POO : Atelier n°2 Surcharges et Héritages

Exercice 1 (clé) :

1) Créer une classe rectangle composé des attributs longueur et largeur, de la méthode périmètre qui retourne le périmètre d'un rectangle, et de la méthode surface qui retourne la surface d'un rectangle.
2) Utiliser la fonction __add__() pour surcharger l'opérateur + et pouvoir ajouter de rectangle (ajouter deux rectangles revient à ajouter la largeur et la longueur des rectangles).
3) Utiliser la fonction __str__() pour surcharger la fonction print pour afficher le résultat suivant :

```
rec1 = Rectangle(5, 5)
print(rec1)
```

```
Longueur : 5 et largeur : 5
```

4) Utiliser la fonction __eq__() pour comparer 2 rectangles puis utiliser une condition et afficher le message suivant si 2 rectangles sont égaux : "Les rectangles sont égaux".

Exercice 2 (clé) :

1) Créer une classe Personne qui comprend les attributs suivants :
   - nom
   - prenom
   - age
2) Créer une méthode affiche() qui afficher les attributs d'un objet de cette classe
3) Créer 1 objet à partir de cette classe et tester la méthode affiche().
4) Créer une classe Employe qui hérite de la classe Personne et qui comprend les attributs suivants :
   - salaire
   - type_contrat
5) Redéfinissez la méthode affiche() pour pouvoir afficher l'ensemble des attributs d'un Employe (polymorphisme).
6) Créer 1 objet à partir de cette classe et tester la méthode affiche().

Exercice 3 (clé) :

1) Créer une classe abstraite Animal qui comprend les attributs suivants :
    - espece
    - age
2) Dans cette classe, créer 2 méthodes abstraites **crier()** et **hiberner()**.
3) Créer une classe Reptile qui hérite de la classe Animal qui comporte un attribut venin.
   Puis redéfinissez les méthodes crier() et hiberner() (les reptiles n'hibernent pas).
4) Créer une classe Marmotte qui hérite de la classe Animal qui comporte un attribut pelage.
   Puis redéfinissez les méthodes crier() et hiberner (les marmottes hibernent).
5) Tester le bon fonctionnement des classes Reptile et Marmotte en créant des objets puis en faisant appel aux méthodes crier() et hiberner().

Exercice 4 (clé) :

1) Créer une classe Cercle qui comprend un attribut rayon
2) Créer un constructeur permet de gérer les 2 scénarios suivants :
    - Ne possède pas d'arguments et initialise l'attribut rayon à 1.
    - Possède un argument rayon.
3) Créer une méthode aire() qui retourne l'aire du cercle (vous pouvez vous aider de la librairie math).
4) Créer une classe Cylindre qui hérite de la classe Cercle et qui possède un attribut supplémentaire : hauteur (type double).
5) Créer une méthode volume() qui retourne le volume d'un objet de la classe.
6) Créer des objets et tester l'ensemble de votre programme.

Exercice 5 (secondaire) :

Définir une classe Point2D avec un constructeur fournissant les coordonnées par défaut un point d'un plan à 2 dimensions (par exemple : x = 0 et y = 0).

Dans le programme principal, instanciez un Point2D sans paramètre, un Point2D avec ses deux paramètres, et affichez-les.

Dans un premier temps, implémentez dans la classe les fonctions permettant d'ajouter, de soustraire et de comparer des points.

Dans un second temps, surcharger la fonction print() pour afficher un point.

Exercice 6 (secondaire) :

1) Créer une classe abstraite Forme contenant une seule méthode abstraite : getSurface().
2) Créer une classe Carré qui hérite de la classe de base Forme. Cette classe est composé :
    - D'un attribut cote
    - D'une méthode qui retourne le périmètre du carré
    - De la méthode getAire() qui retourne l'aire d'un Carre
3) Créer une classe Cercle qui hérite de la classe de base Forme. Cette classe est composé :
    - D'un attribut rayon
    - D'une méthode qui affiche le rayon du cercle
    - De la méthode getAire() qui retourne la surface d'un Cercle

Exercice 7 (secondaire) :

On veut développer un logiciel pour générer des sujets d'examen. Un programmeur a écrit la classe suivante avant de démissionner (il a gagné au loto). Vous devez continuer le développement.

```
class Question:

    def __init__(self, enonce, difficulte=50):
      self.enonce = enonce
      self.difficulte = difficulte

    def setDifficulte(self, difficulte):
      self.difficulte = difficulte
```

On veut représenter les questions de QCM (Questionnaire à Choix Multiples). Chaque question à choix multiple a, en plus d'un énoncé et d'une difficulté, un ensemble de réponses possibles, chaque réponse ayant un énoncé et une valeur de vérité (une réponse peut être vraie ou fausse).

a) Proposez du code pour représenter les questions à choix multiples.

b) Un sujet d'examen est une liste de questions (de QCM ou autre) avec un barême. Chaque question du sujet a donc un nombre de points. Sans modifier les classes que vous avez écrites avant, écrivez du code objet pour représenter les sujets, avec en particulier une méthode qui permet d'obtenir retour au sommaire la difficulté moyenne du sujet. La difficulté moyenne est la somme des difficultés des questions du sujet, divisée par le nombre de questions.

c) Vérifier que l’ensemble du programme fonctionne en créant des objets.