TP4

PYTHON : LA PROGRAMMATION OBJET

Exercice 1 :

Ecrire une classe Livre avec les attributs suivants :

- titre : Le titre du livre,
- auteur : L'auteur du livre,
- prix : Le prix du livre,
- annee : L'année d'édition du livre.

2. La classe Livre doit pouvoir être construites avec les possibilités suivantes :

- Livre(),
- Livre(titre),
- Livre(titre, auteur),
- Livre(titre, auteur, prix),
- Livre(titre, auteur, prix, annee)

3. Instancier deux livres de votre choix

4. Créer une fonction qui renvoie vrai si le titre de l'instance passée en paramètre correspond au titre demandé.

Exercice 2 : Compte Bancaire

- Définissez une classe CompteBancaire, qui permette d'instancier des objets tels que compte1, compte2, etc.
- Le constructeur de cette classe initialisera deux attributs nom et solde, avec les valeurs par défaut DSI et 2500.
- Trois autres méthodes seront définies :
- depot(somme) permettra d'ajouter une certaine somme au solde
- retrait(somme) permettra de retirer une certaine somme du solde
- Créer une méthode Agios() permettant d'appliquer les agios à un pourcentage de 5 % du solde
- Créer une méthode afficher() permettant d'afficher les détails sur le compte
- Donner le code complet de la classe CompteBancaire.

affiche() permettra d'afficher le nom du titulaire et le solde de son compte.

Exemples d'utilisation de cette classe :

compte1 = CompteBancaire (' Sami ' , 800)

compte1 . depot (3 5 0)

compte1 . r e t r a i t (2 0 0)

compte1 . a f f i c h e ( )

Le solde du compte b a n c ai r e de Sami e s t de 950 dinars

compte2 = CompteBancaire ( )

compte2 . depot (2 5)

compte2 . a f f i c h e ( )

Exercice 3 : nombre complexe

Un complexe z est composé d'une partie réelle x et d'une partie imaginaire y, tel que :

$z = x + i * y$.

Nous proposons ici d'implémenter une classe Complex.

1. **Création**

- Définir une classe Complex
- Commenter la classe Complex
- Ajouter un constructeur :
- Ajouter deux attributs x et y.
- Ajouter au constructeur la possibilité de spécifier les valeurs de x et y

2. **Instanciation :**

Créer un objet c1, instance de la classe Complex

3. **Identité :**

Effectuer les manipulations avec le nouvel objet c1 suivantes :

print c1. doc

print c1

Que remarquez vous ?

4. **Comportement :**

Affecter la valeur 3 à l'attribut x et 4 à l'attribut y de l'objet c1

Les fonctions peuvent utiliser des objets comme paramètres. D´définir une fonction affichecomplex qui affiche les valeurs du complex passé en paramètre.

Testez la avec l'objet c1.

Créer une m´méthode (fonction dans la classe) show qui affiche les attributs de la classe Complex. Testez-la avec l'objet c1.

5. Manipulation de plusieurs objets.

Créer une fonction compareComplex qui renvoie vrai si chaque attribut de deux objets sont identiques.

Créer un objet c2 de type Complex, tester compareComplex

Affecter la valeur 3 à l'attribut x et 4 à l'attribut y de l'objet c2.

Ajouter une méthode clone qui recopie un objet de type Complex.

Effectuer les manipulations suivantes :

```
print (c1 == c2)

c1=c2

print (c1 == c2)

c3=c1.clone()

print (c1 == c3)
```

Que remarquez vous ?

Exercice 4 : Collaboration entre objets

Pour définir un rectangle, nous spécifions sa largeur, sa hauteur et précisons la position du coin supérieur gauche.

Une position est définie par un point (coordonnées x et y).

- Définir la classe Point contenant les attributs x et y (coordonnées) .
- Définir la classe Rectangle.
- Instancier un objet rectangle1 de largeur 50, de hauteur 35, et dont le coin supérieur gauche se situe aux coordonnées (12,27)
- Améliorer le constructeur de la classe Point et celui de la classe Rectangle avec des possibilités que vous jugerez utiles (valeurs par d´efaut ...)
- Préciser le destructeur de la classe Point et de la classe Rectangle
- Tester le destructeur de la classe Rectangle
- Modifier la taille d'un rectangle (sans modifier sa position), en réassignant ses attributs hauteur (hauteur actuelle +20) et largeur (largeur actuelle -5).

