Chapitre 2

TP. Programmation orientée Objet en Python

Exercice 1.

1. Créer une classe Python nommé « **Point** » définie par deux attributs d'instances **x = 3.0** et **y = 4.0** représentants les coordonnées d'un point.
2. Créer une instance **p1** de la classe **Point.**
3. Définir une méthode **afficher(self)** permettant d'afficher un point.
4. Définir une classe Python nommé « **Rectangle** » définie par un point qui représente la position du coin supérieur gauche du rectangle et sa taille (largeur et hauteur).
5. Créer une instance **boite** de la classe **Rectangle** où la valeur de son coin supérieur gauche est un Point (12.0 ; 27.0), sa largeur = 50.0 et sa hauteur = 35.0.
6. Créer une fonction Python (en dehors de la classe Rectangle) nommée **trouverCentre** qui peut être appelée avec un argument de type **Rectangle** et renvoie un objet de type **Point**, lequel contiendra les coordonnées du centre du rectangle.
7. Appeler cette fonction en utilisant comme argument l'objet **boite** défini plus haut et afficher le **centre** trouvé.
8. Modifier la taille (**hauteur** et **largeur**) de l'objet **boite** sans modifier sa position (**coin**) ;

Exercice 2.

1. Définir une classe Python nommé « **Time** » définie par les variables d'instances pour mémoriser les **heures**, **minutes** et **secondes.** Exemple : (12 : 32 : 45).
2. Créer un objet **instant** de type **Time**.
3. Créer une fonction **affiche_heure(t)** qui serve à visualiser les attributs d'un objet **t** de la classe **Time**.
4. Redéfinir la méthode **affiche_heure (self)**, à l'intérieur de la classe **Time**. Le premier paramètre **« self »** est obligatoire, car il représente l'instance à laquelle la méthode sera associée.
5. Créer un objet **maintenant** de type **Time** et tester la nouvelle méthode sur cet objet.

Exercice 3.

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 **'Salim'** et **1000**.

1. Trois autres méthodes seront définies :
   a. **depot(self,somme)** permettra d'ajouter une certaine somme au solde ;
   b. **retrait(self,somme)** permettra de retirer une certaine somme du solde ;
   c. **__repr__(self)** 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(350)
>>> compte1.retrait(200)
>>> compte1
Le solde du compte bancaire de Sami est de 950 Dinars.
>>> compte2 = CompteBancaire()
>>> compte2.depot(25)
>>> compte2
Le solde du compte bancaire de Salim est de 1025 Dinars
```

Exercice 4.

Définissez une classe **Voiture( )**, qui permette d'instancier des objets reproduisant le comportement de voitures automobiles. Le constructeur de cette classe initialisera les attributs d'instance suivants, avec les valeurs par défaut indiqués : **marque = 'Ford', couleur = 'rouge', pilote = 'personne', vitesse = 0.**

Lorsque l'on instanciera un nouvel objet Voiture ( ), on pourra choisir sa marque et sa couleur, mais pas sa vitesse, ni son conducteur.

Définir les méthodes suivantes :

1. **choix_conducteur (self)** permettra de désigner (ou changer) le nom du conducteur.
2. **accelerer (self,taux, duree)** permettra de faire varier la vitesse de la voiture. La variation de la vitesse de la voiture sera égale au produit : taux × duree. Par exemple, si la voiture accélère au taux de 1,3 m/s pendant 20 secondes, son gain de vitesse doit être égal à 26 m/s. Des taux négatifs seront acceptés (ce qui permettra de décélérer). La variation de vitesse ne sera pas autorisée si le conducteur est « personne ».
3. **affiche_tout (self)** permettra de faire apparaître les propriétés présentes de la voiture, c'est-à-dire sa marque, sa couleur, le nom de son conducteur et sa vitesse.
4. Définir une méthode spéciale **__str__(self)** qui joue le même rôle que celui de **affiche_tout ( ).**

Exemples d'utilisation de cette classe :

