

Chapitre 1 : Programmation orientée objet

TP ~ Corrigé 1.0

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 endehors 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("nouveau conducteur:")
10.        self.pilote = nom
11.

```

```
12.     def accelerer(self,taux,duree):
13.         if self.pilote != "personne":
14.             self.vitesse += taux * duree
15.         else:
16.             print("Cette voiture n'a pas de conducteur")
17.
18.     def affiche_tout(self):
19.         print("{0} {1} pilotée par {2}, vitesse = {3} m/s"\
20.             .format(self.marque, self.couleur, self.pilote,\
21.                 self.vitesse))
22.
23.     def __str__(self):#print(v)=>print(str(v)) ou str(v)
24.         return "{0} {1} pilotée par {2}, vitesse = {3} m/s"\
25.             .format(self.marque, self.couleur, self.pilote,\
26.                 self.vitesse)
```

Exercice 5 :

```
1.     def test(val, typeVal = str):
2.         if typeVal==str:
3.             return val.replace(' ','').isalpha() and len(val)>2
4.         else:
5.             return type(val)==typeVal
6.
7.     class Personne:
8.         def __init__(self,nom,prenom, email, tel):
9.             assert test(nom,str), "nom invalide"
10.            self.nom = nom
11.            assert test(prenom,str), "prenom invalide"
12.            self.prenom = prenom
13.
14.            self.email = email
15.
16.            self.tel = tel
17.
18.         def __str__(self):
19.             #str(p)
20.             #print(p)==>print(str(p))=> print(Personne.__str__(p))
21.             return "{}, {} -- Tel: {} -- Email:{}"\
22.                 .format(self.nom, self.prenom,\
23.                     self.tel, self.email)
24.     class Travailleur(Personne):
25.         def __init__(self,nom,prenom, email, tel,entr,telEntr,AdrEntr):
26.             Personne.__init__(self,nom,prenom, email, tel)
27.             self.entr = entr
28.             self.telEntr = telEntr
29.             self.AdrEntr = AdrEntr
30.
31.         def __str__(self):
32.             ch = Personne.__str__(self)
33.             ch += ", Entr: {}, tel Entr: {}, adresse Entr: {}"\
34.                 .format(self.entr,self.telEntr,self.AdrEntr)
35.             return ch
36.
```

```
37. class Scientifique(Travailleur):
38.     def __init__(self,nom,prenom, email, tel,entr,telEntr,AdrEntr,ds,ts):
39.         Travailleur.__init__(self,nom,prenom, email, tel,entr,\
40.                               telEntr,AdrEntr)
41.         assert test(ds), "disicplince invalide"
42.         self.ds = ds
43.         assert test(ts,list), "type scientifique invalide"
44.         self.ts = ts
45.
46.     def __str__(self):
47.         ch = Travailleur.__str__(self)
48.         ch += ", disicplince: {}, type scientifique: {}"\
49.             .format(self.ds,','.join(self.ts))
50.         return ch
51.
52. try:
53.     s = Scientifique('Ben salem','amir','555555','amir@email.com',\
54.                     'Ipein','722222','Nabeul','chimie',['info','exper'])
55.     print(s)#print(str(s))
56. except AssertionError:
57.     print("paramètres invalides")
58.
59.
60. try:
61.     p = Personne('Ben salem1','amir','555555','amir@email.com')
62.     print(p)
63. except AssertionError:
64.     print("paramètres invalides")
65.
66. try:
67.     t = Travailleur('Ben salem','amir','555555','amir@email.com',\
68.                   'Ipein','722222','Nabeul')
69.     print(t)#print(str(t))
70. except AssertionError:
71.     print("paramètres invalides")
```

Exercice 6 :

```
1. class Personne(object):
2.     def __init__(self,nom,prenom,tel,email):
3.         self.nom = nom
4.         self.prenom = prenom
5.         self.tel = tel
6.         self.email = email
7.
8.     def __str__(self):
9.         ch="{}, {} -- Téléphone: {} -- Email: {}"
10.        ch = ch.format(self.nom,self.prenom,self.tel,self.email)
11.        return ch
12.
13. #le carnet d'adresse contient une liste de personnes
14. class CarnetAdresse():
15.     def __init__(self):
16.         self.contacts = []
```

```
17.
18.     def ajouter_contact(self,p):
19.         assert type(p)==Personne, "Erreur: On attend une personne"
20.         self.contacts.append(p)
21.
22.     def chercher_contact(self,nom,prenom=None):
23.         for i in range(len(self.contacts)):
24.             p = self.contacts[i]
25.             if prenom==None:
26.                 if p.nom == nom:
27.                     print(p)
28.             else:
29.                 if p.nom == nom and p.prenom==prenom:
30.                     print(p)
31.
32.     def __str__(self):
33.         ch = ""
34.         for p in self.contacts:
35.             ch += str(p) + "\n"
36.         return ch
37.
38.
39. c1 = CarnetAdresse()
40. amir = Personne('Ben Salem','Amir','333333','email@email.com')
41. c1.ajouter_contact(amir)
42. c1.ajouter_contact(amir)
43. c1.chercher_contact('Ben Salem','Amfir')
44. print(c1)
```

Exercice 7 :

