Exercice 1 : utilisation d'objet

On suppose écrite la classe **Carte** dont on vous donne les en-têtes de méthodes. Cette classe diffère légèrement de la classe **Carte** vue plus haut, prenez le temps d'analyser ce code et de voir les différences :

```
class Carte:
  def __init__(self, nom, couleur):
    # Affectation de l'attribut nom  et  de l'attribut couleur
    couleur = ('CARREAU', 'COEUR', 'TREFLE', 'PIQUE')
    noms = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Valet',  'Dame',
      'Roi', 'As']
    valeurs = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
    '10': 10, 'Valet': 11, 'Dame': 12, 'Roi': 13, 'As': 14}

  def setNom(self, nom):
    # Mutateur de l'attribut nom (de la liste noms)

  def getNom(self):
    # renvoie le  nom  de la carte (de la liste noms): Accesseur

  def getCouleur(self):
    # renvoie la couleur de la carte (de la liste couleur): Accesseur

  def getValeur(self):
    # renvoie la valeur de la carte (du dictionnaire valeurs) : Accesseur

  def egalite(self, carte):
    ''' Renvoie True si les cartes self et carte ont même valeur, False sinon
    carte: Objet de type Carte    '''

  def estSuperieureA(self, carte):
    ''' Renvoie True si la valeur de self est supérieure à celle de carte,
    False sinon
    carte: Objet de type Carte
    '''

  def estInferieureA(self, carte):
    ''' Renvoie True si la valeur de self est inférieure à celle de carte,
    False sinon
    carte: Objet de type Carte
    '''
```

Écrire (sur feuille) un programme principal qui va :

1. Créer la carte Valet de COEUR que l'on nommera c1.
2. Afficher le nom, la valeur et la couleur de c1.
3. Créer la carte As de PIQUE que l'on nommera c2.
4. Afficher le nom, la valeur et la couleur de c2.
5. Modifier le nom de la carte c2 en Roi et afficher le nom, la valeur et la couleur de c2.
6. Créer la carte 8 de TREFLE que l'on nommera c3.
7. Comparer les cartes c1 et c2 puis c1 et c3.

Exercice 2 : utilisation d'objet

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

    # Accesseurs: retournent les attributs d'un objet de cette classe
    def getSurface(self):
        return self.surface
    def getNom(self):
        return self.nom

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

class Appartement:
    # nom est une string
    def __init__(self,nom):
        # L'objet est une liste de pièces (objets issus de la classe Piece)
        self.listeDePieces=[]
        self.nom=nom

    def getNom(self):
    # Accesseurs:
        return self.nom

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

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

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

    # retourne la liste des pièces avec les surfaces
    def getListePieces(self): # sous forme d'une liste de tuples
        ...
```

Écrire (sur feuille) un programme principal utilisant ces deux classes qui va :

1. créer une pièce « chambre1 » , de surface 20 m^2 et une pièce « chambre2 » », de surface 15 m^2 ,
2. créer une pièce « séjour » », de surface 25 m^2 et une pièce « sdb » », de surface 10 m^2 ,
3. créer une pièce « cuisine » », de surface 12 m^2 ,
4. créer un appartement « appart205 » qui contiendra toutes les pièces créées,
5. afficher la surface totale de l'appartement créé.
6. afficher la liste des pièces et surfaces de l'appartement créé.

Exercice 3 : rédaction des méthodes

On repose sur les mêmes classes que dans l'exercice 1 : **Piece** et **Appartement**. Les en-têtes de méthodes sont les mêmes mais vous allez devoir les compléter(...) :

```
# Proposition   expert:
# Dans l'éditeur PYTHON
class Piece:
    # nom est une string et surface est un float
    def __init__(self,nom,surface):
        # chaque objet a pour attributs le nom de la pièce(string)
        # et la surface de celle ci(float) en m2.
        # on doit rentrer le couple nom de la pièce et la surface
        # pour chaque pièce.
        ...
    # Accesseurs: retournent les attributs d'un objet de cette classe
    def getNom(self):
        ...
    def getSurface(self):
        ...
    # Mutateur: modifient les attributs, ici la surface d'une pièce déjà
    # renseignée
    def setSurface(self,s): # s est un float
        ...

class Appartement:
    # nom est une string
    def __init__(self,nom):
    #nomme l'appartement et une liste de pièces vide à remplir
        ...
    def ajouter(self,piece):
    # ajoute une pièce (instance(=objet) de la classe Piece)

    def nbPieces(self):
    # retourne le nombre de pièces de l'appartement
        ...
    def getSurfaceTotale(self):
    # retourne la surface totale de l'appartement (un float)
        ...
    def getListePieces(self): # retourne la liste des pièces
        ...
```

Consignes :

1. Écrire les méthodes constructeurs des deux classes.
   En cas de difficulté, rendez vous à la version intermédiaire. (Différenciation)
2. Finaliser la classe Piece.
   (a) Écrire les méthodes accesseurs et mutateurs de la classe Piece.
3. Finaliser la classe Appartement.
   (a) Écrire la méthode qui permet d'ajouter une pièce(ajouter(self,piece)) de la liste de pièces présentes dans l'appartement. (Rappel utile a)
   (b) Écrire la méthode qui permet de retourner le nombre de pièces(nbPieces(self)) présentes dans l'appartement. (Rappel utile b)
   (c) Écrire la méthode getSurfaceTotale(self), qui renvoie la surface totale de l'appartement. (Rappel utile b)
   (d) Écrire la méthode getListePieces(self), qui renvoie la liste des pièces de l'appartement.
4. Créer un tableau qui classe les méthodes de ces deux classes selon leur type : constructeur, accesseur, mutateur ou autre.

Aide

Rappels utiles :

— (a) pour ajouter un élément à la fin d'une liste : *nomDeListe.append(élément)*;

— (b) la longueur d'une listel s'obtient avec *len(l)*.

Le test sera le 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
```

```
# Proposition  intermédiaire:
# Dans l'éditeur PYTHON

class Piece:
    # nom est une string et surface est un float
    def __init__(self,nom,surface):
        # chaque objet a pour attributs le nom de la pièce(string)
        # et la surface de celle ci(float) en m2.
        # on doit rentrer le couple nom de la pièce et la surface
        # pour chaque pièce.
        self.nom=nom
        self.surface=surface
    # Accesseurs: retournent les attributs d'un objet de cette classe
    def getNom(self):
        return self.surface
    def getSurface(self):
        ...
    # Mutateur: modifient les attributs, ici la surface d'une pièce
    # déjà renseignée
    def setSurface(self,s): # s est un float
        ...

class Appartement:
    # nom est une string
    def __init__(self,nom):
    #nomme l'appartement et une liste de pièces vide à remplir
        self.listeDePieces=[]
        self.nom=nom
    def ajouter(self,piece):
    # ajoute une piece (instance(=objet) de la classe Piece)

    def nbPieces(self):
    # retourne le nombre de pièces de l'appartement
        ...
    def getSurfaceTotale(self):
    # retourne la surface totale de l'appartement (un float)
        ...
    def getListePieces(self): # retourne la liste des pieces
        ...
```