =int(input("saisir l'indice où supprimer l'élément : "))
print(tab)
supprimer_element_a_lindice_p(tab,p)
print(tab)

#compter le nombre d'occurence d'un élément p
occurence = 0
p =int(input("saisir l'élément à compter : "))
occurence = compter_occurence(tab,p)
print("l'élément ",p," apparait ",occurence," fois dans le tableau")

#union de deux tableaux
print("union de deux tableaux")
print(tab)
print(tab2)
tab3 = tab + tab2
print(tab3)

#tri du tableau
print("tri du tableau")
print(tab3)
tri_bulle(tab3)
print(tab3)
```

```python
#intersection de deux tableaux
print("intersection de deux tableaux")
tab3 = []
print(tab)
print(tab2)
tab3 = intersection(tab,tab2)
print(tab3)

#intersection de deux tableaux sans doublons
print("intersection de deux tableaux sans doublons")
tab3 = []
print(tab)
print(tab2)
tab3 = intersection_sans_doublons(tab,tab2)
print(tab3)
```

jeux d'essais

```
avant l'appel :
['un', 'deux', 'trois', 'quatre']  entier =  10
modifications dans la procedure :
['zero', 'deux', 'trois', 'quatre']  valeur =  20
apres l'appel :
['zero', 'deux', 'trois', 'quatre']  entier =  10
deuxième appel :
modifications dans la procedure :
['zero', 'un', 'trois', 'quatre']  valeur =  20
apres l'appel :
['zero', 'un', 'trois', 'quatre']  entier =  10
on constate que la liste a été modifiée mais pas l'entier
décalage vers la gauche
['zero', 'un', 'trois', 'quatre']
['un', 'trois', 'quatre', 'zero']
décalage vers la droite
['un', 'trois', 'quatre', 'zero']
['zero', 'un', 'trois', 'quatre']
ajout d'un élément à la fin
['zero', 'un', 'trois', 'quatre']
['zero', 'un', 'trois', 'quatre', 'cinq']
```





Les tests effectués assurent que les indices demandés sont toujours présents et valides, garantissant ainsi un traitement précis des données. Chaque fonction et procédure est testée pour s'assurer qu'elle exécute correctement la tâche assignée et produit les résultats escomptés. Ces exigences de validation permettent d'assurer la fiabilité et la robustesse du code, en vérifiant que chaque composant fonctionne comme prévu et en évitant tout comportement inattendu. En procédant ainsi, on garantit la cohérence du programme et son alignement avec les spécifications initiales.