Polynésie – 2023 – sujet1 - Correction

Exercice 2 (5 points)

1.

a. Rôle de l'instruction `from math import sqrt`

Cette instruction permet d'importer la fonction `sqrt` du module `math`.

👉 **Commentaire :** *Elle permet d'utiliser directement `sqrt()` sans écrire `math.sqrt()`.*

---

b. Pourquoi `0.1 + 0.2 == 0.3` renvoie False

Les nombres flottants sont représentés en binaire de manière approximative.

Ainsi :

```
0.1 + 0.2
```

donne en réalité une valeur très proche de 0.3 mais pas exactement 0.3.

👉 **Commentaire :** *Les flottants sont codés en binaire, ce qui entraîne des erreurs d'arrondi.*

---

c. Explication de l'erreur sur le tuple

Un tuple est **immutable**.

L'instruction :

```
point_A[0] = 2
```

tente de modifier un élément du tuple.

👉 **Commentaire :** *Un tuple ne peut pas être modifié après sa création. Il faut créer un nouveau tuple.*

2.

a. Compléter le constructeur de la classe Segment

Distance entre deux points :

$$\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Ligne à compléter :

```
self.longueur = sqrt((point2[0] - point1[0])**2 + (point2[1] - point1[1])**2)
```

👉 **Commentaire :** *On applique directement la formule de la distance euclidienne.*

---

b. Fonction liste_segments complétée

```python
def liste_segments(liste_points):
    n = len(liste_points)
    segments = []
    for i in range(n):
        for j in range(i+1, n):
            seg = Segment(liste_points[i], liste_points[j])
            segments.append(seg)
    return segments
```

👉 **Commentaire :** *On commence j à i+1 pour éviter les doublons et ne pas construire [AB] et [BA].*

---

c. Longueur de la liste segments

Nombre total de segments : $\sum_{i=1}^{n-1} i$

Soit :

$$\frac{n(n-1)}{2}$$

👉 **Commentaire :** *On construit toutes les paires possibles de points.*

---

d. Complexité en temps de liste_segments

Deux boucles imbriquées dépendant de n.

Complexité : O(n^2)

👉 **Commentaire :** *On parcourt toutes les paires de points. La complexité est quadratique.*

3.

a. Fonction plus_court_segment (diviser-pour-régner)

```
def plus_court_segment(liste_segments):
    if len(liste_segments) == 1:
        return liste_segments[0]

    gauche = moitie_gauche(liste_segments)
    droite = moitie_droite(liste_segments)

    min_gauche = plus_court_segment(gauche)
    min_droite = plus_court_segment(droite)

    if min_gauche.longueur < min_droite.longueur:
        return min_gauche
    else:
        return min_droite
```

👉 **Commentaire :** *On applique strictement la méthode diviser-pour-régner : découpage, récursion, comparaison.*

4.

a. Construction de nuage_points

Points :

A(3,4)
B(2,3)
C(-3,-1)

Instruction :

```
nuage_points = [(3,4), (2,3), (-3,-1)]
```

👉 **Commentaire :** *Chaque point est représenté par un tuple.*

---

b. Affichage des deux points les plus proches

```
segments = liste_segments(nuage_points)
plus_proche = plus_court_segment(segments)

print(plus_proche.p1, plus_proche.p2)
```

👉 **Commentaire :** *On génère tous les segments puis on recherche récursivement le plus court.*
