Chapitre 1 : Programmation orientée objet

TP ~ Corrigé v1.1

Exercice 1 :

```
1.    class Point:
2.        def __init__(self,x=0,y=0):
3.            #constructeur paramétrable
4.            self.x = x
5.            self.y = y
6.        #méthode : fonction définie dans la classe
7.        def afficher(self):
8.            print("({},{})".format(self.x, self.y))
9.
10.   class Rectangle:
11.       def __init__(self,p,larg,haut):
12.           assert type(p)==Point
13.           self.coinSupG = p
14.           assert type(larg)==float and type(haut)==float
15.           self.largeur = larg
16.           self.hauteur = haut
17.
18.   #Q2
19.   p1 = Point()
20.   p1.afficher()
21.   #Q5
22.   p = Point(12.0,27.0)
23.
24.   #Q6
25.   def trouverCentre(r):
26.       assert isinstance(r,Rectangle)
27.       x = r.coinSupG.x + r.largeur/2
28.       y = r.coinSupG.y - r.hauteur/2
29.       return Point(x,y)
30.
31.   larg = 50.0
32.   haut = 35.0
33.   boite = Rectangle(p,larg,haut)
```

Exercice 2 :

```
1.    class Time:
2.        def __init__(self):
3.            self.heures = 0
4.            self.minutes = 0
5.            self.secondes = 0
6.
7.        def affiche_heure(self):
8.            assert isinstance(self,Time), "Erreur: ...."
9.            print(self.heures,":",self.minutes,":",self.secondes)
10.
11.   instant = Time()
```

```
12.
13.   def affiche_heure(t):
14.       assert isinstance(t,Time), "Erreur: ...."
15.       print(t.heures,":",t.minutes,":",t.secondes)
16.
17.
18.   affiche_heure(instant)#appel à une fonction (déclarée endéhors de classe)
19.
20.   maintenant = Time()
21.   maintenant.heures = int(input("heures:"))
22.   maintenant.minutes = int(input("minutes:"))
23.   maintenant.secondes = int(input("secondes:"))
24.   maintenant.affiche_heure() #appel à une méthode(déclarée dans la classe)
```

Exercice 3 :

```
1.    class CompteBancaire:
2.        def __init__(self,nom='salim',solde=1000):
3.            self.nom = nom
4.            self.solde = solde
5.
6.        def depot(self,somme):
7.            #somme : réel et >0
8.            self.solde += somme
9.
10.       def retrait(self,somme):
11.           #somme : réel et >0 10
12.           assert (type(somme)==int or type(somme)==float) and somme >=0
13.           if self.solde >= somme :
14.               self.solde -= somme
15.           else:
16.               print("Solde insuffisant")
17.
18.      def __repr__(self): #>>> c1 ==> __repr__(c1)
19.          return "Le solde du CB de {} est de {} Dinars"\
20.                 .format(self.nom,self.solde)
21.
22.   c1 = CompteBancaire(solde=2000)
```

Exercice 4 :

```
1.    class Voiture:
2.        def __init__(self,marque="Ford",couleur="Rouge"):
3.            self.pilote = "personne"
4.            self.vitesse= 0
5.            self.marque = marque
6.            self.couleur = couleur
7.
8.        def choix_conducteur(self):
9.            nom = input("nouveu conducteur:")
```

```
            self.pilote = nom

        def accelerer(self,taux,duree):
            if self.pilote != "personne":
                self.vitesse += taux * duree
            else:
                print("Cette voiture n'a pas de conducteur")

        def affiche_tout(self):
            print("{0} {1} pilotée par {2}, vitesse = {3} m/s"\
                  .format(self.marque, self.couleur, self.pilote,\
                          self.vitesse))

        def __str__(self):#print(v)=>print(str(v)) ou str(v)
            return "{0} {1} pilotée par {2}, vitesse = {3} m/s"\
                  .format(self.marque, self.couleur, self.pilote,\
                          self.vitesse)
```

