Algorithmie
Épreuve pratique
Recherche d'occurrences

1 Spécifications

1.1 Spécifications algorithme

Ecrire en pseudo code une fonction nommée **rech_ occ(T,elt)** qui prend en argument un tableau T et un entier elt, et qui renvoie False si elt n'est pas dans T, l'indice de elt dans T dans le cas inverse.
La méthode utilisée pour la recherche est une méthode séquencielle.

1.2 Spécifications implémentation Python

Ecrire en langage Python une fonction nommée **rech_ occ(T,elt)** qui prend en argument une liste Python T et un entier elt, et qui renvoie l'indice de elt dans T si elt est présent dans T, False sinon. La méthode utilisée pour la recherche est une méthode séquentielle.

2 Résolution

2.1 Un algorithme

☞ Ecrire un algorithme de recherche d'une occurrence utilisant une boucle POUR.
L'algorithme est une fonction qui prend en argument un tableau d'entiers et une valeur entière, et qui renvoie l'occurrence de cette valeur (FAUX si la valeur n'est pas présente dans le tableau).

```
1 FONCTION OCCURRENCE(T :tableau d'entiers, element :entier)
2 |  POUR i DE O A longueur(T)-1 FAIRE
3 |  |  SI T[i] = element ALORS
4 |  |  |  RENVOYER i
5 |  RENVOYER False
```

☞ Ecrire un algorithme de recherche d'une occurrence en utilisant une boucle TANT QUE.
L'algorithme est une fonction qui prend en paramètres un tableau d'entiers et une valeur entière,

et qui renvoie l'occurrence de cette valeur (FAUX si la valeur n'est pas présente dans le tableau)

```
1 FONCTION OCCURRENCE(T :tableau d'entiers, element :entier)
2 |  i ← 0
3 |  TANT QUE i ≤ longueur(T)-1 FAIRE
4 |  |  SI T[i] = element ALORS
5 |  |  |  RENVOYER i
6 |  |  i ← i + 1
7 |  RENVOYER False
```

2.2 Des implémentations possibles en Python

```
def rech_occurrence(T,elt):
    for i in range(len(T)):
        if T[i] == elt:
            return i
    return False
```

```
def rech_occurrence(T,elt):
    i = 0
    while i <= len(T) - 1:
        if T[i] == elt:
            return i
        i = i + 1
    return False
```

⚠ Il faut être capable de :

- Documenter la fonction
- Ecrire les pré-conditions
- Ecrire les post-conditions
- Donner un jeu de tests pertinents
- Utiliser des asserts pour vérifier ces tests
- Connaître la complexité temporelle
- Démontrer la terminaison de ces algorithmes

2.3 Complexité temporelle

Propriété 0.1

La complexité temporelle d'une recherche séquentielle est, dans le pire des cas, en $\mathcal{O}(n)$. On parle de complexité linéaire.

3 Exercices complémentaires

Exercice 0.1

1. Créer l'algorithme de la fonction **maximum(T)** qui reçoit en argument un tableau d'entiers et qui retourne le maximum du tableau.
2. Calculer la complexité temporelle de cet algorithme.
3. Implémenter cet algorithme en Python

Exercice 0.2

1. Créer l'algorithme de la fonction **minimum(T)** qui reçoit en argument un tableau d'entiers et qui retourne le minimum du tableau.
2. Calculer la complexité temporelle de cet algorithme.
3. Implémenter cet algorithme en Python

Exercice 0.3

1. Créer l'algorithme de la fonction **moyenne(T)** qui reçoit en argument un tableau d'entiers et qui retourne la moyenne des éléments du tableau.
2. Calculer la complexité temporelle de cet algorithme.
3. Implémenter cet algorithme en Python