Exercice 3 : Correction

1. Implémenter la class **Carte**.

```python
# Dans l'éditeur PYTHON
class Carte:    # Définition de la classe
    "Une carte d'un jeu de 32 ou 52 cartes"
    couleurs= [chr(9824),chr(9825),chr(9826),chr(9827)]
    valeurs = {'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'10':10,
                       'V':11,'D':12,'R':13,'A':14}

    def __init__(self,valeur,couleur):   #constructeur
        self.valeur=valeur # 1er attribut  {de 2  à 14}
        self.couleur=couleur # {'pique', 'carreau', 'coeur', 'trefle'}

    def getAttributs(self): # méthode 2 : accesseur
        return (self.valeur,self.couleur)

    def getValeur(self): # méthode 3 : accesseur
        return   self.valeur

    def getCouleur(self): # méthode 4 : accesseur
        return   self.couleur

    def setValeur(self,v):  # méthode 5  : mutateur de l'attribut valeur
        if 2<=int(v)<=14: # contrôle la validite de v
            self.valeur=v
            return True
        else:
            return False

    def __repr__(self):
        '''(Carte)->str
        retourne une représentation de l'objet (print)'''
        return 'Carte : '+self.valeur+' de '+self.couleur

    def __eq__(self, autre):
        '''(Carte,Carte)->bool
        self == autre si la valeur et la couleur sont les memes'''
        return self.valeur==autre.valeur and self.couleur==autre.valeur

    def __lt__(self,autre):  #(< less than)
        '''
        (Carte,Carte)->bool
        self <  autre  si la valeur de self est < à la valeur de autre'''
        return  self.valeur < autre.valeur

    def __gt__(self,autre):  #(> great than)
        '''
        (Carte,Carte)->bool
        self >  autre  si la valeur de self est > à la valeur de autre'''
        return  self.valeur > autre.valeur
```

2. Tester la fonction ci-dessous :

```
def testCarte():
        valetCoeur=Carte('Valet','COEUR')
        print('Couleur:', valetCoeur.getCouleur())
        print('Valeur:', valetCoeur.getValeur())
        print('Valeur modifiee:', valetCoeur.getValeur())
```

3. Écrire un programme qui va :

a. Créer la carte 3 de COEUR que l'on nommera **c1**.

b. Créer la carte 10 de PIQUE que l'on nommera **c2**.

c. Afficher **c1** et **c2**.

d. Tester si la valeur de **c1** est supérieure à celle de **c2**.

e. Modifier la valeur de la carte **c2** en 4.

f. Tester si les cartes **c1** et **c2** ont la même valeur.

g. Tester si la valeur de **c1** est supérieure à celle de **c2**.

```
c1=Carte('3','COEUR')
c2=Carte('10','PIQUE')
print(c1)
print(c2)
print(c1>c2)
c2.setValeur('4')
print(c2.getValeur()==c1.getValeur())
print(c1>c2)
```

1 Notion d'agrégation

La conception d'une classe a pour but généralement de pouvoir créer des objets qui suivent tous le même modèle de fabrication. Un objet dans la vraie vie, par exemple votre stylo, est composé d'autres objets : une pointe (ou plume), un réservoir d'encre, éventuellement un capuchon et un ressort.... Votre stylo est ce qu'on appelle un **objet agrégat** et son réservoir d'encre est donc un **objet composant**. Nous avons créé une classe **Carte** qui permet de créer une carte à jouer standard. Un jeu de cartes est donc un **objet agrégat** de 32 ou 52 cartes à jouer. Si un jeu de carte est un objet, on peut donc définir une classe **JeuDeCartes**.

Cette classe a (entre autre) pour attributs :

— 32 ou 52 instances (objets) de la classe Carte.

— le nombre de cartes que le jeu contient.