```
>>> a1 = Voiture('Peugeot', 'bleu')
>>> a2 = Voiture(couleur='verte')
>>> a3 = Voiture('Mercedes')
>>> a1 = Voiture('Peugeot', 'bleu')
>>> a1.choix_conducteur("Ramzi")
>>> a2.choix_conducteur("Safa")
>>> a2.accelerer(1.8, 12)
>>> a3.accelerer(1.9, 11)
Cette voiture n'a pas de conducteur
>>> a2.affiche_tout( )
Ford verte pilotée par Safa, vitesse = 21.6 m/s
>>> a3.affiche_tout( )
Mercedes rouge pilotée par personne, vitesse = 0 m/s
>>> print(a3)
Mercedes rouge pilotée par personne, vitesse = 0 m/s
```

Exercice 5.

1. Les classes sont souvent utilisées pour modéliser des objets dans le monde réel. Nous pouvons représenter les données sur une personne dans un programme par une classe **Personne**, contenant le nom de la personne, son prénom, son numéro de téléphone, et son email. Une méthode **__str__** peut imprimer les données de la personne.

   Exemples :

   ```
   >>> amir = Personne('Ben Salem', 'Amir', '55593581', "amirbs@ipein.rnu.com")
   >>> print(amir)
   Ben Salem, Amir -- Telephone: 55593581 -- Email: "amirbs@ipein.rnu.com"
   >>> mariem = Personne('Abassi', 'Mariem', '55519403',"mariem12@gmail.com")
   >>> print(mariem)
   Abassi, Mariem -- Telephone : 55519403 -- Email :"mariem12@gmail.com"
   ```

   Travail à faire : Implémentez la classe Personne.

2. Un travailleur est une personne ayant un emploi. Dans un programme, un travailleur est naturellement représenté comme une classe **Travailleur** dérivée de la classe **Personne**, parce qu'un travailleur est une personne, c'est-à-dire, nous avons une relation **est-un**. La classe Travailleur étend la classe Personne avec des données supplémentaires, par exemple le nom de l'entreprise, l'adresse de l'entreprise et le numéro de téléphone du travail. La fonctionnalité d'impression (la méthode spéciale __str__) doit être modifiée en conséquence.
   Travail à faire : Mettre en œuvre cette classe Travailleur.

3. Un scientifique est un type spécial de travailleur. La classe **Scientifique** peut donc être dérivée de la classe **Travailleur**. Ajouter des données sur la discipline scientifique (physique, chimie, mathématiques, informatique, ...). On peut aussi ajouter le type de scientifique : théorique, expérimental ou informatique. La valeur d'un tel attribut de type ne doit pas être limitée à une seule catégorie, car un scientifique peut être classé comme, par exemple, à la fois expérimental et informatique (c'est-à-dire, vous pouvez représenter la valeur sous la forme d'une liste ou d'un tuple).
   Travail à faire : Mettre en œuvre la classe Scientifique.

4. Enfin, faites un programme principal de démonstration où vous créez et imprimez des instances de classes **Personne**, **Travailleur** et **Scientifique**. Imprimez le contenu des attributs de chaque instance.

Exercice 6.

Dans cet exercice vous allez construire un carnet d'adresses en utilisant la classe **Personne** définie dans le l'exercice précédent afin d'enregistrer vos contacts (amis, famille, ...). Chaque entrée du carnet d'adresses sera une instance de la classe **Personne**.

Le carnet d'adresses doit vous permettre de chercher et retourner les informations de vos contacts.

Travail à faire :

Créer une classe **CarnetAdresses** contenant les méthodes suivantes :

1. La méthode constructeur __init__
2. La méthode d'impression __str__
3. Une méthode nommée **ajouter_contact** qui vous permet d'ajouter une personne à votre carnet d'adresses.
4. Une méthode **chercher_contact** qui recherche un contact par son **nom** parmi les personnes enregistrées dans votre carnet d'adresses et affiche dans une nouvelle ligne chaque contact possédant le nom en question. Cette méthode doit accepter deux arguments :
   - Un argument obligatoire : le **nom** du contact à rechercher.
   - Un deuxième argument optionnel : le **prénom** du contact, permettant de réduire le résultat si plusieurs contacts ont le même nom.

Ceci est un exemple qui vous montre comment votre classe doit fonctionner après l'implémentation de ces deux méthodes.

Par exemple, votre carnet d'adresses contient les entrées "Ali Ben Salem" et sa sœur "Sana Ben Salem":

