Programmation objet

I. Définir et utiliser une classe

1. Dans un module ou un programme **vehicules.py**, créer la classe **Voiture()** vue en cours.

2. Instancier deux objets **voiture_1** et **voiture_2** à partir de cette classe, avec des paramètres différents. Que retournent **print(voiture_1)** et **print(voiture_2)** ?

3. Accéder directement aux valeurs des attributs des deux objets dans le programme principal.

4. Modifier directement les attributs des deux objets dans le programme principal.

5. Utiliser la méthode **caractéristiques()** pour obtenir les valeurs des attributs. Comment s'appelle ce type de méthode ?

6. Utiliser la méthode **modifie_couleur()** pour modifier la couleur des deux objets. Comment s'appelle ce type de méthode ?

7. Ajouter les méthodes manquantes permettant de modifier convenablement les valeurs des attributs.

8. Ajouter les attributs **nombre_portes**, **masse**, **boite_vitesse**, **masse**, **masse_vide**, **masse_PTAC**.

9. Modifier la méthode **caractéristiques()** pour qu'elle retourne les valeurs de tous les attributs.

10. Cette méthode retourne un dictionnaire. Décrire cette structure de donnée.

11. Les dictionnaires **Python** sont des objets. Que font leurs méthodes **.keys()**, **.values()** et **.items()** ?

12. Les utiliser dans des boucles pour afficher successivement toutes les clés de **voiture_2.caracteristiques()**, puis toutes ses valeurs et enfin toutes ses paires **(clé, valeur)**.

13. Créer une méthode à la classe **Voiture()** permettant d'afficher les caractéristiques sous la forme :

Les caractéristiques du véhicule sont :
moteur : essence
couleur : violet
… etc …

14. Créer une méthode **charge()** qui augmente la masse du véhicule. La masse sera modifiée et la méthode retournera **True** si la nouvelle masse est autorisée, **False** sinon.

15. Créer une méthode **decharge()** qui diminue la masse du véhicule. La masse sera modifiée et la méthode retournera **True** si la nouvelle masse est autorisée, **False** sinon.

16. Passer quelques méthodes en privé et vérifier que c'est le cas avec la fonction **help()**. Montrer avec la fonction **dir()** que rien n'est vraiment privé en Python. Les rendre à nouveau publiques.

II. Vecteurs

1. Dans un module **maths_a_moi.py**, créer une classe **Vecteur()**. Un vecteur sera modélisé par ses trois composantes **x**, **y** et **z**.

2. Construire et tester les méthodes suivantes :

- **set_composantes_xyz() :** fixe les composantes du vecteur.
- **set_composantes_AB()** : fixe les composantes du vecteur à partir des coordonnées des deux points A et B qui le définissent.
- **get_composantes()** : retourne les coordonnées du vecteur sous la forme d'un tuple.
- **addition()** : effectue l'addition de deux vecteurs. Le résultat remplace les coordonnées du vecteur appelant la méthode.
- **soustraction()** : effectue la soustraction de deux vecteurs.
- **norme()** : retourne la norme du vecteur.
- **produit_scalaire()** : retourne le produit scalaire de deux vecteurs.
- **angle()** : donne l'angle en radian entre deux vecteurs.
- **is_colineaire()** : retourne **True** le vecteur est colinéaire à un autre.
- **is_orthogonal()** : retourne **True** le vecteur est orthogonal à un autre

3. Diviser la classe **Vecteur()** en deux classes différentes, une permettant de créer des objets **Vecteur**, l'autre rassemblant les opérations sur les objets **Vecteur**.

III. Polynômes

1. Toujours dans le module **maths_a_moi.py**, créer une classe **Polynome()**. Un polynôme sera modélisé par une liste dont les éléments correspondent aux coefficients de ses différents degrés.

2. Construire une première méthode **degre()** donnant le degré du polynôme.

3. Dans un premier temps, ajouter deux méthodes **addition()** et **soustraction()** permettant d'additionner et de soustraire deux polynômes.

4. Dans un second temps, ajouter deux méthodes **multiplication()** et **division()** permettant de multiplier et de diviser deux polynômes.

5. Ajouter une méthode privée **discriminant_degre2()** calculant le discriminant du polynôme s'il est de degré **2**. La méthode stoppera le programme avec le message **Le degré doit être de 2** si ce n'est pas le cas. On utilisera pour cela une assertion avec le mot clé **assert**.

6. Utiliser cette méthode dans une méthode **racines_degre2()** calculant les racines réelles du polynôme s'il est de degré 2. La méthode stoppera le programme avec le message **Le degré doit être de 2** si ce n'est pas le cas. La méthode retournera **None** s'il ce n'a pas de racines réelles.

7. Diviser la classe **Polynome()** en deux classes différentes, une permettant de créer des objets **Polynome**, l'autre rassemblant les opérations sur les objets **Polynome**.