netmind

WeKnowIT

Roche

Python

PROGRAMACIÓN ORIENTADA A OBJETOS

1. Definiendo una clase
2. Métodos y atributos de instancia
3. Métodos y atributos de clase
4. Herencia
5. Módulos

1 Definiendo una clase

Definiendo una clase

- Para poder realizar una programación orientada a objetos necesitaremos gestionar clases, métodos, propiedades y objetos
- Para definir una clase

```
class ClassName:
   'Optional class documentation string'
   #class_suite
```

- Creando un objeto

```
obj= ClassName()
```

Definiendo una clase

```
class Antena():
    color = ""
    longitud = ""

class Pelo():
    color = ""
    textura = ""

class Ojo():
    forma = ""
    color = ""
    tamanio = ""
```

Modela un usuario

- Crea una clase que modele un usuario

2 Métodos y atributos de instancia

Métodos y atributos de instancia

- Los atributos de una clase definen las propiedades de la misma
- Los métodos definen el comportamiento que puede mostrar una clase
- Los métodos y atributos de instancia son aquellos que se definen en la clase pero pueden cambiar de instancia a instancia

Métodos y atributos de instancia

```
class Employee:
   'Common base class for all employees'
   #esta variable es estática y se comparte entre instancias de Employee
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
      print ("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary)
```

Métodos y atributos de instancia

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)

emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)

emp1.age = 7  # Add an 'age' attribute.
emp1.age = 8  # Modify 'age' attribute.
del emp1.salary  # Delete 'salary' attribute.
```

Métodos y atributos de instancia

- En este caso son todos los métodos que están descritos en la clase y que no están anotados con @classmethod
  - Esto haría que fuera un método de la clase y no de la instancia
- Por lo demás los métodos son funciones normales, salvo que pueden usar la palabra reservada **self** (como el this) para acceder los métodos o atributos internos

Métodos y atributos de instancia

- El constructor se llamará __init o __init__ y se le podrán poner parámetros predefinidos

```
class Song(object):
    def __init__(self, lyrics):
        self.lyrics = lyrics

    def sing_me_a_song(self):
        for line in self.lyrics:
            print(line)
```

```
happy_bday = Song(["Happy birthday to you",
                   "I don't want to get sued",
                   "So I'll stop right there"])

bulls_on_parade = Song(["They rally around the family",
                        "With pockets full of shells"])

happy_bday.sing_me_a_song()

bulls_on_parade.sing_me_a_song()
```

3 Métodos y atributos de clase

Métodos y atributos de clase

- En el caso de los métodos de clase deberían encabezarse con ella anotación **@classmethod**

```
@classmethod
def introduce(cls):
        print("Hello, I am %s!" % cls)
```

Métodos y atributos de clase

- En el caso de los atributos de clase valdría con colocar el atributo fuera de la definición de cualquier método

```
class Employee:
    """Esta variable es estática y se comparte entre instancias de Employee"""
    empCount = 0
```

- Estos atributos de clase se mantienen entre ejecuciones de objetos de la clase

Jugando con objetos

- Queremos definir un programa que calcule el coche más rápido
- Para ello usaremos objetos que modelen los coches y motores
  - Coche
    - marca: String
    - motor: Motor
    - diametro_ruedas: int
    - posicion=0: int
    - avanza(): float # en función del tipo de motor avanza x metros/minuto
      - Cálculo simple: circ_neumáticos*pi*rpm
  - Motor
    - tipo: String
    - rpm: int

- Genera dos instancias de coche
- Simula el coche más rápido en recorrer 100km

4 Herencia

Herencia

- Para Heredar de una clase padre, simplemente basta con indicar la clase como atributo de clase que se está definiendo
  - class Child(Parent):
- En Python disponemos de la posibilidad de tener herencia múltiple
  - Pero no es recomendable usar herencia múltiple
- Desde la clase hija podemos acceder a la clase padre con el método **super()**

Herencia

```
class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print ("Calling parent constructor")
   def parentMethod(self):
      print ('Calling parent method')
   def setAttr(self, attr):
      Parent.parentAttr = attr
   def getAttr(self):
      print ("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class
   def __init__(self):
      print ("Calling child constructor")
   def childMethod(self):
      print ('Calling child method')
```

Herencia

```
c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

Herencia múltiple

- Se definde indicando las clases padres como parámetros para la definición de clases

