TP 2. Python : Programmation orientée Objet

Exercice 1.

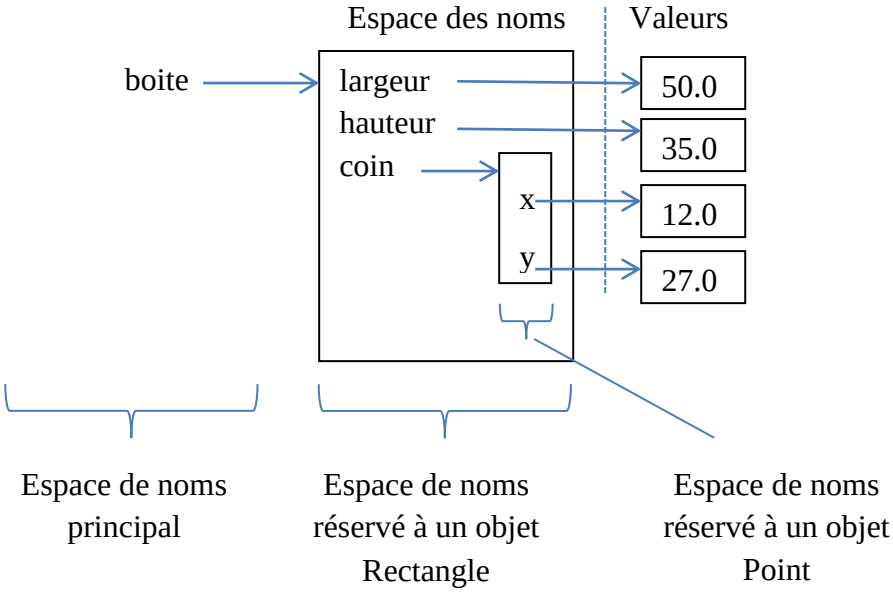
1. Créer une classe Python nommé « **Point** » définie par deux attributs d'instances **x = 3.0** et **y = 4.0** représentant les coordonnées d'un point.

   Les attributs d'instances sont déclarés à l'intérieur de la classe, précisément dans une méthode connue comme « constructeur », et nommée _ _init_ _ et ils seront initialisés au moment de la création d'objets.

   A chaque fois qu'on instancie un nouvel objet par exemple « p » à partir de la classe Point, la méthode constructeur __init__ est appelée automatiquement, non pas par son propre nom « __init__ », mais plutôt avec le nom de la classe à instancier « Point ».
2. Créer deux instances **p1** et **p2** de la classe **Point.** Tester l'égalité des objets **p1** et **p2**.
3. Essayer : `p1 = p2 ; p1.x = 7.0 ; print(p2.x) ;`
4. Définir une méthode **setX(self, valeur)** permettant de modifier l'abscisse d'un point.
5. Définir une méthode **getX(self, valeur)** permettant de retourner l'abscisse d'un point.
6. Définir une méthode **afficher(self)** permettant d'afficher un point.
7. Afficher les objets p1 et p2. `>>> p1` et `>>> print(p1)`
8. Définir les méthode spéciales __rep__(self) et __str__(self).
9. Réafficher les objets p1 et p2.

Exercice 2.

1. Définir une classe Python nommé « **Rectangle** »
2. Créer une instance **boite** de la classe **Rectangle** définie par un point qui représente la position du **coin** supérieur (12.0 ; 27.0) et sa taille : **largeur = 50.0** et **hauteur = 35.0**.



3. Créer une fonction Python **trouverCentre** qui peut être appelée avec un argument de type **Rectangle** et elle renvoie un objet de type **Point**, lequel contiendra les coordonnées du centre du rectangle. Appeler cette fonction en utilisant comme argument l'objet **boite** défini plus haut. Afficher le **centre** trouvé.
4. Modifier la taille (**hauteur** et **largeur**) de l'objet **boite** sans modifier sa position (**coin**) ;

Exercice 3.

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 4.

Soit la classe Time définie comme suit :

```
class Time():
     """défintion d'objets temporels"""
     def __init__(self):
           self.heure = 10
           self.minute = 45
           slef.seconde = 22

     def affiche_heure(self):
           print("{}:{}:{}".format(self.heure, self.minute,
           self.seconde))
```

1. Créer un objet **tstart** de la classe **Time** et tester la méthode **affiche_heure( ).**
2. Modifier la méthode __init__ afin qu'elle comporte 3 paramètres, avec pour chacun une valeur par défaut.
3. Créer un objet **recreation** de la classe **Time**, en transmettant des arguments à sa méthode constructeur. Tester la méthode **affiche_heure( ).**

Exercice 5.