```
>>> a = CarnetAdresses()
>>> a.ajouter_contact(Personne('Ben Salem', 'Ali', ... ))
>>> a.ajouter_contact(Personne('Ben Salem', 'Sana', ... ))
>>> a.chercher_contact('Ben Salem')
Ben Salem, Ali -- Telephone : ...
Ben Salem, Sana -- Telephone : ...
>>> a.chercher_contact('Ben Salem', 'Ali')
Ben Salem, Ali -- Telephone : ...
```

Exercice 7.

1. Créer une classe **Intervalle** possédant une méthode __init__ permettant d'initialiser une borne inférieure et une borne supérieure pour un objet de type Intervalle. Vérifier que les bornes sont **numériques**, **positives**, **non nulles** et **placées dans le bon ordre**, sinon générer une exception de type « **IntervalError** » affichant le message d'erreur « `Erreur : Bornes invalides !` ». Le type « IntervalError » **est une** Exception à définir.
2. En effet, avec les contrôles définis dans la méthode __init__, on ne peut plus créer un intervalle mal formé. Toutefois, il est toujours possible à un programmeur d'écrire directement a.borne_sup = -2, ce qui mettra -2 dans la borne supérieure de l'intervalle a.
   Modifier la **portée** des attributs **borne_inf** et **borne_sup** afin qu'ils ne seront visibles que depuis les méthodes de la classe, mais pas de l'extérieur.
3. Pour modifier une valeur de l'intervalle, écrire maintenant dans la classe Intervalle une méthode **modif_borne_sup** qui permettra de protéger la borne supérieure en ne pouvant y écrire que des nombres supérieurs à la borne inférieure.
4. Ajoutez une méthode **modif_borne_inf** à la classe Intervalle. Faites attention à ce qu'une valeur négative ne puisse pas être enregistrée.
5. Écrivez deux méthodes d'accès **lire_inf(self)** et un **lire_sup(self)**qui retourneront les valeurs des bornes.
6. Ecrire une méthode spéciale **__str__(self)** permettant de retourner une chaine indiquant les valeurs des deux bornes de l'intervalle.
7. Ecrire une méthode spéciale **__contains__(self, val)** qui teste si une valeur val appartient ou non à l'intervalle (cette méthode remplace l'opérateur in).
8. Ecrire une méthode spéciale **__add__(self, autre)** qui retourne un nouvel Intervalle addition des deux intervalles. Exemple : [2,5] + [3,4] = [5,9]
9. Ecrire une méthode spéciale **__sub__(self, autre)** qui retourne un nouvel Intervalle soustraction des deux intervalles. **NB :** [a,b]-[c,d]=[a-d,b-c] ; [2,5]-[3,4]=[-2,2]
10. Ecrire une méthode spéciale **__mul__(self, autre)** qui retourne un nouvel Intervalle multiplication des deux intervalles. Exemple : [2,5] * [3,4] = [6,20]
11. Ecrire une méthode spéciale **__and__(self, autre)** qui retourne l'intersection des deux intervalles et « **None** » si leur intersection est vide. Exemple : [2,5]∩[3,6]=[3,5]
12. Dans le programme principal:
    a. Afficher l'intervalle ["0.35","0.8"]. Que constatez-vous ?
    b. Stocker l'intervalle [1,6] dans une variable **a** et l'intervalle [4,8] dans une variable **b.**
    c. Modifier la valeur de la borne supérieure de **a** à 5.
    d. Afficher l'intersection, la somme, la différence ainsi que le produit des deux intervalles **a** et **b**.
    e. Afficher l'intersection de **a** avec l'intervalle [9,11].

Exercice 8 :

Programmons une classe **Vect2d** dont les objets modéliseront des vecteurs à deux composantes réelles dans un repère orthonormé (O ; I ; J), gradué avec la même unité (OI = OJ = 1 unité). Lorsque le mathématicien dit : « Soit le vecteur $\vec{u}$(3,-2) », le programmeur Python dira : « u = Vect2d (3,-2) ».

1. Programmez le *constructeur*, chargé d'initialiser les attributs x et y de l'instance courante, qui se nomme self. Par défaut, x et y seront mis à 0.
2. Si je veux additionner deux vecteurs, j'ai deux solutions. Ou bien je programme une fonction mathématique ***add(v1,v2)*** à l'extérieur de la classe. Ou bien, dans une optique de pure programmation par objets, je programme **add** comme méthode d'instance à l'intérieur de la classe. Adoptez cette seconde solution, pour laquelle le résultat de l'addition avec la méthode **add** sera un nouveau vecteur.

3. Programmez une méthode ***mul_ext*** réalisant la multiplication $k\vec{u}$ d'un réel *k* par un vecteur $\vec{u}$. Le résultat sera un nouveau vecteur.
