TP 3 : Listes

Exercice 1. Écrire une fonction `Maxliste` qui prend en argument une liste `L` d'entiers et qui renvoie la plus grande valeur de cette liste ainsi que la position de la première occurrence de ce maximum (sans utiliser les fonctions `max` ni la méthode `index`).

Réponse :

Exercice 2. Écrire une fonction `SecondMaxliste` qui prend en argument une liste `L` d'entiers et qui renvoie la deuxième plus grande valeur de cette liste ainsi que la position de la première occurrence de cette valeur (sans utiliser les fonctions `max` ni la méthode `index`).

Réponse :

Exercice 3. (Liste des termes d'une suite)

1. Soit (u_n) la suite définie par $u_0 = 0$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = u_n^2 + 1$.
   Écrire un script qui construit puis renvoie la liste de tous les u_k tels que $u_k \leqslant 1000$.
2. Soit (u_n) la suite définie par $u_0 = 0$, $u_1 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+2} = 3u_{n+1} - u_n$.
   Écrire une fonction `suiteU` qui prend en argument un entier `n` et qui renvoie la liste des n premiers termes de la suite.

Réponse :

Exercice 4. Écrire une fonction `inverse` qui prend en argument une liste `L` et qui renvoie une liste avec les mêmes élément rangé dans le sens inverse.

Réponse :

Exercice 5. (Comptage) Dans une liste de nombres, on dit qu'une **montée** a lieu lorsque un nombre est strictement plus grand que le nombre qui le précède. Par exemple, pour `L=[5,2,9,1,4,2,4,6]`, il y a 4 montées (2<9, 1<4, 2<4 et 4<6).

Écrire une fonction `nbMont` qui prend en argument une liste `L` et qui renvoie le nombre de montées de `L`.

Réponse :

Exercice 6. Écrire une fonction `Repete` qui prend une liste `L` et qui renvoie `True` si la fonction contient deux éléments identiques et `False` sinon.

Réponse :

Exercice 7. (Recherche par dichotomie dans une liste triée)

1. Écrire une fonction `appartient1` qui prend en argument un élément `x` et une liste `L` et qui parcours la liste `L` et qui renvoie `True` si `x` appartient à `L` et `False` sinon.
2. Écrire une fonction `appartient2` qui fait la même chose que la précédente sans écrire un parcours de la liste `L`.
3. Nous allons maintenant supposer que `L` est une liste d'entiers rangés dans l'ordre croissant et que `x` est un entier lui aussi. Dans ce cas, on peut être plus efficace pour déterminer si `x` appartient à `L`. On procéder par dichotomie, c'est-à-dire que nous allons encadrer l'indice où devrait se situer `x` dans un intervalle $[\![a, b]\!]$ dont on divisera la taille par deux à chaque ittération de la boucle de la manière suivante :
   — On regarde l'élément du milieu de la liste `L[a:b+1]` ;
   — Si l'élément est égal à `x`, alors on s'arrête (parce qu'on a gagné) ;
   — Si l'élément est strictement plus grand que `x`, on remplace `b` par l'indice du milieu ;
   — Sinon on remplace `a` par l'indice du milieu.

   On continue jusqu'à ce que a et b soit égaux.

   Écrire une fonction `appartient3` qui fait la même chose que les précédentes mais qui procède par dichotomie.
4. Comparer la vitesse des différentes méhodes. Pour cela, on créera la liste `L` contenant tout les entiers de 1 à 1 000 000 et cherchera 10 000 fois de suite la valeur `x` égale 900 000 dans cette liste (cette recherche n'est pas très intéressante mais elle nous permet de tester nos programmes).

   Pour connaitre le temps qu'a pris l'opération, on utilisera la fonction `time` de la bibliothèque `time` qui renvoie la valeur de l'horloge interne de votre ordinateur.

Réponse :