Exercice 1 :

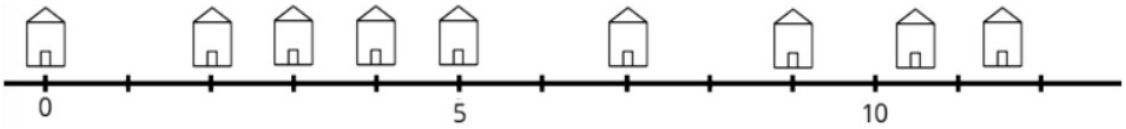
1. Création des deux instances :

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

2. Création de l'antenne :

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

3. Schéma :



4.

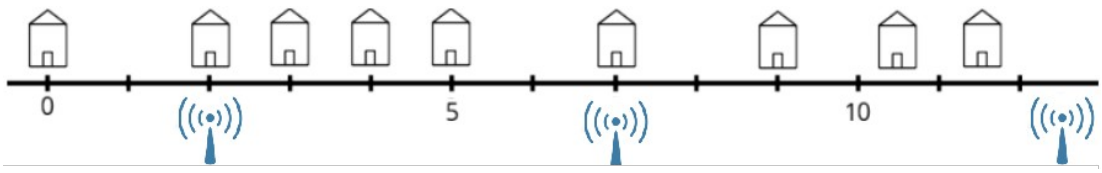
```
def creation_rue(pos):
    pos.sort()
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(m)
    return maisons
```

5.

```
# Méthode à ajouter dans la classe Antenne
def couvre(self, maison):
    distance = abs(self.get_pos_antenne() - maison.get_pos_maison())
    return distance <= self.get_rayon()
```

6. Cette suite d'instruction renvoie les positions des antennes selon la stratégie 1 qui consiste à place une antenne sur la première maison puis vérifie si la maison suivante est couverte par celle-ci, sinon ajoute une antenne au niveau de la maison suivante et ainsi de suite.
Le résultat est : [0, 3, 7, 10.5]

7.



8.

```
def strategie_2(maisons, rayon):
    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. Quelque soit la stratégie, il faut parcourir la liste des maisons (taille du problème : n) ainsi le coût est linéaire : O(n).

Exercice 2 :

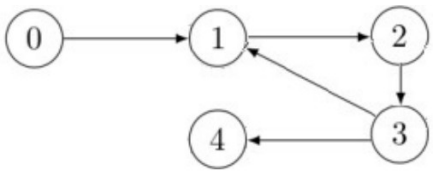
1. C'est un graphe orienté.

2. - réaliser la tâche (f) puis la tâche (g) : Possible
   - réaliser la tâche (g) puis la tâche (f) : Impossible
   - réaliser la tâche (i) puis la tâche (j) : Possible
   - réaliser la tâche (j) puis la tâche (i) : Possible

3. Pour réaliser la tâche (k), il faut avoir réaliser les tâches (a), (h), (i), (c) et (j).

4. Il n'y a pas de cycle.

5. Il est possible de réaliser les tâche dans l'ordre : 2, 0 puis 1 et 3 puis 5 puis 4.

6. 

7. Il est impossible de trouver un ordre car le graphe possède un cycle 1-2-3.

8.
```
M = [[0, 1, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 1, 0],
     [0, 1, 0, 0, 1],
     [0, 0, 0, 0, 0]]
```

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 variable ok vaut `False` car le dernier appel de `mystere` du tableau renvoie `False` et chaque appel de la pile d'appel récursif renvoie `False`.

9. La fonction `mystere` renvoie `False` à chaque fois qu'un cycle est détecté dans le graphe.

10. Après l'exécution des instructions la variable `elt` est associée à la valeur 2.

11. La fonction `mystere` réalise un parcours en profondeur du graphe. Dès qu'il arrive sur une étape finale (bout d'une branche), il la marque comme terminée et remonte dans la branche.
Il faut ainsi empiler cette étape afin qu'elle soit réalisée en dernier :
`resultat.empiler(s)`

Exercice 3 : Partie A :

1. `personneA = Personne(112, "LESIEUR", "Isabelle", 1982, 2005)`

2. `personneA.num_badge`

3.
```
        # Méthode de la classe Personne
        def annee_anciennete(self):
            return 2024 - self.annee_entree
```

4.
```
        # Méthode de la classe Personnel
        def ajouter(self, personne):
            self.liste.append(personne)
```

5.
```
        # Méthode de la classe Personnel
        def effectif(self):
            return len(self.liste)
```

6.
```
        # Méthode de la classe Personnel
        def donne_nom(self, num):
            for elt in self.liste:
                if elt.num_badge == num:
                    return elt.nom
            return None
```

7.
```
def nb_personne_honneur(self, annee):
    total = 0
    for elt in self.liste:
        if elt.annee_anciennete() == 10:
            total += 1
    return total
```
ou
```
def nb_personne_honneur(self, annee):
    return sum([1 for elt in self.liste if elt.annee_anciennete() == 10])
```

8. En considérant que l'on ne retient que ceux ayant la même ancienneté maximale :

Solution 1 :
```
def plus_anciens(self):
    maxi = 0
    for p in self.liste:
        if p.annee_anciennete() >= maxi:
            maxi = p.annee_anciennete()
    liste = []
    for p in self.liste:
        if p.annee_anciennete() == maxi:
            liste.append(p.num_badge)
    return liste
```

Solution 2 :
```
def plus_anciens(self):
    liste = []
    maxi = 0
    for p in self.liste:
        if p.annee_anciennete() >= maxi:
            if p.annee_anciennete() > maxi:
                maxi = p.annee_anciennete()
                liste = []
            liste.append(p.num_badge)
    return liste
```

Exercice 3 : Partie B :

9. La requête donner le nom et le prénom de tout le personnel travaillant dans le centre n° 2.

10.
```
UPDATE Personnel SET num_centre = 3 WHERE num_badge = 135
```

11. Utiliser deux tables permet d'éviter la redondance d'information et facilite les mises à jour.

12. Ces deux tables sont mises en relation par la clé étrangère `num_centre` de la table `Personnel` qui fait référence à la clé primaire `num` de la table `Centre`.

13.
```
SELECT Personnel.nom FROM Personnel
JOIN Centre ON Personnel.num_centre = Centre.num
WHERE Centre.nom = 'Lille' AND Personnel.annee_debut BETWEEN 2015 AND 2020 ;
```

14. Cet requête renvoie un erreur car elle viole la contrainte d'intégrité de référence de la base de donnée. En effet, il est impossible de supprimer un enregistrement auquel d'autres font référence.

(Erreur de l'énoncé : La requête devrait être :
```
DELETE FROM Centre WHERE nom = 'Normandie';)
```