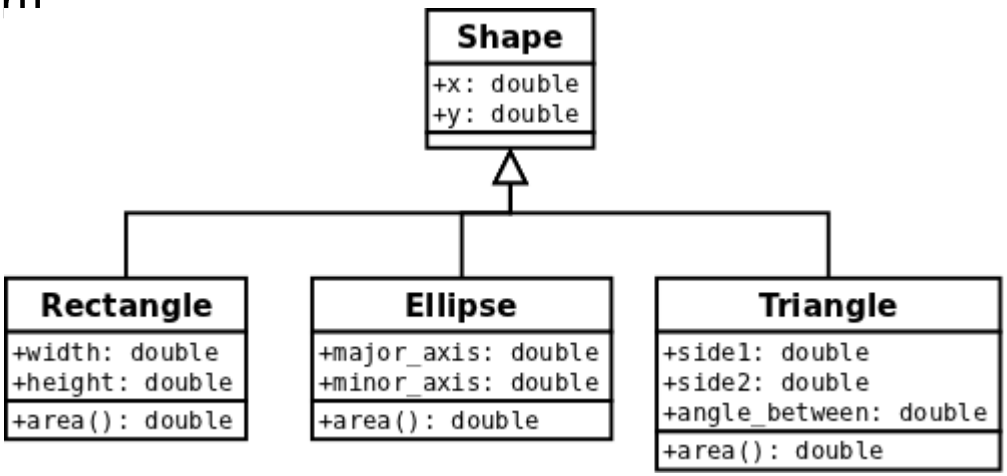
```
class A:        # define your class A
    def __init__(self):
        self

class B:        # define your class B
    def __init__(self):
        self


class C(A, B):   # subclass of A and B
    def __init__(self):
        self
```

Jugando con la herencia

- Queremos implementar un programa que calcule áreas, siguiendo el siguiente diagrama uml



- Calcula el área de un rectángulo, una elipse y un triángulo
- Funcionaría para un cuadrado?

5 Módulos

Módulos

- Los módulos permiten dividir el código de las aplicaciones y poder reutilizar el código de las aplicaciones
- En Python, cada uno de nuestros archivos .py se denominan módulos.
- Los módulos pueden ser cargados mediante la orden

  import nombre_modulo

- Para crear un módulo es tan sencillo como crear un fichero con **nombre_modulo.py**
- Referencia:
  - https://docs.python.org/3/tutorial/modules.html

Módulos empaquetados

- Para que una carpeta pueda ser considerada un paquete, debe contener un archivo de inicio llamado **__init__.py**
- Los paquetes, a la vez, también pueden contener otros sub-paquetes
- Los módulos, no necesariamente, deben pertenecer a un paquete

```
.
├── modulo1.py
└── paquete
    ├── __init__.py
    ├── modulo1.py
    └── subpaquete
        ├── __init__.py
        ├── modulo1.py
        └── modulo2.py
```

Módulos

- Se pueden importar módulos enteros o funciones o valores específicos definidos en un módulo

```
import modulo          # importar un módulo que no pertenece a un paquete
import paquete.modulo1 # importar un módulo que está dentro de un paquete
import paquete.subpaquete.modulo1
```

Namespaces

- Para acceder (desde el módulo donde se realizó la importación), a cualquier elemento del módulo importado, se realiza mediante el namespace, seguido de un punto (.) y el nombre del elemento que se desee obtener.
- En Python, un namespace, es el nombre que se ha indicado luego de la palabra import, es decir la ruta (namespace) del módulo:

```
print(modulo.CONSTANTE_1)
print(paquete.modulo1.CONSTANTE_1)
print(paquete.subpaquete.modulo1.CONSTANTE_1)
```

Alias

- Es posible también, abreviar los namespaces mediante un alias. Para ello, durante la importación, se asigna la palabra clave as seguida del alias con el cuál nos referiremos en el futuro a ese namespace importado

```
import modulo as m
import paquete.modulo1 as pm
import paquete.subpaquete.modulo1 as psm

print(m.CONSTANTE _1)
print(pm.CONSTANTE _1)
print(psm.CONSTANTE_1)
```

Importar módulos sin utilizar namespaces

- En Python, es posible también, importar de un módulo solo los elementos que se desee utilizar. Para ello se utiliza la instrucción **from** seguida del **namespace**, más la instrucción **import** seguida del elemento que se desee importar

```
from paquete.modulo1 import CONSTANTE_1
print(CONSTANTE _1)

from paquete.modulo1 import CONSTANTE_1 as C1, CONSTANTE_2 as C2
from paquete.subpaquete.modulo1 import CONSTANTE_1 as CS1, CONSTANTE_2
as CS2

print(C1)
print(C2)
print(CS1)
print(CS2)
```

Creando módulos

- Crea un módulo empaquetado que permita calcular cualquier tipo de forma plana
- Genera luego un script que pregunte por el tipo de forma a calcular y entregue la respuesta

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es