Exercice 5 :

```
    def test(val, typeVal = str):
        if typeVal==str:
            return val.replace(' ','').isalpha() and len(val)>2
        else:
            return type(val)==typeVal

    class Personne:
        def __init__(self,nom,prenom, email, tel):
            assert test(nom,str), "nom invalide"
            self.nom = nom
            assert test(prenom,str), "prenom invalide"
            self.prenom = prenom

            self.email = email

            self.tel = tel

        def __str__(self):
            #str(p)
            #print(p)==>print(str(p))==> print(Personne.__str__(p))
            return "{}, {} -- Tel: {} -- Email:{}"\
                   .format(self.nom, self.prenom,\
                           self.tel, self.email)
    class Travailleur(Personne):
        def __init__(self,nom,prenom, email, tel,entr,telEntr,AdrEntr):
            Personne.__init__(self,nom,prenom, email, tel)
            self.entr = entr
            self.telEntr = telEntr
            self.AdrEntr = AdrEntr

        def __str__(self):
            ch = Personne.__str__(self)
            ch += ", Entr: {}, tel Entr: {}, adresse Entr: {}"\
```

```
                .format(self.entr,self.telEntr,self.AdrEntr)
        return ch

class Scientifique(Travailleur):
    def __init__(self,nom,prenom, email, tel,entr,telEntr,AdrEntr,ds,ts):
        Travailleur.__init__(self,nom,prenom, email, tel,entr,\
                             telEntr,AdrEntr)
        assert test(ds), "disicplince invalide"
        self.ds = ds
        assert test(ts,list), "type scientifique invalide"
        self.ts = ts

    def __str__(self):
        ch = Travailleur.__str__(self)
        ch += ", disicplince: {}, type scientifique: {}"\
              .format(self.ds,','.join(self.ts))
        return ch

try:
    s = Scientifique('Ben salem','amir','555555','amir@email.com',\
                     'Ipein','722222','Nabeul','chimie',['info','exper'])
    print(s)#print(str(s))
except AssertionError:
    print("paramètres invalides")


try:
    p = Personne('Ben salem1','amir','555555','amir@email.com')
    print(p)
except AssertionError:
    print("paramètres invalides")

try:
    t = Travailleur('Ben salem','amir','555555','amir@email.com',\
                    'Ipein','722222','Nabeul')
    print(t)#print(str(t))
except AssertionError:
    print("paramètres invalides")
```

Exercice 6 :

```
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):
        ch="{}, {} -- Téléphone: {} -- Email: {}"
        ch = ch.format(self.nom,self.prenom,self.tel,self.email)
        return ch

```

```python
#le carnet d'adresse contient une liste de personnes
class CarnetAdresse():
    def __init__(self):
        self.contacts = []

    def ajouter_contact(self,p):
        assert type(p)==Personne, "Erreur: On attend une personne"
        self.contacts.append(p)

    def chercher_contact(self,nom,prenom=None):
        for i in range(len(self.contacts)):
            p = self.contacts[i]
            if prenom==None:
                if p.nom == nom:
                    print(p)
            else:
                if p.nom == nom and p.prenom==prenom:
                    print(p)

    def __str__(self):
        ch = ""
        for p in self.contacts:
            ch += str(p) + "\n"
        return ch


c1  = CarnetAdresse()
amir = Personne('Ben Salem','Amir','333333','email@email.com')
c1.ajouter_contact(amir)
c1.ajouter_contact(amir)
c1.chercher_contact('Ben Salem','Amfir')
print(c1)
```

Exercice 7 :

```python
class IntervallError(Exception):
    pass

class Intervalle:
    def __init__(self,binf,bsup):
        if (type(binf)== int or type(binf)==float) and binf > 0:
            self.__binf = binf #attribut privé : modifié seulement dans la classe
        else:
            raise IntervallError("Erreur: Bornes invalides!")
        if (type(binf)== int or type(binf)==float) and bsup>binf :
            self.__bsup = bsup
        else:
            raise IntervallError("Erreur: Bornes invalides!")

    def getBinf(self): #getNomAttribut : getBinf()
        return self.__binf

    def setBinf(self,val):#setNomAttribut : setBinf(self,val)
        if (type(val)== int or type(val)==float):
            self.__binf = val
```

```
            else:
                raise IntervallError("Erreur: Bornes invalides!")


        def getBsup(self):
            return self.__bsup

        def setBsup(self,val):
            if (type(val)== int or type(val)==float):
                self.__bsup = val
            else:
                raise IntervallError("Erreur: Bornes invalides!")

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

        def __contains__(self,val):
            if (type(val)== int or type(val)==float):
                return  self.__binf <= val <= self.__bsup
            else:
                raise IntervallError("Erreur: valeur passée .. invalides!")

        def __add__(self,other):#i1 + i2 <=> self + other
            binf = self.__binf + other.__binf
            bsup = self.__bsup + other.__bsup
            return Intervalle(binf,bsup)




        def __sub__(self,other):#i1 - i2 <=> self - other
            binf = self.__binf - other.__bsup
            bsup = self.__bsup - other.__binf
            return Intervalle(binf,bsup)

        def __mul__(self,other):#i1 * i2 <==> self * other
            binf = self.__binf * other.__binf
            bsup = self.__bsup * other.__bsup
            return Intervalle(binf,bsup)


        def __and__(self,other):#i1 & i2 <==> self & other
            binf = max(self.__binf, other.__binf)
            bsup = min(self.__bsup , other.__bsup)
            return Intervalle(binf,bsup)

     #Remarque: assert cond==False <=> if cond==False  : raise IntervallError


    try:
        a = Intervalle(1,6)
        b = Intervalle(4,8)
        a.setBsup(5)
        print("a&b=",a&b)
        print("a+b=",a+b)
```

```
        print("a*b=",a*b)
        print("a&<9,11>",a&Intervalle(9,11))
        print("a-b=",a-b)
    except IntervallError:
        print("Erreur: Bornes invalides!")
```

