Tp ALGORITHME des k plus proches voisins

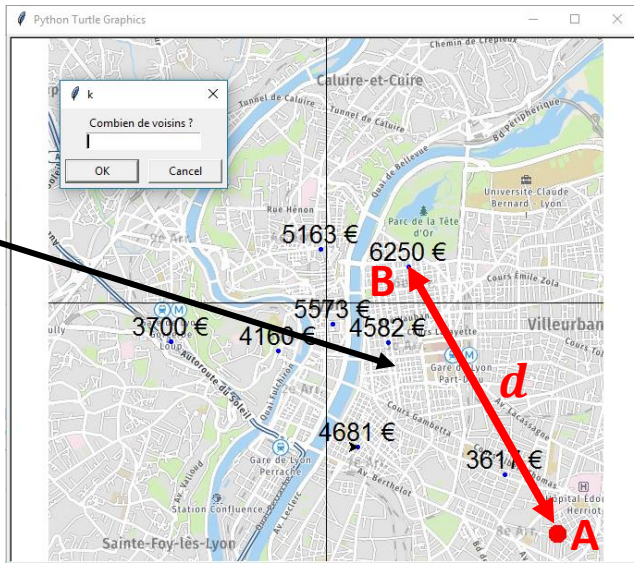
TRAVAIL A FAIRE :

PARTIE B : 1ère étape : « Pour chacun des biens de la liste *liste_transactions*, utiliser les coordonnées x et y pour calculer la distance géométrique qui le sépare de la position de *emplacement_bien* »

La distance géométrique $d = AB$ entre 2 points $A(x_A\ , y_A)$ et $B(x_B\ , y_B)$ est donnée par la relation :

$d = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$. En python, cela donne :

```
d = sqrt((xb-xa)**2 + (yb-ya)**2)
```



1- Compléter la fonction *distance()* qui prend en arguments les coordonnées de $A(x_A\ , y_A)$ et $B(x_B\ , y_B)$ et **renvoie la valeur de d .**

```
6  def distance(xa,ya,xb,yb) :
7      """
8          Renvoie la distance entre les points de coordonnées (xa,ya)
9           et (xb,yb)
10     """
11     d = sqrt((xb-xa)**2 + (yb-ya)**2)
12     return d
```

2- Compléter la fonction *complete_distance()* qui prend en argument la liste *liste_transactions* et la liste *emplacement_bien* et renvoie la même liste *liste_transactions* en ayant remplacé pour chaque élément i , le ***None*** de *liste_transactions[i] par la distance d qui sépare ce bien de celui à estimer.*

```
15 def complete_distance(liste,point) :
16     """
17         Renvoie la liste mise en argument. Le champs liste[i][3] prend la valeur
18         de la distance d entre point et l'emplacement du bien de liste[i]
19     """
20     xa , ya = point
21     for i in range(len(liste)) :
22         x = liste[i][0]
23         y = liste[i][1]
24         d = distance(xa,ya,x,y)
25         liste[i][3] = d
26     return liste
```

3- Appeler la fonction *complete_distance()* dans le programme principal afin que *liste_transactions* soit modifié et complété.

```
53 # --------------- MAIN ---------------------
54 liste_transactions = bdd_biens_deja_vendus()
55 emplacement_bien = [250,-250]
56 k = visualiser(liste_transactions,emplacement_bien,None,None)
57
58 liste_transactions = complete_distance(liste_transactions,emplacement_bien)
```

PARTIE C : 2nde étape : « Rechercher dans *liste_transactions* les k plus proches voisins de *emplacement_bien* ».

4- Compléter la fonction *tri()* qui prend en argument *liste_transactions* et tri ses éléments *liste_transactions[i]* par ordre croissant de la valeur de la distance contenu dans *liste_transactions[i][3]* . Ne pas utiliser la fonction *sorted()* de python : réutiliser **en les adaptant à cette nouvelle situation**, le codes tri sélection vu en TP 2.

liste_transactions_trie[0] sera parmi les 8 listes ci-contre, celle qui aura la valeur de la distance (qui remplace **None**) la plus petite.

```
[ [-52 , -52 , 4160 , None],\
  [-168 ,-42 , 3700 , None],\
  [-6 ,58 , 5163 , None],\
  [7 ,-23 , 5573 , None],\
  [89 ,39 , 6250 , None],\
  [67 ,-43 , 4582 , None],\
  [193 ,-186 , 3617 , None],\
  [34 ,-156 , 4681 , None] ]
```

Info utile : pour échanger le contenu de 2 variables a et b, on peut écrire en python : a , b = b , a
Si ces variables sont des listes de listes, on peut faire la même chose. Par exemple, si $\ell = [\,[a1, b1]\,,\, [a2, b2]\,]$
L'exécution de : $\ell[0]\,, \ell[1] = \ell[1]\,, \ell[0]$ permet d'obtenir : $\ell = [\,[a2, b2]\,,\, [a1, b1]\,]$
L'exécution de : $a = \ell[0]$, puis $\ell[0] = l[1]$ puis $\ell[1] = a$ permet d'obtenir aussi le même résultat.

```
29 def tri(liste):
30     """
31         Renvoie liste triée par rapport à la valeur de la distance contenues
32         dans liste[i][3]. On utilise un algorithme de tri par sélection
33     """
34     for i in range(len(liste)) :
35         indice = i
36         for j in range(i , len(liste)) :
37             if liste[j][3] < liste[indice][3] : indice = j
38         liste[i] , liste[indice] = liste[indice] , liste[i]
39     return liste
```

Appeler ensuite la fonction *tri()* dans le programme principal afin que la liste retournée et donc triée ait le nom : *liste_transactions_trie*

```
58 liste_transactions = complete_distance(liste_transactions,emplacement_bien)
59 liste_transactions_trie = tri(liste_transactions)
```

Tp ALGORITHME des k plus proches voisins

PARTIE D : 3ième étape : « Pour les k plus proches voisins, calculer la moyenne de leur prix au m² . Cette moyenne sera celle du bien que l'on veut estimer».

5- Compléter la fonction *k_voisins()* qui prend en argument la valeur de k saisie par l'utilisateur et la liste *liste_transactions_trie*. Cette fonction renvoie la moyenne de prix au m² des k premiers éléments de la liste *liste_transactions_trie*.

```
42  def k_voisins(k,liste) :
43      """
44          Renvoie la moyenne des k premieres valeurs V contenues
45          dans liste
46      """
47      s = 0
48      for i in range(k) :
49          s += liste[i][2]
50      return s / k
```

6- Appeler la fonction *k_voisins()* dans le programme principal, suivi de l'appel de la fonction *visualiser()* afin d'afficher sur la carte l'estimation du bien en € et de souligner les k voisins pris en compte :

```
prix_moyen = k_voisins(k,liste_transactions_trie)

visualiser(liste_transactions_trie,emplacement_bien,k,prix_moyen)
```

PARTIE E : Conclusion

Faire des exécutions pour déterminer le prix moyen au m² pour un bien à estimer dont l'emplacement et le nombre de voisins à prendre en compte est défini sur le tableau ci-dessous :

x	*250*	*250*	*250*	*125*	*125*	*- 250*	*-250*
y	*-250*	*-250*	*-250*	*200*	*200*	200	*-250*
k	$k = 1$	$k = 3$	$k = 7$	$k = 1$	$k = 3$	$k = 4$	$k = 2$
Prix estimé au m² en €	3617,0 €	4293,0 €	4861,0 €	6250,0 €	5332,0 €	4349,0 €	3930,0 €