```
1.     class IntervallError(Exception):
2.         pass
3.
4.     class Intervalle:
5.         def __init__(self,binf,bsup):
6.             if (type(binf)== int or type(binf)==float) and binf > 0:
7.                 self.__binf = binf #attribut privé : modifié seulement dans la classe
8.             else:
9.                 raise IntervallError("Erreur: Bornes invalides!")
10.            if (type(binf)== int or type(binf)==float) and bsup>binf :
11.                self.__bsup = bsup
12.            else:
13.                raise IntervallError("Erreur: Bornes invalides!")
14.
15.        def getBinf(self): #getNomAttribut : getBinf()
16.            return self.__binf
17.
18.        def setBinf(self,val):#setNomAttribut : setBinf(self,val)
19.            if (type(val)== int or type(val)==float):
20.                self.__binf = val
21.            else:
22.                raise IntervallError("Erreur: Bornes invalides!")
23.
24.
25.        def getBsup(self):
```

```
26.         return self.__bsup
27.
28.     def setBsup(self,val):
29.         if (type(val)== int or type(val)==float):
30.             self.__bsup = val
31.         else:
32.             raise IntervallError("Erreur: Bornes invalides!")
33.
34.     def __str__(self):
35.         return "<{},{}>".format(self.__binf,self.__bsup)
36.
37.     def __contains__(self,val):
38.         if (type(val)== int or type(val)==float):
39.             return self.__binf <= val <= self.__bsup
40.         else:
41.             raise IntervallError("Erreur: valeur passée .. invalides!")
42.
43.     def __add__(self,other):#i1 + i2 <=> self + other
44.         binf = self.__binf + other.__binf
45.         bsup = self.__bsup + other.__bsup
46.         return Intervalle(binf,bsup)
47.
48.     def __sub__(self,other):#i1 - i2 <=> self - other
49.         binf = self.__binf - other.__bsup
50.         bsup = self.__bsup - other.__binf
51.         return Intervalle(binf,bsup)
52.
53.     def __mul__(self,other):#i1 * i2 <==> self * other
54.         binf = self.__binf * other.__binf
55.         bsup = self.__bsup * other.__bsup
56.         return Intervalle(binf,bsup)
57.
58.
59.     def __and__(self,other):#i1 & i2 <==> self & other
60.         binf = max(self.__binf, other.__binf)
61.         bsup = min(self.__bsup , other.__bsup)
62.         return Intervalle(binf,bsup)
63.
64.     #Remarque: assert cond==False <=> if cond==False : raise IntervallError
65.
66.
67.     try:
68.         a = Intervalle(1,6)
69.         b = Intervalle(4,8)
70.         a.setBsup(5)
71.         print("a&b=",a&b)
72.         print("a+b=",a+b)
73.         print("a*b=",a*b)
74.         print("a<9,11>",a&Intervalle(9,11))
75.         print("a-b=",a-b)
76.     except IntervallError:
77.         print("Erreur: Bornes invalides!")
```

Exercice 8 :

```
1. class Vect2d:
2.     def __init__(self,x=0,y=0):
3.         self.x = x
4.         self.y = y
5.
6.     def add(self,other):#v1.add(v2)
7.         return Vect2d(self.x + other.x,self.y + other.y)
8.
9.     def mul_ext(self,k):
10.        #tester de validité de k
11.        assert type(k)==float
12.        return Vect2d(self.x * k , self.y * k)
13.
14.    def zoom(self,k):
15.        #tester de validité de k
16.        assert type(k)==float
17.        self.x *= k
18.        self.y *= k
19.
20.    def prodscal(self,other):
21.        return Vect2d(self.x * other.x + self.y * other.y)
22.
23.    def norme(self):
24.        from math import sqrt
25.        return sqrt(prodscal(self,self))
26.
27.    def __add__(self,other):# v1 + v2
28.        return Vect2d(self.x + other.x, self.y + other.y)
29.
30.    def __repr__(self): #>>>v1
31.        return "<{},{}>".format(self.x ,self.y)
32.
33.    def __eq__(self,other): # v1 == v2
34.        return self.x == other.x and self.y == other.y
35.
36. #prog principal
37. u = Vect2d(3,-2)
38. v = Vect2d(4,1)
39. w = u + v
40. u1 = add(u,w).prodscal(v)
41. u2 = u.prodscal(v) + w.prodscal(v)
42. print(u1 == u2)
43. #remarque :
44. #sans la méthode __eq__, le test u1==u2 compare si id(u1)==id(u2)
45. print(u.mul_ext(5.).prodscal(v) == u.prodscal(v)*5)
```