Exercices d'informatique - Série n° 1

Cours 4INOC01 Série distribuée le 31.8.2017

1. L'instruction *class*.

Dans le script qui suit, on définit une classe Point et on crée un objet de ce type.

```
1  class Point:
2      "Point mathematique"
3  p=Point()
4  print "documentation de la classe Point: ",p.__doc__,"\n"
5  print "Information concernant l'objet p: ", p,"\n"
```

(1) Que contient l'attribut __doc__ de l'objet p ?
(2) Déterminer l'effet de la ligne 5.
(3) Ecrire un script qui permette d'afficher la documentation d'un objet de la classe Tk du module *Tkinter*.

2. L'instruction *class*.

Les fonctions peuvent utiliser des objets comme paramètres (elles peuvent également fournir un objet comme valeur de retour). Créer une fonction qui affiche la chaîne de documentation d'un objet.

3. Attributs.

On peut définir des variables associées à un objet comme dans le script suivant:

```
1  class Point:
2      "Point mathematique"
3  p=Point()
4  p.couleur='vert'
5  p.rayon=1.5
6  p.x=3
7  p.y=4
8  print p.couleur
9  couleur=p.couleur
10 p.couleur='rouge'
11 print p.couleur, couleur
```

La variable couleur est-elle la même que l'attribut couleur de l'objet p ?

4. Méthodes.

Une fonction placée à l'intérieur de la définition d'une classe est appelée une *méthode* de la classe. Le premier paramètre utilisé par une méthode doit toujours être une référence d'instance. Il est d'usage d'appeler ce paramètre self. L'appel à une méthode se fait comme dans l'exemple donné ci-dessous:

```
1  class Point:
2      "Point mathematique"
3      def abscisse(self):
4          print self.x
5  p=Point()
6  p.x=3
7  p.abscisse()
```

(1) Que se passe-t-il si on enlève la ligne 6 ?
(2) Ajouter à la classe Point une méthode Distance qui renvoie la distance d'un point à l'origine.

5. La méthode *init*.

Déterminer l'action de la méthode `__init__` en exécutant le script suivant:

```
1  class Point:
2      "Point mathematique"
3      def __init__(self):
4          self.x=12
5          self.y=15
6  p=Point()
7  print p.x,p.y
```

6. La fonction *init*.

Déterminer le rôle des paramètres après le paramètre `self` de la méthode `__init__` en exécutant le script suivant:

```
1  class Point:
2      "Point mathematiques"
3      def __init__(self,albert=0,bernard=0):
4          self.x=albert
5          self.y=bernard
6  p=Point(12,33)
7  print p.x, p.y
```

7. Similitude et unicité.

Deux objets d'une même classe sont-ils distincts ? Pour répondre à cette question, exécuter le script suivant:

```
1  class Point:
2      "Point mathematique"
3  p1=Point()
4  p2=Point()
5  print p1==p2
6  print p1
7  print p2
8  #
9  p2.couleur='vert'
10 p2=p1
11 print p1==p2
12 p1.couleur='rouge'
13 print p2.couleur
14 print p1
15 print p2
```

8. Points et rectangles.

Définir deux classes `Point` et `Rectangle` où chaque point est donné par ses coordonnées `x` et `y` et chaque rectangle par sa `largeur` , sa `hauteur` et son coin inférieur gauche qui doit être un objet de la classe `Point`. La classe `Rectangle` doit contenir une méthode `centre` renvoyant un objet de la classe `Point` donnant le centre du rectangle. Sauvegarder ces deux classes dans un fichier nommé "geometrie.py". Il faut que le script suivant fonctionne correctement.

```
1  from geometrie import *
2  boite=Rectangle()
3  print "Le centre du rectangle est: ",boite.centre().x,boite.centre().y
4  boite.largeur=15
5  boite.hauteur=10
6  alfred=Point(10,12)
7  boite.coin=alfred
8  print "Le centre du rectangle est: ",boite.centre().x,boite.centre().y
9  boite.largeur=boite.largeur+7
10 boite.hauteur=boite.hauteur-2
```

```
11 print "Le centre du rectangle est: ",boite.centre().x,boite.centre().y
```