Exercice 8 :

```
class Vect2d:
    def __init__(self,x=0,y=0):
        self.x = x
        self.y = y

    def add(self,other):#v1.add(v2)
        return Vect2d(self.x + other.x,self.y + other.y)

    def mul_ext(self,k):
        #tester de validité de k
        assert type(k)==float
        return Vect2d(self.x * k , self.y * k)

    def zoom(self,k):
        #tester de validité de k
        assert type(k)==float
        self.x *= k
        self.y *= k

    def prodscal(self,other):
        return Vect2d(self.x * other.x + self.y * other.y)

    def norme(self):
        from math import sqrt
        return sqrt(prodscal(self,self))

    def __add__(self,other):# v1 + v2
        return Vect2d(self.x + other.x, self.y + other.y)

    def __repr__(self): #>>>v1
        return "<{},{}>".format(self.x ,self.y)

    def __eq__(self,other): # v1 == v2
        return self.x == other.x and self.y == other.y

#prog principal
u = Vect2d(3,-2)
v = Vect2d(4,1)
w = u + v
u1 = add(u,w).prodscal(v)
u2 = u.prodscal(v) + w.prodscal(v)
print(u1 == u2)
#remarque :
#sans la méthode __eq__, le test u1==u2 compare si id(u1)==id(u2)
print(u.mul_ext(5.).prodscal(v) == u.prodscal(v)*5)
```

Exercice 9 :

```
1.    class Vol_direct():
2.        def __init__(self, dep, arr, jour, heure):
3.            self.dep = dep
4.            self.arr = arr
5.            assert jour in ['Lundi','Mardi','Mercredi','Jeudi',\
6.                            'Vendredi','Samedi','Dimanche']
7.            self.jour = jour
8.            assert type(heure) == int and 0<=heure<=23
9.            self.heure = heure
10.
11.       def affiche(self):
12.           print("Ce vol part de '{}' vers '{}' le '{}' à '{}' heure"\
13.                 .format(self.dep,self.arr,self.jour,self.heure))
14.
15.   class Vols:
16.       def __init__(self, vols=[]):
17.           for v in vols:
18.               assert isinstance(v,Vol_direct)
19.           self.vols = vols
20.
21.       def liste_successeurs(self,ville):
22.           l = []
23.           for vol in self.vols:
24.               if vol.dep == ville:
25.                   l.append(vol.arr)
26.           return l
27.
28.       def appartient(self,ville):
29.           for vol in self.vols:
30.               if ville in [vol.dep,vol.arr]:
31.                   return True
32.           return False
33.
34.       def affiche(self):
35.           for vol in self.vols:
36.               vol.affiche()
37.
38.   #Q3
39.   #a
40.   import random
41.   LV = [Vol_direct(saisie_ville(), saisie_ville(), saisie_jour(),
42.                    saisie_heure())for i in range(random.randint(1,4)]
43.
44.   v = Vols(LV)
45.   v.affiche()
46.
47.   while 1:
48.         ville = saisie_ville()
49.         if v.appartient(ville): break
50.
51.   l = v.liste_successeurs(ville)
52.   print(len(l), l)
```

Exercice 10 :

```
1. class PolynomeCreux:
2.     def __init__(self):
3.         self.data = {}
4.
5.     def ajout_monome(self,monome={}):
6.         if len(monome)==0:
7.             while 1:
8.                 try:
9.                     coef = float(input("coef="))
10.                    deg = int(input("deg="))
11.                    if deg >=0 and coef != 0:
12.                        break
13.                except:
14.                    continue
15.            if deg in self.data.keys() and self.data[deg] != -coef:
16.                self.data.update({deg:coef})
17.
18.        def degree(self):
19.            return max(self.data.keys())
20.
21.        def __call__(self,x0):
22.            assert type(x0)==float
23.            s = 0
24.            for monome in self.data.items():
25.                deg = monome[0]
26.                coef = monome[1]
27.                s += coef * (x0**deg)
28.            return s
29.
30.        def __add__(self,other):
31.            assert isinstance(other,PolynomeCreux)
32.            p = PolynomeCreux()
33.            p.data = self.data.copy()
34.            for monome in other.data.items():
35.                if monome[0] in p.data.keys():
36.                    if p.data[monome[0]]+monome[1] != 0:
37.                        p.data.update({monome[0]:p.data[monome[0]]+monome[1]})
38.                    else:
39.                        del p.data[monome[0]]
40.                else:
41.                    p.data.update({monome[0]:monome[1]})
42.            return p
43.
44.        def __mul__(self,other):
45.            assert isinstance(other,PolynomeCreux)
46.             p = PolynomeCreux()
47.             m = self.degree() + other.degre()
48.             for deg in range(m+1):
49.                 s=0
50.                 for i in self.data.keys():
51.                     for j in other.data.keys():
52.                         if deg == i+j:
53.                             s+= self.data[i] * other.data[j]
```

```
54.                 if s!=0:
55.                     p.ajout_monome({deg,s})
56.
57.             return p
58.
59.         def primitive(self):
60.             p = PolynomeCreux()
61.             for item in self.data.items():
62.                 p.ajout_monome({item[0]+1,item[1]/(item[0]+1)})
63.             return p
64.
65.         def integrale(self,a,b):
66.             assert type(a)==float and type(b)==float
67.             return self.primitive()(b)-self.primitive()(a)
```