Les fonctions peuvent utiliser des objets comme paramètres. Elles peuvent également transmettre une instance comme valeur de retour :

- Définir la fonction trouveCentre() qui est appelée avec un argument de type Rectangle et qui renvoie un objet Point, lequel contiendra les coordonnées du centre du rectangle.
- Tester cette fonction en utilisant l'objet rectangle1

Exercice 5 : héritage

1. Proposer une structure de classe individu : nom, prénom, date de naissance. Prévoir un constructeur pour la classe et une méthode qui affiche le nom, prénom et date de naissance
2. Proposer une méthode permettant de calculer l'age d'un individu,
3. Proposer une structure de classe étudiant qui hérite de la classe individu. Modifier le constructeur pour tenir compte des informations : bac, notes.
4. Proposer une méthode permettant d'ajouter une note.
5. Proposer une méthode de calcul de moyenne générale des notes.

Exercice 6 : surcharge d'opérateur

On considère la classe point ci-dessous :

```
import math

class Point:
    def __init__(self, x=0, y=0):
        self.__x = x
        self.__y = y

    # Opérateur +
    def __add__(self, p):
        return Point(self.__x + p.__x, self.__y + p.__y)

    # Opérateur -
    def __sub__(self, p):
        return Point(self.__x - p.__x, self.__y - p.__y)

    # Opérateur <
    def __lt__(self, p):
        m_self = math.sqrt((self.__x ** 2) + (self.__y ** 2))
        m_p = math.sqrt((p.__x ** 2) + (p.__y ** 2))
        return m_self < m_p

    # Opérateur <=
    def __le__(self, p):
        m_self = math.sqrt((self.__x ** 2) + (self.__y ** 2))
        m_p = math.sqrt((p.__x ** 2) + (p.__y ** 2))
        return m_self <= m_p

    # Opérateur >
    def __gt__(self, p):
        m_self = math.sqrt((self.__x ** 2) + (self.__y ** 2))
        m_p = math.sqrt((p.__x ** 2) + (p.__y ** 2))
        return m_self > m_p
```

```
    # Opérateur >=
    def __ge__(self, p):
        m_self = math.sqrt((self.__x ** 2) + (self.__y ** 2))
        m_p = math.sqrt((p.__x ** 2) + (p.__y ** 2))
        return m_self >= m_p

    # Opérateur ==
    def __eq__(self, p):
        m_self = math.sqrt((self.__x ** 2) + (self.__y ** 2))
        m_p = math.sqrt((p.__x ** 2) + (p.__y ** 2))
        return m_self == m
```

On vous demande de :

- Ajouter de la documentation décrivant en détail ce que permet la classe et ses différentes méthodes
- créer des instances de la classe point
- tester les différentes méthodes de la classe

Exercice 7 :

Nous allons définir une classe appelée **Temps**, qui enregistre le moment de la journée.

La définition de la classe ressemble à ceci :.

```
class Temps:
  """Représente le moment de la journée.

  attributs : heure, minute, seconde
  """
```

1. Ecrire le **constructeur** de la classe, une méthode appelée **afficher_temps** qui prend un objet Temps et l'affiche dans la forme heure:minute:seconde.

Indice : la séquence de formatage **'%.2d'** affiche un entier en utilisant au moins deux chiffres, en ajoutant un zéro en début si nécessaire.

Ecrire une fonction booléenne nommée **est_apres** qui prend deux objets Temps, t1 et t2, et retourne True si t1 suit chronologiquement t2 et False sinon.

Défi : n'utilisez pas d'instruction if.

2. Parfois, il est utile qu'une fonction puisse modifier les objets qu'elle reçoit comme paramètres. Dans ce cas, les changements sont visibles par la procédure appelante. Ce genre de fonctions s'appellent **modificateurs**.

Créer une méthode **incremente**, qui ajoute un nombre donné de secondes à un objet Temps

Consignes : Pour les méthodes **ajouter_temps** et **increment**, je vous suggère l'approche suivante : ***convertir des objets Temps en entiers et profiter du fait que l'ordinateur sait comment faire l'arithmétique avec des entiers.***

Voici une fonction qui convertit des Temps en entiers :

```
def temps_vers_int(temps):
    minutes = temps.heure * 60 + temps.minute
    secondes = minutes * 60 + temps.seconde
    return secondes
```

Et voici une fonction qui convertit un nombre entier vers un Temps (rappelez-vous que **divmod** effectue une division entière du premier argument par le second et renvoie le quotient et le reste sous la forme d'un tuple).

```
def int_vers_temps(secondes):
    temps = Temps()
    minutes, temps.seconde = divmod(secondes, 60)
    temps.heure, temps.minute = divmod(minutes, 60)
    return temps
```

3. Définir une méthode nommée __add__ pour la classe **Temps**, pour utiliser l'opérateur + sur les objets de type Temps.