Exercices d'informatique - Corrigé de la série n° 1

Cours 4INOC01

1. L'instruction *class*.

(1) L'attribut `__doc__` contient la chaîne de caractères placée à la ligne 2.

(2) Le message renvoyé par *Python* indique que `p` est un instance de la classe `Point` (*i.e.* un objet de ce type) qui est définie elle-même au niveau principal du programme. Elle est située dans un emplacement bien déterminé de la mémoire vive, dont l'adresse apparaît ici en notation hexadécimale .

(3) On peut, par exemple, utiliser le script suivant:

```
1 import Tkinter
2 t=Tkinter.Tk()
3 print "documentation de la classe Tkinter: ",t.__doc__,"\n"
```

2. L'instruction *class*.

On peut, par exemple, utiliser la script suivant:

```
1 class Point:
2     "Point mathematique"
3 def infoobjet(objet):
4     print "Documentation: ",objet.__doc__
5 p=Point()
6 infoobjet(p)
```

3. Attributs.

Il n'y a pas de conflit entre la variable `couleur` et l'attribut `couleur` de l'objet `p`. L'objet `p` contient en effet son propre espace de noms, indépendant de l'espace de nom principal où se trouve la variable `couleur`.

4. Méthodes.

(1) Une erreur se produit. C'est normal, car l'attribut `x` de l'objet `p` n'est pas défini. Il n'est pas recommandable de créer ainsi les attributs d'instance en dehors de l'objet lui-même, ce qui conduit (entre autres désagréments) à des erreurs comme celle que nous venons de rencontrer.

(2) On peut, par exemple, utiliser la script suivant:

```
1 class Point:
2     "Point mathematique"
3     def dist(self):
4         return (self.x**2+self.y**2)**0.5
5 p=Point()
6 p.x=3
7 p.y=4
8 print p.dist()
```

5. La méthode *init*.

La fonction `__init__` est une méthode (appelée " constructeur") qui est exécutée automatiquement lorsque l'on instancie un nouvel objet à partir de la classe. On peut y placer tout ce qui semble nécessaire pour initialiser automatiquement l'objet que l'on crée.

6. La fonction *init*.

La méthode __init__ comporte à présent 3 paramètres, avec pour chacun une valeur par défaut. Pour lui transmettre les arguments qui correspondent, il suffit de placer ceux-ci dans les parenthèses qui accompagnent le nom de la classe, lorsque l'on écrit l'instruction d'instanciation du nouvel objet.

7. Similitude et unicité.

Les deux objets p1 et p2 sont distincts, même s'ils ont des contenus similaires. La ligne 5 teste l'égalité de ces deux objets et le résultat est *False*. L'information affichée par les lignes 6 et 7 confirme cela: les deux variables p1 et p2 référencent bien des objets différents. Par l'instruction p2 = p1, nous assignons le contenu de p1 à p2. Cela signifie que désormais ces deux variables référencent le même objet. Les variables p1 et p2 sont des alias l'une de l'autre. Le test d'égalité de la ligne 11 renvoie cette fois la valeur *True*. p1 et p2 désignent bien toutes deux un seul et unique objet.

8. Points et rectangles.

On peut, par exemple, utiliser la script suivant:

```
1  class Point:
2      "Point mathematique"
3      def __init__(self,lucien=0,antoine=0):
4          self.x=lucien
5          self.y=antoine
6  class Rectangle:
7      "Rectangle"
8      def __init__(self,marcel=0,felix=0,corner=Point()):
9          self.largeur=marcel
10         self.hauteur=felix
11         self.coin=corner
12     def centre(self):
13         p=Point()
14         p.x=self.coin.x+self.largeur/2.0
15         p.y=self.coin.y+self.hauteur/2.0
16         return p
```

9. Espaces de noms.

Dans cet exemple, le même nom aa est utilisé pour définir trois variables différentes : une dans l'espace de noms de la classe (à la ligne 2), une autre dans l'espace de noms principal (à la ligne 5), et enfin une dernière dans l'espace de nom de l'instance (à la ligne 7).

La ligne 4 et la ligne 9 montrent comment vous pouvez accéder à ces trois espaces de noms (de l'intérieur d'une classe, ou au niveau principal), en utilisant la qualification par points. Notez encore l'utilisation de self pour désigner l'instance.

10. Héritage.

(1) Oui.
(2) Non.

Cet exemple montre comment il faut procéder pour dériver une classe à partir d'une classe parente: on utilise l'instruction **class**, suivie comme d'habitude du nom que l'on veut attribuer à la nouvelle classe, et on place entre parenthèses le nom de la classe parente.

11. Héritage et polymorphisme.

La définition de la classe Atome() commence par l'assignation de la variable table. C'est un attribut de classe. Le constructeur __init__() sert à générer trois attributs d'instance, le nombre de protons, le nombre d'électrons et le nombre de neutrons. La méthode affiche() affiche le nom de l'élément

ainsi que les nombres de protons, d'électrons et de neutrons. La classe Ion() est dérivée de sa classe parente Atome(). Les méthodes de cette classe font appel aux méthodes de la classe Atome().

12. Classes. Oui.

13. Cercles et cylindres.
On peut, par exemple, utiliser le script suivant:

```
class Cercle:
    def __init__(self,r=0):
        self.rayon=r
    def surface(self):
        return self.rayon**2*3.1415926535;
class Cylindre(Cercle):
    def __init__(self,r=0,h=0):
        self.rayon=r
        self.hauteur=h
    def volume(self):
        return self.surface()*self.hauteur
cyl=Cylindre(5,7)
print cyl.surface()
print cyl.volume()
```

14. Graphiques.
On peut, par exemple, utiliser le script suivant:

```
class Graphiques:
    def __init__(self,f=lambda x:x,a=0,b=1):
        self.f=f
        self.a=a
        self.b=b
    def dessin(self):
        import Tkinter
        cadre=Tkinter.Tk()
        cadre.geometry("610x410")
        a=Tkinter.Canvas(width=600,height=400,bg='light blue')
        p0=[]
        for n in xrange(0,600+1,1):
            x=float((self.b-self.a))/600*n+self.a
            p0.append(x*20+300)
            p0.append(-self.f(x)*20+200)
        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)
        cadre.mainloop()
from math import *
lis=Graphiques(lambda x:cos(x),-2*pi,2*pi)
lis.dessin()
```

15. Ordre lexicographique.
On peut, par exemple, utiliser le script suivant:

```
class Mots:
    "mots avec ordre lexicographique"
    def __init__(self,mot=''):
        self.mot=mot
    def apres(self,mot):
        x=self.mot
        l1=len(x)
        l2=len(mot)
        l=max(l1,l2)
```

```
10          if l==0:
11              return 0
12          else:
13              if l1-l2>0:
14                  mot=mot+(l1-l2)*'A'
15              elif l2-l1>0:
16                  x=x+(l2-l1)*'A'
17              j=0
18              d=0
19              while d==0 and j<l:
20                  d=ord(x[j])-ord(mot[j])
21                  j=j+1
22              if d!=0:
23                  return int(d/abs(d))
24              else:
25                  return 0
```

ou plus simplement, comme l'ordre lexicographique existe en Python, le script:

```
1   class Mots:
2       "mots avec ordre lexicographique"
3       def __init__(self,mot=''):
4           self.mot=mot
5       def apres(self,mot):
6           if self.mot>mot:
7               return 1
8           elif self.mot==mot:
9               return 0
10          else:
11              return -1
```