1) Dans le cas de l'algorithme des k plus proches voisins, on est amené à déterminer une valeur majoritaire relative dans un tableau L de k éléments à valeurs dans un ensemble E (qui représente les classes des différents éléments). Proposer une fonction `majoritaire(L)` qui étant donné un tableau L de longueur $k \geq 1$ renvoie en temps linéaire $O(n)$ une valeur majoritaire, et s'il en existe plusieurs, renvoie la valeur majoritaire qui apparait la première dans le tableau.

Remarque : On pourra éventuellement supposer $E \subset [\![0, p-1]\!]$ ou bien utiliser des dictionnaires.

2) On considère une liste P de n points de $\mathbb{R}^d$ codés par des listes de d flottants, représentant des éléments de référence dont on connaît la classe. On considère un point $a \in \mathbb{R}^d$.

a) Écrire une fonction `distances(a,P)` qui renvoie la,liste triée `D` des distances de a aux points de la liste P. On pourra utiliser la fonction `sorted()` qui renvoie la liste triée de la liste donnée en argument.

b) On se donne une distance r de référence. Écrire une fonction `nombreVoisins(D,r)` qui renvoie en temps $O(\log n)$ le nombre k de voisins qui sont à une distance de a inférieure ou égale à r.

Corrigé

1)

```
def majoritaire(L) :
    dico = {}
### les clés sont les classes et la valeur d'une clé est la liste [nombre d'occurrences,première occurrence]
    for k in range(len(L)) :
        if L[k] not in dico :
            dico[L[k]] = [1,k]
        else :
            dico[L[k]][0] += 1
###
    classeMaj = L[0]
    for classe in dico :
        if ( dico[classe][0] < dico[classeMaj][0]
           or (dico[classe][0] = dico[classeMaj][0]
           and dico[classe][1] < dico[classeMaj][1]) ) :
        classeMaj = classe
    return classeMaj
```

2) a)

```
def distancePoints(a,b) :
    d = len(a)
```

```
    for i in range(d) :  s = s + (a[i]-b[i])**2
    return s ** 0.5

def distances(a,P) :
    return sorted([distancePoints(a,b) for b in P])
```

b) On procède par dichotomie.

```
nombreVoisins(D,r) :
    i = -1 ; j = len(D)    ### avec comme convention D[0] ≤ r et D[j] > r
    while j-i > 1 :
        k = (i+j)//2  ### on a toujours i ≤ k < j et D[i] ≤ r < D[j]
        if D[k] > r :  j = k
        else :  i  = k
    return j     ###    on a ici  j = i + 1, nombre de termes entre 0 et i inclus
```