1 Recherche

1.1 Recherche dans une liste

Il suffit de parcourir toute la liste `L`. Dès que l'on tombe sur une valeur égale à `x`, on interrompt la boucle pour renvoyer `True`. A la fin de la boucle, si celle-ci n'a pas été interrompue, cela signifie que `x` n'est pas dans `L`. On peut alors renvoyer `False`.

```
def appartient(L,x):
    for i in range(len(L)):
        if L[i]==x:
            return True
    return False
```

Remarque 1

Une erreur ultra-classique consiste à écrire

```
def appartient(L,x):
    for i in range(len(L)):
        if L[i]==x:
            return True
        else:
            return False
```

Avec ce type de rédaction, la boucle s'interrompt dès le premier test et renvoie `True` si et seulement si le premier élément de la liste est différent de `x`, quels que soient les autres.

Exercice 1

Modifier l'algorithme précédent pour qu'il renvoie la liste des indices `i` tels que `L[i]` soient égaux à `x` (et donc une liste vide si `x` n'est pas dans `L`).

1.2 Recherche de maximums dans une liste

Là encore, on parcourt la liste en stockant dans une variable la plus grand valeur trouvée jusqu'à présent.

```
def maximum(L):
    m=L[0]
    for i in range(1,len(L)):
        if L[i]>m:
            m=L[i]
    return m
```

Si on veut récupérer les deux plus grandes valeurs, ou seulement la deuxième plus grande, il suffit d'utiliser deux compteurs en faisant attention aux mises à jour.

```
def deuxieme_maximum(L):
    if L[0]<L[1]:
        m1,m2=(L[1],L[0])
    else:
        m1,m2=(L[0],L[1])
    for i in range(2,len(L)):
        if L[i]>m1:
            m1,m2=(L[i],m1)
        elif L[i]>m2:
            m2=L[i]
    return m2
```

2 Comptage

Pour compter le nombre d'occurrences d'un tableau d'entiers, on commence par recherche le maximum `m` du tableau, puis on renverra une liste `r` de taille $m+1$ telle que `r[i]` contiendra le nombre d'occurrences de l'entier `i`. La fonction s'écrit alors ainsi :

```
def comptage_entiers(L):
    m=max(L)
    r=[0 for i in range(m+1)]
    for x in L:
        r[x]+=1
    return r
```

Si l'on veut compter des éléments quelconques (et pas seulement des entiers), la structure de liste n'est plus adaptée pour le résultat. Il faut alors utiliser un dictionnaire. En informatique, un dictionnaire (ou table d'association) est une structure de donnée permettant des stocker de couples de la forme (clé, valeur), les clés étant nécessairement deux à deux distinctes. En Python, cette structure de donnée est explicitement au programme de deuxième année, pour une utilisation en programmation dynamique. On peut cependant s'en servir dès maintenant en mode « boite noire » pour représenter des graphes dont les sommets sont quelconques (plus précisément pas forcément entiers) par listes d'adjacence. Voici les commandes de bases :

- **Création :** Le dictionnaire vide est crée par l'instruction {}. On peut également le créer avec initialement quelques couples.

  ```
  D = {3 : "maths", 4: "vincent", "z": 5}
  ```

  L'élément à gauche de chaque : est la clé, l'élément à droite est sa valeur. Notons qu'en Python, les clés ne sont pas forcément du même type (ce n'est pas le cas dans tous les langages de programmation).

- **Ajout / modification / suppression :** L'ajout d'un couple (clé,valeur) s'écrit de la même façon que l'on modifie une valeur dans une liste.

  ```
  D[2] = "toto"
  ```

  Si la clé (ici 2) n'existe pas, cette commande crée une nouvelle entrée dans le dictionnaire. Si la clé existe déjà, la valeur est modifiée. La suppression se fait là encore de la même manière que pour une liste.

  ```
  del D["z"]
  ```

- **Récupération de clés/valeurs :** Pour récupérer la valeur associée à une clé, on utilise la méthode `get`. Si la clé est absente, cette fonction ne renvoie rien. On peut également utiliser la même syntaxe que pour les listes, mais attention, on obtient cette fois une erreur si la clé n'existe pas. On dispose également des méthodes `keys` (resp. `values`) pour récupérer la liste des clés (resp. valeurs).

  ```
  D.get(2); D[2];  D.keys(); D.values()
  ```

  Tester la présence d'un élément `x` dans un dictionnaire `d` peut s'écrire comme pour les listes à l'aide du raccourci `if x in d`. Dans la plupart des implémentations (dont les détails ne sont pas au programme), la structure est optimisée de façon à ce que cette opération soit de coût constant.

La fonction de comptage des éléments d'une liste d'objets de nature quelconque peut alors s'écrire de la manière suivante :

```
def comptage(L):
    d={}
    for x in L:
        if x in d:
            d[x]+=1
        else:
            d[x]=1
    return d
```