1 Recherche d'un élément

Parcours séquentiel d'un tableau **par élément** :

```
def cherche(tab, e):
    """ Recherche si l'élément e est présente dans le tableau tab """
    for element in tab:
        if element == e:
            return True         # occurrence trouvée : on stoppe la recherche
    return False
```

Parcours séquentiel d'un tableau **par indice** :

```
def cherche(tab, e):
    """ Recherche si l'élément e est présente dans le tableau tab """
    for i in range(len(tab)):
        if tab[i] == e:
            return True         # occurrence trouvée : on stoppe la recherche
    return False
```

On propose de modifier légèrement ce code pour que la fonction renvoie la position de l'élément dans le tableau s'il est présent ou -1 dans le cas contraire.

```
def cherche_position(tab, e):
    """ Renvoie l'indice de 1ère occurrence de l'élément e s'il est présent
    dans le tableau tab, ou -1 dans le cas contraire """
    for i in range(len(t)):
        if tab[i] == e:
            return i            # 1ère occurrence trouvée : on stoppe la recherche
    return -1
```

2 Recherche d'un extremum

```
def maximum(tab):
    """ Recherche la valeur maximum dans le tableau tab """
    # préconditions : tableau non vide,
    # contenant des éléments tous comparables entre eux
    maxi = tab[0]                  # valeur maxi par défaut
    for i in range(1, len(tab)): # parcours par indice
        if tab[i] > maxi:
            maxi = tab[i]
    return maxi
```

On adaptera aisément cet algorithme à la recherche d'un minimum.

3 Calcul d'une moyenne

```
def cherche(tab):
    """ Calcule la moyenne des valeurs dans le tableau tab """
    # précondition : le tableau contient des valeurs numériques
    somme = 0 # somme des valeurs du tableau (nulle a priori)
    for nb in tab:          # parcours par élément
        somme = somme + nb
    # la moyenne est la somme divisée par le nb de valeurs
    return somme / (len(t))
```

4 Coût des algorithmes

Lorsqu'on écrit un algorithme, il est intéressant (indispensable ?) de réfléchir au coût de cet algorithme en fonction de la taille des données à traiter.
Dans les exemples présentés, les algorithmes développés **parcourent un tableau de façon séquentielle** dans son intégralité (même pour la recherche d'occurrence dans *le pire des cas*), élément par élément.
Si on double la taille du tableau, on double donc le nombre d'opérations à effectuer dans l'algorithme, et donc on double aussi le temps de calcul.

> Le coût de ces algorithmes est donc proportionnel à la taille du tableau, on dit que le coût de ces algorithmes est linéaire.