TP. Programmation Orientée Objet en Python

Corrigé

Exercice 5 :

```
#Q1
class Personne: #class Personne(): # class Personne(object):
    def __init__(self,nom, prenom, tel, email):
        self.nom = nom
        self.prenom = prenom
        self.tel = tel
        self.email = email

    def __str__(self):
        #doit retourner une chaine (pas print)
        ch ="{}, {} --Téléphone: {} -- Email :{}"
        return ch.format(self.nom,self.prenom, self.tel,self.email)

#Q2
class Travailleur(Personne):
    def __init__(self,nom,prenom,tel,email,nomEntr,adrEntr,telEntr):
        Personne.__init__(self,nom, prenom, tel, email)
        self.nomEntr = nomEntr
        self.adrEntr = adrEntr
        self.telEntr = telEntr

    def __str__(self):
        ch1 = Personne.__str__(self)
        ch2 = ", Entreprise : {},Adresse Entreprise: {} , Tel Entreprise : {}"
        ch2 = ch2.format(self.nomEntr,self.adrEntr,self.telEntr)
        return ch1 + ch2

#Q4
class Scientifique(Travailleur):
   def __init__(self,nom,prenom,tel,email,nomEntr,adrEntr,telEntr,ds,typeSci):
       Travailleur.__init__(self,nom,prenom,tel,email,nomEntr,adrEntr,telEntr)
       self.ds = ds
       self.typeSci = typeSci #doit être de type list

   def __str__(self):
       ch1 = Travailleur.__str__(self)
       ch2 = ", Discipline : {}, Type Scientifique: {} "
       ch2 = ch2.format(self.ds,"-".join(self.typeSci))
       return ch1 + ch2


#Q4
#programme principal
amir = Personne("ben salem","amir","4444444","amir@gmail.com")
print(amir)
```

```
# print(amir) fait appel à str(amir) donc : print(str(amir))
# et
# str(amir) fait appel à Personne.__str__(amir)

ali = Travailleur("ben salem","ali","4444444",\
"ali@gmail.com","ipein","mrazga","72000000")
print(ali)

ahmed =  Scientifique("ben salem","ali","4444444","ali@gmail.com",\
"ipein","mrazga","72000000","Chimie",["info","théorique"])
print(ahmed)
```

Exercice 6 :

```
class CarnetAdresses():
    """
    CarnetAdresses contient des Personnes : PAS d'héritage
    CarnetAdresses enregistre des Personnes ==> Composition

    Donc, on a besoin d'un attribut de type composé et modifiable

    Les tructures de données composées et modifiables sont : les listes,
    les dictionnaires et les ensembles

    On choisira pour cet execerice : une liste

    Dans cette liste nommée "contacts" on peut ajouter, modifier et
    supprimer des personnes.
    """

    def __init__(self):
        self.contacts = [] #liste de Personnes initialement vide

    #Q2
    def ajouter_contact(self,p): #p est une personne
        self.contacts.append(p)

    def supprimer_contact(self,p):#p est une personne
        if p in self.contacts:
            self.contacts.remove(p)

    #Q3
    def __str__(self): #ici self est un carnet d'adresses
        # __str__ doit retourner une chaine(pas return liste:pas print(liste))
        #etape 1 : convertir  les elements de la liste contacts
        # de type Personne en type str
        l=[]
        for p in self.contacts:
            l.append(str(p)) #appel à Personne.__str__(p) avec p une Personne
        #etape 2 : jointure des élements de la liste en une seule chaine
```

```
        ch = "\n".join(l)
        #etape 3: retourner cette chaine
        return ch

    #Q4
    def chercher_contact(self,nom, prenom=None): #None => vide ,null, rien
        for p in self.contacts:
            #si le nom est donné et le prenom est non donné
            if(p.nom.lower() == nom.lower() and prenom == None) or \
            #ou bien : si le nom et le prenom sont donnés
            (p.nom.lower() == nom.lower() and \
             p.prenom.lower() == prenom.lower()):
                print(p)

#programme principal
a = CarnetAdresses()
print(a.contacts) #affiche une liste des personnes (objets de type Personne)
                  #Cette liste est non lisible

#Créer un objet de type Personne
amir = Personne("ben SAlem","amir","4444444","amir@gmail.com")
# amir : est une instance de la classe Personne

#Ajouter l'instance amir à la liste contacts de la classe CarnetAdresses
a.ajouter_contact(amir)
print(a.contacts) #afficher le contenu de la liste contacts

#Créer un objet de type Personne
ali = Personne("ben sAlem","ali","4444444","ali@gmail.com")
a.ajouter_contact(ali)

print(a) #Afficher le contenu du carnet d'adresses
# a : est un carnet d'adresses
# print(a) fait appel à str(a) donc : print(str(a))
# et
# str(a) fait appel à CarnetAdresses.__str__(a)

#Question 4
print("Resulats de recherche :")
#Recherche avec le nom de la personne
a.chercher_contact("Ben salem")

#Recherche avec le nom et le prenom de la personne
a.chercher_contact("Ben salem","amir")
```