Exercice 5 : Correction

1. Implémenter la classe **JeuDeCartes**

```
# Dans l'éditeur PYTHON
from random import shuffle
class JeuDeCartes:
    def __init__(self):
        'initialise le paquet de 52 cartes'
        self.paquet = []          # paquet vide au debut
        self.nombreCarte=0
        for couleur in Carte.couleurs:
            # Carte.couleurs -> Va chercher la variable couleurs dans la classe Carte
            for valeur in Carte.valeurs:
            # Carte.valeurs -> Va chercher la variable valeurs dans la classe Carte
                # ajoute une Carte de valeur et couleur
                self.paquet.append(Carte(valeur,couleur))
                self.nombreCarte=self.nombreCarte+1

    # Accesseur de l'attribut self.nombreCarte
    def getNombreCartes(self):
        return  self.nombreCarte

    # Accesseur de self.paquet
    def getPaquet(self):
        return self.paquet

    def tireCarte(self):
        '''(JeuDeCartes)->Carte
        distribue une carte, la premiere du paquet'''
        return self.paquet[0]

    def battre(self):
        '''(JeuDeCartes)->None
        pour battre le jeu des cartes'''
        return shuffle(self.paquet)

    def __repr__(self):
        '''retourne une representation de l'objet'''
        return str(self.paquet)
```

2. Vérifier le bon fonctionnement des commandes de l'exercice précédent.
3. Tester la fonction ci-dessous :

```python
def testJeuDeCarte():
        jeu52=JeuDeCartes()
        jeu52.battre()
        L=jeu52.getPaquet()
        print(jeu52.getNombreCartes())
        carte=L[2]
        print('Couleur:', carte.getCouleur())
        print('Valeur:', carte.getValeur())
        jeu52.tireCarte()
        print(jeu52.getNombreCartes())
```

Exercice 6 : Correction

1. Créer une classe **Joueur** ayant les attributs suivants :
   — **nom** : Nom du joueur ;
   — **nbCartes** : Correspond au nombre de cartes dans la main du joueur ;
   — **mainJoueur** : Liste des cartes (objets de type Carte) dans la main du joueur.
   Cette classe devra implémenter les méthodes suivantes :
   — **getNom()** : Accesseur de l'attribut **nom** ;
   — **getNbCartes()** : Accesseur du champ **nbCartes** ;
   — **setMain()** : Définit la main du joueur, donc la liste de ses cartes au début du jeu ;
   — **jouerCarte**() : Enlève et renvoie la dernière carte (objet de type Carte) de la main du joueur pour la jouer, ou retourne **None** s'il n'y a plus de cartes dans la main du joueur ;
   — **remporteCarte**(liste de cartes) : Fonction qui insère les cartes de la liste de cartes donnée en paramètre.
   — **__repr__()** : Qui affiche les cartes de la main du joueur.

```python
class Joueur:
    def __init__(self):
        self.nom=""
        self.mainJoueur=[] # main vide au debut
        self.nbCartes=len(self.mainJoueur) # main vide au debut

    def getNom(self):
        return  self.nom

    def getnbCartes(self):
        return  self.nbCartes

    def getMain(self):
        return  self.mainJoueur

    def setMain(self,ListeCartes):
        self.mainJoueur=ListeCartes
        self.nbCartes=len(self.mainJoueur)

    def jouerCarte(self):
        if self.nbCartes!=0:
            self.nbCartes=self.nbCartes-1
            carte=self.mainJoueur[0]
            self.mainJoueur=self.mainJoueur[1::]
            return carte
        else :
            return None
    def remporteCarte(self,ListeCarte):
        self.mainJoueur=self.mainJoueur+ListeCarte
        self.nbCartes=len(self.mainJoueur)

    def __repr__(self):
        '''retourne une representation de l'objet'''
        return "La main de " + self.nom + " contient " +
        str(self.getnbCartes()) + " cartes  : " + str(self.mainJoueur)
```

2. Comme pour les exercices précédents, créer une fonction **testJoueur()** qui doit permettre de vérifier la bonne implémentation de la classe **Joueur**.

```python
def testJoueur():
    jeu52=JeuDeCartes()
    jeu52.battre()
    ben=Joueur()
    ben.nom="M.Fourlegnie"
    ben.mainJoueur=jeu52.getPaquet()[0:3]
    ben.nbCartes=len(ben.mainJoueur)
    print(ben)
    print(ben.jouerCarte())
    print(ben)
    ben.remporteCarte(jeu52.getPaquet()[5:7]) # On ajoute 2  plis
    print(ben)
```

