TD: programmation orientée objet:

Exercice 1: On s'interesse au programme suivant:

```python
class Personne:
    """Classe representant une personne"""

    def __init__(self, nom, prenom):
        """Constructeur de notre classe"""
        self.nom = nom
        self.prenom = prenom
        self.age = 33

    def __repr__(self):
        """Quand on entre notre objet dans l'interpreteur"""
        return "Personne: nom({}), prenom({}), age({})".format(
                self.nom, self.prenom, self.age)
```

personne_bis.py

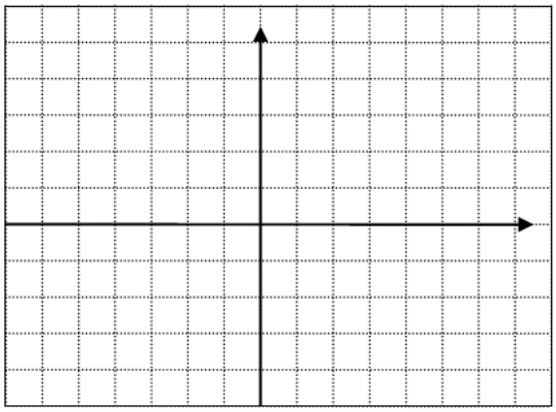
1. Quel type d'objet est représenté par la classe Personne?
2. Expliquer le rôle de la première méthode. Quels en sont les attibuts?
3. Expliquer le rôle de l'autre méthode.
4. A l'aide de l'éditeur Python:
   - 4.1 Créer un homme ayant pour prénom Tryphon et pour nom Tournesol.
   - 4.2 Afficher son âge.
   - 4.3 Tournesol a en réalité 61 ans. Modifier son âge et l'afficher à nouveau.
   - 4.4 Ajouter à la classe un attribut lieu_residence qui serait par défaut Paris.

Exercice 2: On s'interesse au programme suivant:

```
class Point:                                #Classe Point
    "Definition d'un point dans le plan"

    def __init__(self, a = 0, b = 0):       #premiere methode (constructeur)
        self.x = a                          #a et b sont les attributs
        self.y = b

    def dist_origine(self):                 #deuxieme methode
        return (self.x**2 + self.y**2)**0.5

    def dist(self,p):                       #troisieme methode
        return ((p.x-self.x)**2 + (p.y-self.y)**2)**0.5

    def __repr__(self):                     #quatrieme methode (permet l'affichage)
        return "(" + str(self.x) + ";" + str(self.y) + ")"
```

point.py

1. Quel type d'objet est représenté par la classe Point?
2. Expliquer le rôle de la première méthode. Quels en sont les attibuts?
3. Expliquer le rôle des trois autres méthodes.
4. A l'aide de l'éditeur Python:
   - 4.1 Créer trois points p1, p2 et p3 de coordonnées (1,2), (5,4) et (5,4). Les dessiner sur le graphe qui suit.
   - 4.2 Afficher l'abcisse de p1.
   - 4.3 Afficher l'ordonnée de p1.
   - 4.4 Afficher la distance à l'origine de p1.
   - 4.5 Afficher la distance entre p1 et p2
   - 4.6 Que renvoie p1 == p2 ? Comment interpréter ce résultat?

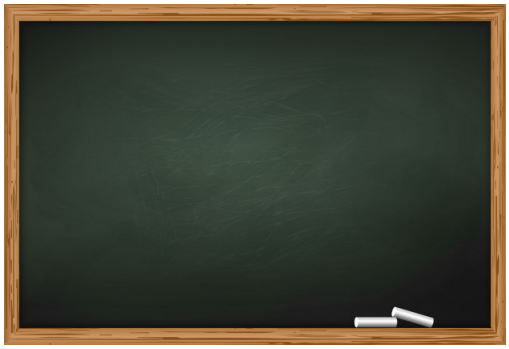


Exercice 3: On s'interesse au programme suivant:

```
class TableauNoir:
    """Classe definissant une surface sur laquelle on peut ecrire,
    que l'on peut lire et effacer, par jeu de methodes. L'attribut modifie
    est 'surface'"""


    def __init__(self):
        """Par defaut, notre surface est vide"""
        self.surface = ""

    def ecrire(self, message_a_ecrire):
        """Methode permettant d'ecrire sur la surface du tableau.
        Si la surface n'est pas vide, on saute une ligne avant de rajouter
        le message a ecrire"""

        if self.surface != "":
            self.surface += "\n"
        self.surface += message_a_ecrire

    def lire(self):
        """Cette methode se charge d'afficher, grace a print,
        la surface du tableau"""
        print(self.surface)
```

tableau.py

1. Quel type d'objet est représenté par la classe TableauNoir?
2. Expliquer le rôle de la première méthode. Quels en sont les attibuts?
3. Expliquer le rôle des autres méthodes.
4. A l'aide de l'éditeur Python:
   - 4.1 Créer un tableau vierge et le lire.
   - 4.2 Écrire un titre et deux courts paragraphes sur ce tableau. Les lire.
   - 4.3 Ajouter une méthode permettant d'effacer le tableau.
   - 4.4 Utiliser cette méthode pour effacer le contenu du tableau et lire à nouveau le tableau.



Exercice 4: surcharge d'opérateur: On s'interesse au programme suivant:

```python
class Vecteur:
    """Classe representant un vecteur en dimension 2"""


    def __init__(self, a = 0, b = 0):
        """Constructeur de la classe"""
        self.x = a
        self.y = b

    def __str__(self):
        """Affichage un peu plus joli de nos vecteurs"""
        return "({},{})".format(self.x, self.y)

    def __mul__(self, objet_a_multiplier):
        nouveau_vect = Vecteur()
        nouveau_vect.x = self.x*objet_a_multiplier
        nouveau_vect.y = self.y*objet_a_multiplier
        return nouveau_vect

    def __rmul__(self, objet_a_multiplier):
        """Cette methode est appelee si on ecrit 4 * objet et que
        le premier objet (4 dans cet exemple) ne sait pas comment multiplier
        le second. On se contente de rediriger sur __mul__ puisque,
        ici, cela revient au meme : l'operation doit avoir le meme resultat,
        posee dans un sens ou dans l'autre"""
        return self * objet_a_multiplier
```

vecteur.py

1. Quel type d'objet est représenté par la classe Vecteur?
2. Expliquer le rôle de la première méthode. Quels en sont les attibuts?
3. Expliquer le rôle de la seconde méthode?
4. Expliquer le rôle des autres méthodes.
5. A l'aide de l'éditeur Python:
   - 5.1 Créer un premier vecteur ayant pour coordonnées (2,3). l'afficher.
   - 5.2 Multiplier ce vecteur par 4 et afficher le résultat. On vérifiera la commutativité de la classe.