4. Programmez une méthode ***zoom*** permettant à un vecteur de se multiplier par *k* et d'être ainsi définitivement modifié ! Faites bien la différence avec ***mul_ext***.
5. Programmez une méthode ***prodscal*** demandant à un vecteur de retourner son produit scalaire avec un autre vecteur.
6. En déduire une méthode ***norme*** demandant à un vecteur de retourner sa longueur.
7. Programmez à l'extérieur de la classe une fonction ***add(v1,v2)*** retournant la somme vectorielle $\overrightarrow{v_1} + \overrightarrow{v_2}$.
   Remarque : En principe il est impossible d'écrire u + v si u et v sont deux vecteurs. Mais en Python, des méthodes spéciales sont cachées sous les opérateurs. Si votre méthode **add** se nomme **__add__**, cela devient possible !
8. Afin de tester votre classe **Vect2d :**
   a. Créez deux vecteurs $\vec{u}$(3,-2) et $\vec{v}$(4,1)
   b. Créez un vecteur $\overrightarrow{w} = \vec{u} + \vec{v}$
   c. Vérifiez que $(\vec{u} + \overrightarrow{w}) \cdot \vec{v} = \vec{u} \cdot \vec{v} + \overrightarrow{w} \cdot \vec{v}$
   d. Vérifiez que $(5\,\vec{u}) \cdot \vec{v} = 5\,(\vec{u} \cdot \vec{v})$

Exercice 9 :

Dans cet exercice on s'intéresse à créer des classes pour gérer les vols d'une compagnie aérienne locale qui organise des vols entre des villes. Plus précisément on s'intéressera aux plans de vol entre les différentes villes. Càd les vols disponibles ainsi que l'heure de départ.

1. Pour créer une classe **Vol_direct** qui représentera un vol direct entre deux villes (pas d'escale dans une ville intermédiaire) , on doit :

1.1. Définir le constructeur de cette classe qui a quatre attributs :

- **dep** et **arr** qui désigne respectivement la ville de départ et la ville d'arrivée
- **jour** qui désigne le jour de la semaine (lundi, mardi, …)
- **heure** (un entier entre 0 et 24 qui représente l'heure de départ)

1.2. Ecrire une méthode **Affiche** qui affiche une chaine bien formatée de la forme :
« Ce vol part de 'Tunis' vers 'Djerba' le 'lundi' à 9 heure »

2. Créer une classe **Vols** qui représentera tous les vols le long de la semaine en utilisant la classe Vol_direct. Pour ce faire on doit :

2.1. Définir le constructeur de cette classe avec un seul attribut qui est une liste de vols

2.2. Ecrire une méthode **Liste_successeurs** qui retourne une liste contenant les villes arrivées d'une ville de départ passée comme paramètre

2.3. Ecrire une méthode **Appartient** qui vérifie si une ville appartient au plan du vol que ce soit comme ville d'arrivée ou de départ

2.4. Ecrire une méthode **Affiche** qui affiche tous les vols directs

3. Ecrire un programme principal permettant de :

a. Créer une liste LV d'objets Vol_direct
   NB : on suppose avoir définie les 3 fonctions suivantes : **Saisie_Jour** qui retourne un jour valide, **Saisie_Heure()** qui retourne une heure valide et **Saisie _Ville()** qui retourne un nom de ville valide.
b. Créer un objet Vol nommé V à partir de la liste déjà créée
c. Afficher tous les vols
d. Saisir une ville qui doit appartenir au plan du vol puis calculer et afficher la liste de ses successeurs

Exercice 10 :

L'objectif de cet exercice est la manipulation des polynômes creux à une seule variable. Un polynôme creux est un polynôme dont certains coefficients sont nuls. Un polynôme est construit à partir de monômes. Un monôme est une expression de la forme $a\,x^n$ où a ($a \neq 0$) est le coefficient du monôme et n ($n > 0$) son degré. Un monôme est représenté par un dictionnaire à un élément dont la clé est le degré n et la valeur est le coefficient a.

Exemple :

Le monôme $8\,x^2$ est représenté par le dictionnaire {2:8 }.

Un polynôme creux est alors défini comme une association de monômes de degrés différents.

Exemples :

Le polynôme $-x^4 + 8\,x^2 - 5x$ est représenté par le dictionnaire { 2:8 , 1:-5 , 4 :-1}.

Le dictionnaire {0:1 , 5:1 , 8 :1} représente le polynôme $x^8 + x^5 + 1$.

On se propose de construire la classe PolynomeCreux à coefficients réels dont le squelette (à compléter) est défini par :