Exercice 7 :

```
class Intervalle :
    def __init__(self,bi,bs):
        assert (type(bi)==int or type(bi)==float) and \
               (type(bs)==int or type(bs)==float) and \
               (0 < bi < bs), "Erreur : Bornes invalides !"
        self.__binf = bi
        self.__bsup = bs

    def modif_borne_inf(self, valeur):
        assert(type(valeur)==int or type(valeur)==float) and \
              (0 < valeur < self.__bsup),"Erreur : Bornes invalides !"
        self.__binf = valeur

    def lire_borne_inf(self):
        return self.__binf

    def modif_borne_sup(self, valeur):
        assert(type(valeur)==int or type(valeur)==float) and \
              (valeur > self.__binf),"Erreur : Bornes invalides !"
        self.__bsup = valeur

    def lire_borne_sup(self):
        return self.__bsup

    def __str__(self):
        return "<{},{}>".format(self.__binf,self.__bsup)

    def __contains__(self,val):
        return self.__binf < val < self.__bsup

    def __add__(self,autreIntervalle):
        binf = self.__binf + autreIntervalle.lire_inf()
        bsup = self.__bsup + autreIntervalle.lire_sup()
        intervalle = Intervalle(binf, bsup)
        return intervalle

    def __sub__(self,autreIntervalle):
        binf = self.__binf - autreIntervalle.lire_sup()
        bsup = self.__bsup - autreIntervalle.lire_inf()
        intervalle = Intervalle(binf, bsup)
        return intervalle

     def __mul__(self,autreIntervalle):
        binf = self.__binf * autreIntervalle.lire_inf()
        bsup = self.__bsup * autreIntervalle.lire_sup()
        return Intervalle(binf, bsup)
```

```
     def __and__(self,autreIntervalle):
        binf = max(self.__binf , autreIntervalle.lire_inf())
        bsup = min(self.__bsup , autreIntervalle.lire_sup())
        return Intervalle(binf, bsup)

#Q12
i1 = Intervalle(float("0.35"),float("0.8"))
a= Intervalle(1,6)
b= Intervalle(4,8)
a.setBSup(5)
print(a+b)
print(a&b)
#...
```

Exercice 8 :

```
#Q1
class Vect2d :
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
#Q2
    def add(self,autre):
        v = Vect2d()
        v.x = self.x + autre.x
        v.y = self.y + autre.y
        return v
#Q3
    def mul_ext(self, k):
        v = Vect2d()
        v.x = self.x * k
        v.y = self.y * k
        return v
#Q4
    def zoom (self, k):
        self.x *= k
        self.y *= k

#Q5
    def prodscal (self,autre):
        x = self.x * autre.x
        y = self.y * autre.y
        return x+y
#Q6
    def norme(self):
        from math import sqrt
        return sqrt(self.x **2 + self.y **2)


#Q7
#En déhors de la classe
```

```
def add(v1,v2):
    v = Vect2d()
    v.x = v1.x + v2.x
    v.y = v1.y + v2.y
    return v

#Q8
#Programme principal
#a
u = Vect2d(3,-2)
v = Vect2d(4,1)
#b
w = u.add(v)
print(w.x, w.y)
#c
ps1 = w.add(u).prodscal(v)
ps2 = u.prodscal(v) + w.prodscal(v)
print(ps1 == ps2 )

#d
from copy import copy
u1 = copy(u)
u1.zoom(5) #5u
ps1 = u1.prodscal(v)

ps2 = u.prodscal(v)
ps2 *=5

print(ps1 == ps2 )

#e
a = u.prodscal(v) /( u.norme() * v.norme() )
from math import acos
alpha = acos(a)
print(alpha)
```

Exercice 10 :

Question 1 : ajout_monome = 5 pts

```
while (True) :
    try :
        degre = int(input("donner le degré du monôme: ")
        if   degre >= 0 :
            break
    except :
        print("erreur de saisie") ou pass ou continue
 while (True) :
    try :
        coeff = float(input("donner le coefficient : "))
        if   coeff >= 0 :
            break
    except :
        print('erreur de saisie') ou pass ou continue
self.data.update({degre : coeff})
```

Question 2 : degree = 2,5 pts

```
return max(list(self.data.keys())) #possible avec la programmation
```

Question 3 : __call__ = 2,5 pts

```
Val = 0
for degre in self.data.keys() :
    Val += self.data[degre] * x0 ** degre
return Val
```

Question 4 : __add__ = 5 pts

```
p = PolynomeCreux()
for d in self.data.keys():
    if not ( d in other.data.keys()):
        p.ajout_monome({d:self.data[d]})
    elif self.data[d]!=-other.data[d] :
        p.data.update({d : self.data[degre]+
other.data[degre]})
return p
```

```
    p = PolynomeCreux()
    for d1 in self.data.keys() :
        for d2 in other.data.keys() :
            d=d1+d2
            c=self.data[d1]*other.data[d2]
            if not(d in p.data.keys()) :
                p.ajout_monome({d:c})
            else :
                p.data[d]+=c
        if p.data[d]==0:
            del p.data[d]

    return p
```

Question 6 : __str__ = 7,5 pts

```
P=sorted(self.data.items(),reverse=True)
Ch=""
for  d,c in P :
    if c==1:
        Ch+=" + "
    elif c>0:
        Ch+=" + " + str(c)
    elif c==-1:
        Ch+=" - "
    elif c<0:
        Ch+=" - " + str(abs(c)
    if d!=0:
        if c!=1 and c!=-1:
            Ch+="*"
        Ch+="x"
        if d!=1:
            Ch+="**" + str(d)
if Ch[0:2]==" +":  #pour enlever l'espace et le premier
plus
return Ch[2:]
```

Question 7 : primitive = 5 pts

```
p=PolynomeCreux()
for d in self.data.keys() :
    p.ajout_monome({d+1:self.data[d]/(d+1)})
return p
```

Question 8 : integrale = 5 pts

```
def integarle(P,a,b):
    return P.primitive()(b) - P.primitive()(a)
```