Algorithmes et Pensée Computationnelle

Programmation orientée objet : Héritage et Polymorphisme - Exercices avancés

Le but de cette séance est d'approfondir les notions de programmation orientée objet vues précédemment. Les exercices sont construits autour des concepts d'héritage, de surcharge d'opérateurs/méthodes et de polymorphisme. Au terme de cette séance, vous devez être en mesure de factoriser votre code afin de le rendre mieux structuré et plus lisible. Cette série d'exercices vous allez utiliser Python comme language de programmation.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Code**.

1 Héritage en Python

Question 1: (*15 minutes*) **Classe Point (Suite)**
Lors de la séance de TP de la semaine 12, vous avez défini une classe **Point** représentant un point de 2 dimensions, avec des coordonnées x et y, ainsi que des opérations basiques sur des points 2D. Pour cet exercice, vous allez implémenter une classe des points à 3 dimensions qui héritera de la classe **Point** créée précédemment.
Voici la classe **Point** qui a été légèrement modifiée (Vous trouverez le fichier sur Moodle, dans le dossier **Ressources**) :

```
import math

class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def get_x(self):
        return self._x

    def get_y(self):
        return self._y

    def set_x(self, x):
        self._x = x

    def set_y(self, y):
        self._y = y

    def distance_euclidean(self, p2):
        return math.sqrt((self._x - p2.get_x()) ** 2 + (self._y - p2.get_y()) ** 2)

    def milieu(self, p2):
        x_M = (self._x + p2.get_x()) / 2
        y_M = (self._y + p2.get_y()) / 2
        M = Point(x_M, y_M)
        return M

    def __str__(self):
        return "Les coordonnées du point sont: x=" + str(self.get_x()) + ", y=" + str(self.get_y())
```

Écrivez une classe qui hérite de **Point**. Nommez-la **Point3D**. Après avoir rajouté la 3ème dimension comme attribut, implémentez les opérations ci-dessous :

- Rajoutez une méthode qui renvoie une représentation vectorielle du point. Vous pouvez utiliser une list en Python.
- Recalculez la distance euclidienne et le milieu pour le point 3D.
- Pour aller plus loin Si vous voulez vous familiariser encore plus avec les méthodes de classe en Python, implémentez deux autres algorithmes de calculs de distance : Manhattan et Minkowski.

```
class Point3D(Point):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.__z = z

    def get_z(self):
        ...

```

```
    def set_z(self, z):
        ...

    def vector_representation(self): # représentée sous forme de liste
        ...

    def distance_euclidean(self, p2): # i.e norme
        ...

    def distance_manhattan(self, p2):
        ...

    def distance_minkowski(self, p2, order=3):
        ...

    def milieu(self, p2):
        ...
```

Conseil

Que fait `super().__init__()` ?
Dans un espace à 3 dimensions, la formule pour calculer la distance entre un $p_1 = (x_1, y_1, z_1)$ et $p_2 = (x_2, y_2, z_2)$ est $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (y_1 - y_2)^2}$.

Solution

```
class Point3D(Point):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.__z = z

    def get_z(self):
        return self.__z

    def set_z(self, z):
        self.__z = z

    def vector_representation(self): # représentée sous forme de liste
        return [self.__x, self.__y, self.__z]

    def distance_euclidean(self, p2): # i.e norme
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()
        return math.sqrt((self.__x - other_x)**2 + (self.__y - other_y)**2 + (self.__z - other_z)**2)

    def distance_manhattan(self, p2):
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()
        return sum((abs(self.__x - other_x), abs()), abs(self.__y - other_y), abs(self.__z - other_z))

    def distance_minkowski(self, p2, order=3):
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()
        return sum((abs(self.__x - other_x)**order), abs(self.__y - other_y)**order,\
                abs(self.__z - other_z)**order)**(1/order)

    def milieu(self, p2):
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()

        x_M = (self.__x + other_x)/2
        y_M = (self.__y + other_y)/2
        z_M = (self.__z + other_z)/2
        return Point3D(x_M, y_M, z_M) # renvoie un point!


point1 = Point3D(1, 2, 3)
point2 = Point3D(3, 4, 5)

# exemple
point1.vector_representation()
```

Question 2: (*15 minutes*) **Un exemple appliqué**

Dans les établissements universitaires, on rencontre souvent des problèmes lors du calcul de salaires du personnel. Sans penser aux recherches effectuées par certains professeurs, on va essayer de calculer les salaires de ceux qui sont reconnus comme 'Professeur' (ordinaire, titulaire, associé ou assistant) à l'université et ceux qui y donnent des cours à temps partiel (on va les considérer comme 'Collaborateurs' dans cet exercice).

La classe mère dans ce cas est nommée **Enseignant**, qui possède une propriété - le salaire annuel moyen. On voudrait que la méthode qui calcule cette quantité renvoie 60 000 (dollars américains) si l'enseignant a moins de 10 ans d'expérience, et 100 000 sinon. Si l'enseignant travaille à temps partiel, la méthode devrait renvoyer une chaîne qui dit 'Le salaire annuel ne s'applique pas aux collaborateurs'.

Ensuite, on veut calculer la paye mensuelle pour chaque type d'employé. Pour les **Professeurs**, la paye devrait être calculée sur la base de deux sources de revenu : un salaire mensuel et une commission pour chaque comité où ils participent.

D'autre part, pour les **Collaborateurs**, la paye est calculée sur une base horaire i.e *taux horaire* $\times$ *nombres*

d'heures de travail (par mois).
Complétez le code ci-dessous :

```
class Enseignant:
    def __init__(self, name, years_experience, full_time):
        ...
    def salaire_annuel_moyen(self):
        ...

class Professeur(Enseignant):
    def __init__(self, name, years_experience, monthly_salary, commission, num_committees):
        ...
    def paye_mensuelle(self):
        ...

class Collaborateur(Lecturer):
    def __init__(self, name, years_experience, hours_per_month, rate):
        ...
    def paye_mensuelle(self):
        ...

prof1 = Professeur("Alexandra", 8, 3000, 200, 4)
prof2 = Collaborateur("David", 10, 40, 30)
# exemples
print(prof1.salaire_annuel_moyen())
print(prof2.paye_mensuelle())
```

Conseil

Pensez à redéfinir les attributs de la classe mère en utilisant `super().__init__()`.

Solution

```
class Enseignant:
    def __init__(self, name, years_experience, full_time):
        self.name = name
        self.years_experience = years_experience
        self.full_time = full_time

    def salaire_annuel_moyen(self):
        if self.full_time:
            if self.years_experience < 10:
                return 60000

            else:
                return 100000

        else:
            return "Le salaire annuel ne s'applique pas aux collaborateurs"


class Professeur(Enseignant):
    def __init__(self, name, years_experience, monthly_salary, commission, num_committees):
        super().__init__(name, years_experience, True)
        self.monthly_salary = monthly_salary
        self.commission = commission
        self.num_committees = num_committees

    def paye_mensuelle(self):
        return self.monthly_salary + self.commission*self.num_committees

class Collaborateur(Enseignant):
    def __init__(self, name, years_experience, hours_per_month, rate):
        super().__init__(name, years_experience, False)
        self.hours_per_month = hours_per_month
        self.rate = rate

    def paye_mensuelle(self):
        return self.hours_per_month*self.rate

prof1 = Professeur("Alexandra", 8, 3000, 200, 4)
prof2 = Collaborateur("David", 10, 40, 30)

# exemples
print(prof1.salaire_annuel_moyen())
print(prof2.paye_mensuelle())
```