```
class PlynomeCreux :
    """Manipulation des polynômes creux à une seule variable """
    def __init__(self):
        self.data = {} #initialisation à un polynôme nul

    def ajout_monome(self, monome ={}):
        """
        Cette méthode ajoute un monôme saisi au clavier si le paramètre
        monome est nul ou ajoute le monôme nommé sinon
        """
        if len(monome) == 0 :
            #réponse à la question 1
            ⋮
        else :
            #si monome est non vide
            degre = list(monome.keys())[0] # extraction du degré
            coeff = list(monome.values())[0] # extraction du coefficient
            try :
                assert degre >= 0
                assert type(degre) == int
                assert type(coeff) == int or type(coeff) == float
                assert len(monome) == 1
                self.data.update(monome) # self.data[degre] = coeff
            except :
                print ("Erreur d'ajout monome")

    def degree(self):
        #réponse à la question 2
        ⋮

    def __call__(self, x0):
        #réponse à la question 3
        ⋮
    def __add__(self, other): #other est un polynôme creux
        #réponse à la question 4
```

```
        ⋮
    def __mul__(self, other): #other est un polynôme creux
        #réponse à la question 5
        ⋮
    def __str__(self):
        #réponse à la question 6
        ⋮

    def primitive(self):
        #réponse à la question 7
        ⋮
```

Travail demandé :

Question 1 :

Compléter le script de la méthode **ajout_monome**. On rappelle que cette méthode ajoute un monôme saisi au clavier (en faisant les contrôles nécessaires) si le paramètre monome est nul ou ajoute le monôme nommé monome sinon.

Question 2 :

Ecrire le script de la méthode, nommée **degree**, qui retourne le degré du polynôme.

Question 3 :

Ecrire le script de la méthode, nommée **__call__** qui retourne la valeur du polynôme pour un réel x0 donné.

Question 4 :

Ecrire le script de la méthode, nommée **__add__** , qui retourne le polynôme somme de deux polynômes. Remarque : aucun monôme nul ne doit apparaître dans le polynôme résultat.

Question 5 :

Ecrire le script de la méthode, nommée **__mul__** , qui retourne le polynôme produit de deux polynômes. Remarque : aucun monôme nul ne doit apparaître dans le polynôme résultat.

Question 6 :

Ecrire le script de la méthode, nommée **__str__**, qui retourne la chaîne représentant l'expression du polynôme ordonné par ordre décroissant.

Pour le polynôme représenté par { 4:4, 0:4 , 12:6 , 9:1 , 7 :-l}, la chaîne retournée est : " 6*x**12 + x**9 - x**7 + 4*x**4 + 4"

Question 7 :

Ecrire le script de la méthode, nommée **primitive**, qui retourne le polynôme représentant la primitive. On suppose que la constante d'intégration est nulle.

Question 8 :

On définit, l'intégrale d'un polynôme creux P en x entre les bornes a et b, par: $S = \int_a^b P dx$

Ecrire le script de la fonction, nommée **integrale**, permettant de retourner la valeur de ***S*** à partir d'un polynôme ***P***, de type PolynomeCreux, et des bornes d'intégration ***a*** et ***b*** réels.