2 Exercice bilan

Exercice n° 1 Bilan

On suppose écrites les classes **Piece** et **Appartement**, dont on vous donne les en-têtes de méthodes :

```
# Dans l'éditeur PYTHON
class Piece:
    # nom est une string et surface est un float
    def __init__(self,nom,surface):
        self.nom=nom
        self.surface=surface

    def getSurface(self):
        return self.surface
    def getNom(self):
        return self.nom

    def setNom(self,nom): # nom est un string,
                ...

    def setSurface(self,s): # s est un float,
        ...

class Appartement:
    # nom est une string
    def __init__(self,nom):
        self.listeDePieces=[] # Liste d'instances de la classe Piece
        self.nom=nom

    def getNom(self):
        return self.nom

    def ajouter(self,piece):
        # pour ajouter une pièce de classe Piece
        ...

    def nbPieces(self):
        # pour avoir le nombre de pièces de l'appartement
        ...


    def getListePieces(self):
        # retourne la liste des pièces avec les surfaces
            ...


    def SurfaceTotale(self):
        # retourne la surface totale de l'appartement (un float)




                ...
```

1. En utilisant les deux classes ci-dessus, écrire (sur feuille) le code Pythan qui va :
   - **a.** créer une pièce « chambre1 » , de surface 20 m^2 et une pièce « chambre2 » », de surface 15 m^2 ,
   - **b.** créer une pièce « séjour » », de surface 25 m^2 et une pièce « sdb » », de surface 10 m^2 ,
   - **c.** créer une pièce « cuisine » », de surface 12 m^2 ,
   - **d.** créer un appartement « appart205 » qui contiendra toutes les pièces créées,
   - **e.** afficher la surface totale de l'appartement créé.
   - **f.** afficher la liste des pièces et surfaces de l'appartement créé.
2. **a.** Compléter les méthodes accesseurs et mutateurs de la classe Piece.
   - **b.** Compléter la méthode **ajouter(self,piece)**, qui permet d'ajouter une pièce de la liste de pièces présentes dans l'appartement.
   - **c.** Compléter la méthode **nbPieces(self)**, qui permet de retourner le nombre de pièce présentes dans l'appartement.
   - **d.** Compléter la méthode **getListePieces(self)**, qui renvoie la liste des pièces de l'appartement.
   - **e.** Compléter la méthode **getSurfaceTotale(self)**, qui renvoie la surface totale de l'appartement.
   - **f.** Créer un tableau qui classe les méthodes de ces deux classes selon leur type : constructeur, accesseur, mutateur ou autre.
3. Implémenter les classes ci-dessus et valider votre travail avec le test suivant :

```
# Dans l'éditeur PYTHON
a=Appartement('appt25')
p1=Piece("chambre", 11.1)
p2=Piece("sdbToilettes", 7)
p3=Piece("cuisine", 7)
p4=Piece("salon", 21.3)
print(p4.getNom(),p4.getSurface())
p1.setSurface(12.6)
a.ajouter(p1)
a.ajouter(p2)
a.ajouter(p3)
a.ajouter(p4)
print(a.getNom(),a.getListePieces())
print('nb pieces =', a.nbPieces(),', Surface totale =',a.SurfaceTotale())
```

Et devra retourner :

```
# Dans la console PYTHON

salon 21.3
appt25 [('chambre', 12.6), ('sdbToilettes', 7), ('cuisine', 7),
('salon', 21.3)]
nb pieces = 4 , Surface totale = 47.9
```

4. En s'inspirant du dernier **print**, créer une méthode __**repr**__ permettant d'avoir un descriptif de l'appartement. (nom de l'appartement, liste des pièces avec surface de chacune, nombre de pièces et surface totale)