Marseille, 05/12/2018 - Achref El Mouelhi

TP 2 : POO avec Python

Exercice 1

Considérons une classe appelée `Point` ayant les attributs suivants :

- `__abs` : un attribut privé de type `float`
- `__ord` : un attribut privé de type `float`

1. Définissez la classe `Point` et un constructeur à deux paramètres
2. Définissez les getters et setters pour les deux attributs en utilisant le décorateur `@property`
3. Définissez la méthode `__str__()` qui retourne la représentation mathématique d'un point : `(abs, ord)`.
4. Écrivez la méthode `calculer_distance(self, p: 'Point') -> float:` qui permet de calculer la distance entre le point de l'objet courant (`self`) et l'objet `p` passé en paramètre. Nous rappelons que la distance entre deux points A(x_1,y_1) et B(x_2,y_2), en mathématiques, est égale à :
   $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2)}$
   Vous pouvez utiliser la fonction `math.sqrt(a)` pour calculer la racine carrée de `a` et `math.pow(x, y)` pour calculer x^y
5. Écrivez la `calculer_milieu(self, p: 'Point') -> 'Point':` qui permet de calculer et de retourner un objet correspondant au milieu du segment défini par le point de l'objet courant (`this`) et l'objet `Point p` passé en paramètre. Nous rappelons que les coordonnées d'un point M(x_M,y_M) milieu de A(x_1,y_1) et B(x_2,y_2), en mathématiques, sont :
   - $x_M = \frac{x_1 + x_2}{2}$
   - $y_M = \frac{y_1 + y_2}{2}$

   La méthode doit retourner un objet `Point` et pas les coordonnées.

Considérons maintenant une deuxième classe appelée `TroisPoints` ayant les attributs suivants :

- `__premier` : un attribut privé de type `Point`
- `__deuxième` : un attribut privé de type `Point`
- `__troisième` : un attribut privé de type `Point`

6. Définissez les getters/setters (avec le décorateur `@property`) et un constructeur acceptant trois paramètres.
7. Écrivez une méthode `sont_alignes(self) -> bool:` qui retourne `True` si les trois points `_premier`, `_deuxième` et `_troisième` sont alignés, `False` sinon. Nous rappelons que trois points A, B et C sont alignés si AB = AC + BC, AC = AB + BC ou BC = AC + AB (AB désigne la distance séparant le point A du point B, pareillement pour AC et BC).
8. Écrivez une méthode `est_isocele(self) -> bool:` qui retourne `True` si les trois points `premier`, `deuxième` et `troisième` forment un triangle isocèle, `False` sinon. Nous rappelons qu'un triangle ABC est isocèle si AB = AC ou AB = BC ou BC = AC.

9. Implémentez une version statique (méthode décorée par `@staticmethod`) des deux méthodes calculant la distance et le milieu.
10. Dans un fichier `main.py`, testez toutes les classes et méthodes que vous avez implémentées.

Exercice 2

Considérons les deux classes `Personne` et `Adresse`. Les attributs de la classe `Adresse` sont :

- `__rue` : un attribut privé de type chaîne de caractères.
- `__ville` : un attribut privé de type chaîne de caractères.
- `__code_postal` : un attribut privé de type chaîne de caractères.

Les attributs de la classe `Personne` sont :

- `__nom` : un attribut privé de type chaîne de caractères.
- `__sexe` : un attribut privé de type chaîne de caractères (cet attribut aura comme valeur soit 'M' soit 'F').
- `__adresses` : un attribut privé de type tableau d'objet de la classe `Adresse`.

1. Créez les deux classes `Adresse` et `Personne` dans deux fichiers différents. N'oubliez pas de définir les getters/setters et les constructeurs.
2. Créez une troisième classe `ListePersonnes` ayant un seul attribut `personnes` : un tableau d'objets `Personne`. Définissez les getters/setters et le constructeur de cette classe.
3. Écrivez la méthode `find_by_nom(s: str)` qui permet de chercher dans le tableau `personnes` si l'attribut `nom` d'un est égal à la valeur du paramètre `s`. Si c'est le cas, elle retourne le premier objet correspondant, sinon null.
4. Écrivez la méthode `exists_code_postal(cp: str)` qui permet de vérifier dans le tableau `personnes` si un objet possède au moins une adresse dont le code postal égal au paramètre `cp`. Si c'est le cas, elle retourne `True`, sinon `False`.
5. Écrivez la méthode `count_personne_ville(ville: str)` qui permet de calculer le nombre d'objets dans le tableau `personnes` ayant une adresse dans la ville passée en paramètre.
6. Écrivez la méthode `edit_personne_nom(oldNom: str, newNom: str)` qui remplace les noms de personnes ayant un nom égal à la valeur `oldNom` par `newNom`
7. Écrivez la méthode `edit_personne_ville(nom: str, newVille: str)` qui remplace les villes de personnes ayant un nom égal à la valeur du paramètre `nom` par `newVille`
8. Dans `ListePersonnes`, définissez un indexeur sur `personnes`.
9. Dans `Personne`, définissez un indexeur sur `adresses`.
10. Dans `main`, testez toutes les méthodes réalisées dans les questions précédentes.