1. Créer une classe **Intervalle** possédant les attributs et les méthodes suivantes :
   a. Deux données (attributs) : une **borne inférieure** et une **borne supérieure**.
   b. La méthode **__init__** qui permet d'initialiser les objets de cette classe aura donc trois paramètres : le premier étant l'objet que l'on initialise (traditionnellement nommé **self**) et deux paramètres qui permettent d'initialiser les bornes.
2. Essayer : `>>> a = Intervalle(2,3) ; print(a) ;`
   Définir dans la classe Intervalle une méthode **__str__(self)** qui permet d'afficher l'intervalle a sous cette forme : `>>> print(a) # <2, 3>`

Exercice 6.

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 7.

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.

Les méthodes suivantes seront définies :

1. **choix_conducteur ( )** permettra de désigner (ou changer) le nom du conducteur.
2. **accelerer (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 ( )** 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 8. [Héritage]

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 9. [Composition]

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 10 .

1. Définissez une classe **Point**. Un Point est définit par deux coordonnées x et y de type réel.
2. Définissez une classe **Cercle**. Un cercle est définit par un **centre** de type *Point* et un **rayon** de type *réel*. Les objets construits à partir de la classe Cercle seront des cercles de tailles variées. Le constructeur utilisera donc deux paramètres : le centre du cercle et le rayon.
3. Définissez dans la classe Cercle une méthode **surface** ( ), qui devra renvoyer la surface du cercle.
4. Définissez ensuite une classe **Cylindre** dérivée de la classe Cercle. Le constructeur de cette nouvelle classe comportera les trois paramètres : le centre, le rayon et la hauteur du cylindre.
5. Vous y ajouterez une méthode **volume** ( ) qui devra renvoyer le volume du cylindre (rappel : volume d'un cylindre = surface de section × hauteur).

   Exemple d'utilisation de cette classe :

```
>>> cyl = Cylindre(p, 5, 7)
>>> print(cyl.surface())
>>> print(cyl.volume())
```

6. Ajouter une classe **Cone**, qui devra dériver cette fois de la classe Cylindre, et dont le constructeur comportera lui aussi les trois paramètres : le centre, le rayon et la hauteur du cone.
7. Cette nouvelle classe possédera sa propre méthode **volume**( ), laquelle devra renvoyer le volume du cône (rappel : volume d'un cône = volume du cylindre correspondant divisé par 3).
   Exemple d'utilisation de cette classe :
   ```
   >>> co = Cone(p, 5, 7)
   >>> print(co.volume())
   ```
8. Ecrire un programme principal permettant de tester ces classes.

Exercice 11.

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].

Attributs d'instances et attributs de classes :

Les attributs de classes : sont des attributs déclarés en dehors du constructeur, accessibles via le nom de la classe et leurs valeurs sont partagés entre tous les objets de cette classe.

Les attributs d'instances : sont des attributs déclarés à l'intérieur du constructeur, accessibles via le nom de l'instance (objet) et leurs valeurs ne sont pas partagés entre les objets de cette classe.

Remarque : il est toujours possible en Python, après la définition d'une classe, d'ajouter des attributs pour un objet instance de cette classe.

Exercice : Soit les classes A et B suivantes. Quelle sera la sortie du programme suivant ?

```
1.  class A:
2.    """ Déf de la classe A """
3.        i = 1
4.        k = 3
5.        def __init__(self):
6.            self.i = 2
7.        def f(self):
8.            return self.g()
9.        def g(self):
10.           return 'A'
11. class B(A):
12.     def g(self):
13.           return 'B'
```

```
1.  a = A(); b = B() ;
2.  type(a) ; help(a) ;
3.  print(a.g(), b.g()) # affiche : ?
4.  print(a.f(), b.f()) # affiche : ?
5.  print(A.i) # affiche ?
6.  print(A().i) # affiche ?
7.  #Ajouter un attribut 'j' à l'objet a
8.  a.j = 0
9.  print(a.j) # affiche ?
10. print(b.j) # affiche ?
11. b = a ; print(b.j) ;
12. b.j = -1 ; print(a.j) # affiche ?
```