Métropole – 2024 – sujet2 - Correction

Exercice 3 (8 points)

Partie A – Analyse des classes Piste et Domaine

1. Attributs de la classe `Piste`

Les attributs sont :

- `nom` : chaîne de caractères (`str`)
- `longueur` : nombre réel (`float`)
- `denivele` : nombre entier ou réel (`int` ou `float`)
- `couleur` : chaîne de caractères (`str`)
- `ouverte` : booléen (`bool`)

👉 *Commentaire : il fallait identifier uniquement les attributs d'instance (ceux définis dans `__init__`) et préciser leur type logique.*

---

2. Méthode `set_couleur`

```
def set_couleur(self):
    if self.denivele >= 100:
        self.couleur = 'noire'
    elif self.denivele >= 70:
        self.couleur = 'rouge'
    elif self.denivele >= 40:
        self.couleur = 'bleue'
    else:
        self.couleur = 'verte'
```

👉 *Commentaire : l'ordre des conditions est essentiel. On commence par la valeur la plus grande pour éviter qu'un dénivelé ≥ 100 soit classé rouge ou bleu.*

---

3. Type de `lievre_blanc.get_pistes()`

Réponse correcte : **Proposition D : une liste d'objets de type `Piste`.**

👉 *Commentaire : la boucle `for piste in lievre_blanc.get_pistes()` montre que chaque élément est un objet sur lequel on peut appeler `set_couleur()`. Il s'agit donc d'objets et non de chaînes.*

---

4. Fermeture des pistes vertes

```
for piste in lievre_blanc.get_pistes():
    if piste.couleur == 'verte':
        piste.ouverte = False
```

👉 *Commentaire : on modifie directement l'attribut `ouverte`. On pouvait aussi utiliser un accesseur si la classe en propose un.*

---

5. Fonction `pistes_de_couleur`

```
def pistes_de_couleur(lst, couleur):
    resultat = []
    for piste in lst:
        if piste.couleur == couleur:
            resultat.append(piste.get_nom())
    return resultat
```

👉 *Commentaire : on renvoie une liste de noms, pas une liste d'objets. Attention à bien utiliser `append()`.*

---

6. Fonction `semi_marathon`

```
def semi_marathon(L):
    distance = 0
    liste_pistes = lievre_blanc.get_pistes()
    for nom in L:
        for piste in liste_pistes:
            if piste.get_nom() == nom:
                distance = distance + piste.get_longueur()
    return distance > 21.1
```

👉 *Commentaire : on initialise `distance` à 0, on compare les noms, puis on additionne les longueurs. La consigne précise "strictement supérieure".*

Partie B – Graphes

7. Longueur de la piste de 'E' à 'F'

```
print(domaine['E']['F'])
```

👉 *Commentaire : on accède au dictionnaire principal puis au dictionnaire des voisins.*

---

8. Fonction `voisins`

```python
def voisins(G, s):
    return list(G[s].keys())
```

👉 *Commentaire : les voisins correspondent aux clés du sous-dictionnaire associé au sommet.*

---

9. Fonction `longueur_chemin`

```python
def longueur_chemin(G, chemin):
    precedent = chemin[0]
    longueur = 0
    for i in range(1, len(chemin)):
        longueur = longueur + G[precedent][chemin[i]]
        precedent = chemin[i]
    return longueur
```

👉 *Commentaire : on additionne les distances entre sommets consécutifs. La variable `precedent` permet de mémoriser le sommet précédent.*

---

10. Pourquoi `parcours` est récursive ?

La fonction est récursive car **elle s'appelle elle-même** pour explorer les chemins à partir des voisins du sommet courant.

👉 *Commentaire : la définition d'une fonction récursive repose sur un appel à elle-même avec un problème plus petit.*

---

11. Fonction `parcours_dep_arr`

```
def parcours_dep_arr(G, depart, arrivee):
    tous_les_chemins = parcours(G, depart)
    resultat = []
    for chemin in tous_les_chemins:
        if chemin[-1] == arrivee:
            if chemin not in resultat:
                resultat.append(chemin)
    return resultat
```

👉 *Commentaire : on filtre les chemins qui se terminent par `arrivee` et on évite les doublons.*

---

12. Fonction `plus_court`

```
def plus_court(G, depart, arrivee):
    chemins = parcours_dep_arr(G, depart, arrivee)
    plus_court_chemin = chemins[0]
    min_longueur = longueur_chemin(G, plus_court_chemin)

    for chemin in chemins:
        l = longueur_chemin(G, chemin)
        if l < min_longueur:
            min_longueur = l
            plus_court_chemin = chemin

    return plus_court_chemin
```

👉 *Commentaire : on compare toutes les longueurs. Cette méthode est dite "force brute" car elle teste tous les chemins possibles.*

---

13. Discussion sur le critère choisi

Choisir la distance minimale ne garantit pas forcément le temps minimal.

En ski de fond, la vitesse dépend :

- du dénivelé,
- de la difficulté (couleur),
- de l’état de la neige,
- de la fatigue.

Un meilleur critère serait :

- minimiser le **temps estimé** (distance / vitesse estimée),
- ou utiliser un poids tenant compte du dénivelé.

👉 *Commentaire : on attend ici une réflexion sur la modélisation : distance minimale ≠ temps minimal.*
