Asie - juin 2024 - sujet 1 (corrigé)

Exercice 1 (Programmation Python, POO et algorithmique)

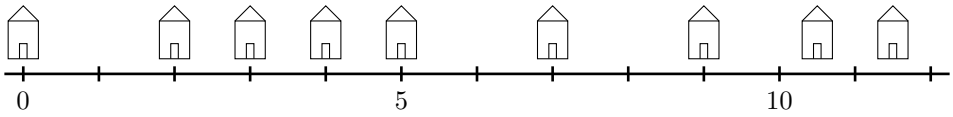
1. Le code suivant convient :

```
m1 = Maison(1)
m2 = Maison(3.5)
```

2. Le code suivant convient :

```
a = Antenne(2.5, 1)
```

3. On a le schéma suivant :



4. Le code suivant convient :

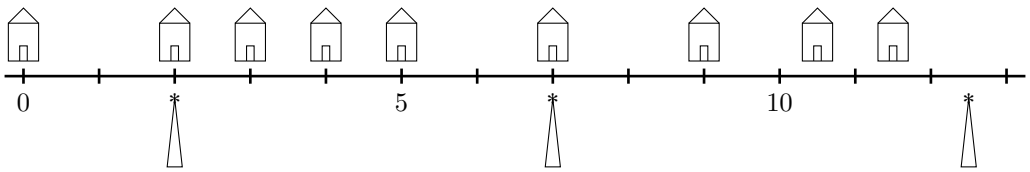
```
def creation_rue(pos):
    pos.sort()
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(m)
    return maisons
```

5. Le code suivant convient :

```
def couvre(self, maison):
    return abs(maison.get_pos_maison() - self.get_pos_antenne()) <= self.get_rayon()
```

6. On obtient la liste des positions des antennes suivante : `[0, 3, 7, 10.5]`

7. On a le schéma suivant :



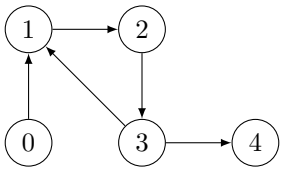
8. La fonction suivante convient :

```
def strategie_2(maisons, rayon):
    ''' Prend en paramètre une liste de maisons et le rayon
        d'action des antennes et renvoie une liste d'antennes
    '''
    antennes = [Antenne(maisons[0].get_pos_maison()+rayon, rayon)]
    for m in maisons[1:]:
        if not antennes[-1].couvre(m):
            antennes.append(Antenne(m.get_pos_maison()+rayon, rayon))
    return antennes
```

9. Dans les deux stratégies, on est obligé de tester chacune des n maisons de la rue. En conséquence, les deux fonctions sont en $O(n)$ et donc ont un coût linéaire.

Exercice 2 (Graphes, programmation Python et piles)

1. Il s'agit d'un graphe orienté.
2. - réaliser la tâche (f) puis la tâche (g) : c'est possible car l'arc (f, g) existe
   - réaliser la tâche (g) puis la tâche (f) : ce n'est pas possible car l'arc (g, f) n'existe pas
   - réaliser la tâche (i) puis la tâche (j) : c'est possible car les deux tâches ne dépendent pas l'une de l'autre
   - réaliser la tâche (j) puis la tâche (i) : c'est possible car les deux tâches ne dépendent pas l'une de l'autre
3. Pour pouvoir réaliser la tâche (k), il faut avoir préalablement réaliser les tâches (a), (c), (h), (i) et (j).
4. La figure 1 ne contient pas de cycle.
5. Un ordre possible : $0-2-1-3-5-4$
6. Le graphe suivant convient :



7. On ne peut pas effectuer toutes les tâches en respectant les dépendances car ce graphe contient un cycle.
8. On a le tableau suivant :

Appel `mystere`	variable `ouverts`	variable `fermes`
Avant l'appel `mystere`	`[F,F,F,F,F]`	`[F,F,F,F,F]`
`mystere(M,1,5,[F,F,F,F,F],[F,F,F,F,F],None)`	`[F,T,F,F,F]`	`[F,F,F,F,F]`
`mystere(M,2,5,[F,T,F,F,F],[F,F,F,F,F],None)`	`[F,T,T,F,F]`	`[F,F,F,F,F]`
`mystere(M,3,5,[F,T,T,F,F],[F,F,F,F,F],None)`	`[F,T,T,T,F]`	`[F,F,F,F,F]`
`mystere(M,1,5,[F,T,T,T,F],[F,F,F,F,F],None)`	`[F,T,T,T,F]`	`[F,F,F,F,F]`

   La fonction renvoie `False`
9. La fonction `mystere` réalise un parcours en profondeur pour détecter un cycle. Il renvoie `False` dans ce cas.
10. A l'issue des quatre premières instructions, la pile `essai` contient les valeurs `3`, `2` et `10` dans cet ordre, `10` étant au sommet de la pile. On dépile ensuite deux fois et donc la variable `elt` contient la valeur `2`
11. Il faut écrire l'instruction `resultat.empiler(s)` à la ligne 24 de la fonction `mystere`

Exercice 3 (Programmation Python, POO, bases de données et SQL)

Partie A

1. L'instruction suivante convient :

```
personneA = Personne(112, 'LESIEUR', 'Isabelle', 1982, 2005)
```

2. L'instruction suivante convient :

```
personneA.num_badge
```

3. Le code suivant convient :

```
def annee_anciennete(self):
    return 2024 - self.annee_entree
```

4. Le code suivant convient :

```
def ajouter(self, personne):
    self.liste.append(personne)
```

5. Le code suivant convient :

```
def effectif(self):
    return len(self.liste)
```

6. Le code suivant convient :

```
def donne_nom(self, num):
    for elt in self.liste:
        if elt.num_badge == num:
            return elt.nom
    return None
```

7. Le code suivant convient :

```
def nb_personne_honneur(self, annee):
    nb = 0
    for elt in self.liste:
        if annee - elt.annee_entree == 10:
            nb += 1
    return nb
```

8. Le code suivant convient :

```
def plus_anciens(self):
    L = [self.liste[0]]
    for i in range(1, len(self.liste)):
        if self.liste[i].annee_anciennete() > L[0].annee_anciennete():
            L = [self.liste[i]]
        elif self.liste[i].annee_anciennete() == L[0].annee_anciennete():
            L.append(self.liste[i])
    return [elt.num_badge for elt in L]
```

Partie B

9. Cette requête renvoie les noms et prénoms de toutes les personnes travaillant dans le centre 2.
10. La requête suivante convient :

```
UPDATE Personnel SET num_centre = 3 WHERE num_badge = 135
```

11. Cela permet d'éviter des redondances d'informations et permet de faciliter les ajouts, les suppressions et les mises à jours de données.

12. Les tables `Centre` et `Personnel` sont mises en relation à l'aide de l'attribut `num` qui est la clé primaire de `Centre` et de l'attribut `num_centre` qui est la clé étrangère associée dans la table `Personnel`.
13. La requête suivante convient :

```
SELECT Personnel.nom
FROM Personnel
JOIN Centre
ON Personnel.num_centre = Centre.num
WHERE Centre.ville = 'Lille' AND 2015 <= Personnel.annee_debut <= 2020
```

14. Cette requête provoque une erreur car on supprime la valeur `1` de la clé primaire `num` de `Centre` et il existe encore dans la table `Personnel` des valeurs égales à `1` dans la clé étrangère `num_centre` associée. Par conséquent, la contrainte d'inclusion n'est pas respectée. Par ailleurs, le mot-clé `DELETE` ne peut être suivi du symbole `*`.