9. Espaces de noms.

Exécuter le script suivant et expliquer le résultat.

```
1  class Espaces:
2      aa=33
3      def affiche(self):
4          print aa, Espaces.aa, self.aa
5  aa=12
6  essai=Espaces()
7  essai.aa=67
8  essai.affiche()
9  print aa, Espaces.aa, essai.aa
```

10. Héritage.

Exécuter le script suivant:

```
1  class Mammifere:
2      caract1="il allaite ses petits ;"
3  class Carnivore(Mammifere):
4      caract2="il se nourrit de la chair de ses proies ;"
5  class Chien(Carnivore):
6      caract3="son cri s'appelle aboiement ;"
7  mirza=Chien()
8  print mirza.caract1, mirza.caract2, mirza.caract3
9  #
10 mirza.caract2="son corps est couvert de poils"
11 print mirza.caract2
12 fido=Chien()
13 print fido.caract2
```

(1) L'objet mirza hérite-t-il des attributs de ses classes parentes ?
(2) L'instance mirza peut-elle modifier les attributs de ses classes parentes ?

11. Héritage et polymorphisme.

Exécuter le script suivant et expliquer le résultat.

```
1  class Atome:
2      table =[None, ('hydrogene',0), ('helium',2), ('lithium',4),
3              ('beryllium',5), ('bore',6), ('carbone',6), ('azote',7),
4              ('oxygene',8), ('fluor',10), ('neon',10)]
5      def __init__(self, nat):
6          self.np=nat
7          self.ne = nat
8          self.nn = Atome.table[nat][1]
9      def affiche(self):
10         print
11         print "Nom de l'element: ", Atome.table[self.np][0]
12         print self.np," protons,",self.ne," electrons,",self.nn," neutrons"
13 class Ion(Atome):
14     def __init__(self, nat, charge):
15         Atome.__init__(self, nat)
16         self.ne = self.ne - charge
17         self.charge = charge
18     def affiche(self):
19         Atome.affiche(self)
20         print "Particule electrisee. Charge =", self.charge
21 a1 = Atome(5)
22 a2 = Ion(3, 1)
23 a3 = Ion(8, -2)
24 a1.affiche()
```

```
a2.affiche()
a3.affiche()
```

12. Classes.

Exécuter le script donné ci-dessous. Un objet peut-il avoir une fonction comme attribut ?

```
class Analyse:
    def __init__(self,f=lambda x:x):
        self.f=f
    def image(self,x):
        return self.f(x)
a=Analyse()
print a.image(3)
a.f=lambda x: x**3
print a.image(2)
#
def g(x):
    return x**2
b=Analyse(g)
print b.image(3)
```

13. Cercles et cylindres.

Définir une classe Cercle(). Les objets construits à partir de cette classe sont des cercles de tailles variables. En plus de la méthode constructeur (qui utilise donc un paramètre rayon), définir une méthode surface(), qui doit renvoyer la surface du cercle.

Définir ensuite une classe Cylindre() dérivée de la précédente. Le constructeur de cette nouvelle classe doit comporter les deux paramètres rayon et hauteur. Ajouter une méthode volume() qui doit renvoyer le volume du cylindre. La classe Cylindre peut, par exemple, s'utiliser comme suit:

```
>>> cyl=Cylindre(5,7)
>>> print cyl.surface()
78.5398163375
>>> print cyl.volume()
549.778714362
```

14. Graphiques.

Construire une classe Graphiques dont les objets ont trois attributs f, a et b. Le premier est une fonction et les autres sont des nombres. En plus d'une méthode constructeur permettant de donner des valeurs par défaut aux attributs, la classe Graphiques est munie d'une méthode qui affiche un graphique rudimentaire (sans échelles sur les axes) de la fonction f. Le nombre 1 correspond à 20 pixels. L'exécution du script

```
from math import *
lis=Graphiques(lambda x:cos(x),-2*pi,2*pi)
lis.dessin()
```

donne le dessin de la figure 1.

Indications: L'exécution du script

```
import Tkinter
cadre=Tkinter.Tk()
cadre.geometry("610x410")
a=Tkinter.Canvas(width=600,height=400,bg='light blue')
p0=[0,0,600,400]
a.create_line(p0,width=1,fill='red')
a.create_line(0,200,600,200,arrow="last")
a.create_line(300,400,300,0,arrow="last")
a.place(x=5,y=5)
```

```
10 cadre.mainloop()
```

donne le dessin de la figure 2.

15. Ordre lexicographique.
Définir une classe Mots munie d'une méthode permettant de comparer les mots avec l'ordre lexicographique comme sur l'illustration de la figure 3. Les mots doivent contenir seulement des lettres majuscules ou minuscules. L'ordre sur les lettres est donné par: ''<'A'<'B'< $\cdots$ <'Z' <'a' <'b'< $\cdots$ <'z'. La commande a.apres(c) donne 1 si a.mot est plus grand que c, 0 si a.mot=c et -1 sinon.

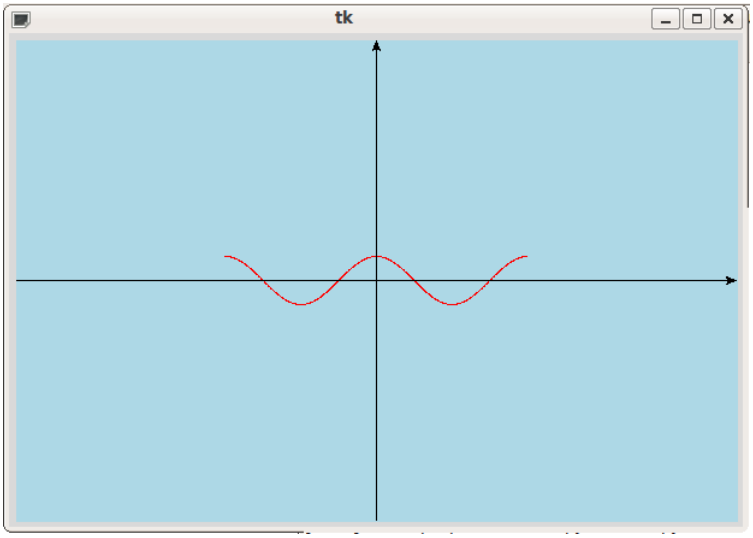

FIGURE 1. Exercice 14

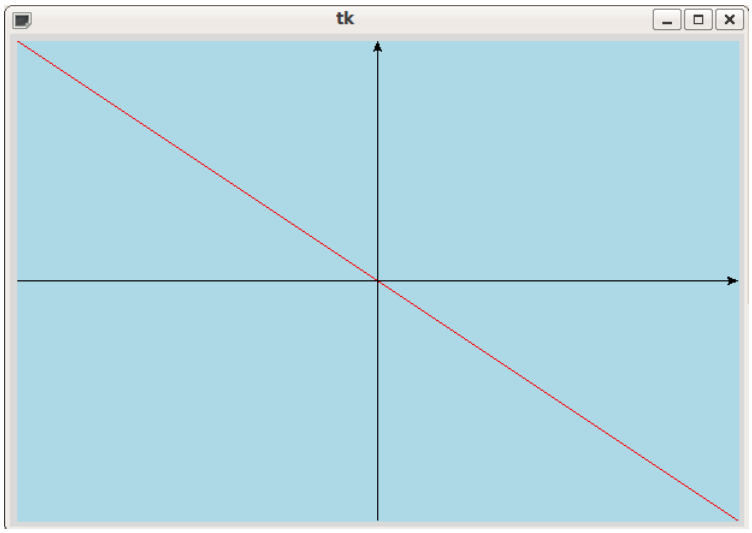

FIGURE 2. Exercice 14

```
Python 2.6.2 (release26-maint, Apr 19 2009, 01:56:41)
[GCC 4.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from Mots import *
>>> a=Mots()
>>> a.mot
''
>>> a.mot='zebre'
>>> a.apres('matin')
1
>>> a.apres('zebre')
0
>>> a.mot='matin'
>>> a.apres('zebre')
-1
>>> a.mot=''
>>> a.apres('A')
-1
>>> a.apres('')
0
>>> a.mot='Z'
>>> a.apres('a')
-1
>>> a.apres(a.mot)
0
```

FIGURE 3. Exercice 15