



## Programador (Orientado a Objetos) [Nivel 2]

Lección 4 / Actividad 1

### Aplicación avanzada de la programación orientada a objetos

#### IMPORTANTE

Para resolver tu actividad, **guárdala** en tu computadora e **imprímela**.

Si lo deseas, puedes conservarla para consultas posteriores ya que te sirve para reforzar tu aprendizaje. No es necesario que la envíes para su revisión.

#### Propósito de la actividad

Reforzar conceptos avanzados del paradigma orientado a objetos e implementar la sintaxis de Python para reutilizar código y extender funciones creadas.

#### Practica lo que aprendiste

- I. Escribe sobre la línea correspondiente una H si el enunciado se refiere a la Herencia, una E si se trata de Encapsulamiento o una A si es sobre clases Abstractas.

  H   Permite a una clase usar los métodos y atributos definidos en otras clases.

  A   Las clases definidas a partir de la estructura declarada en otras clases se les llama implementaciones.

  H   Las clases definidas a partir de la definición otra clase se denominan subclases.

  E   Los atributos y métodos que se usan sólo dentro del objeto pueden ser ocultos para evitar su uso en el programa principal.

  E   Los atributos y métodos se ocultan agregando dos guiones al inicio de su identificador.

  A   La implementación debe contar con una definición para cada método declarado bajo el decorador @abstractmethod.

  H   Si en la clase nueva existe una definición nueva para un método de la superclase se dice que el método está sobreescrito.



E Si se intenta acceder a un método o atributo oculto desde el programa principal, éste no lo reconocerá como miembro del objeto.

A No se puede crear una instancia directa de una clase cuya definición comienza con la instrucción "\_\_metaclass\_\_ = ABCMeta".

- II. Observa el código de Python, anota en la primera línea si se trata de herencia, encapsulamiento o abstracción, y contesta las preguntas.

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre
    def hablar(self, *palabras):
        for frase in palabras:
            print self.nombre, ': ', frase
class Deportista(Persona):
    def __init__(self, edad, nombre, deporte):
        self.edad = edad
        self.nombre = nombre
        self.deporte = deporte
    def practicarDeporte(self):
        print self.nombre, ": voy a practicar",
self.deporte
luis = Deportista(18, "Luis", "natacion")
luis.hablar("Hola ", "Estoy hablando")
luis.practicarDeporte()
```

- a) Tipo: Herencia
- b) ¿Cuál es la superclase?  
Persona
- c) ¿Cuál es la subclase?  
Deportista
- d) ¿Qué método se sobrescribe?  
El metodo constructor



```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre
    def hablar(self, *palabras):
        for frase in palabras:
            print self.nombre, ': ', frase
class Deportista(Persona):
    def __init__(self, edad, nombre, deporte):
        self.edad = edad
        self.nombre = nombre
        self.__deporte = deporte
    def practicarDeporte(self):
        print self.nombre, ": voy a practicar"
    def verMiDeporte(self):
        return self.__deporte;
luis = Deportista(18, "Luis", "natación")
luis.hablar("Hola, estoy hablando", "Este soy yo")
luis.practicarDeporte()
print "Luis practica", luis.verMiDeporte()
```

a) Tipo: Encapsulamiento

b) ¿Qué atributo está oculto?

Deporte

c) ¿Qué método se usa para manipular el atributo oculto?

VerMiDeporte



```
from abc import ABCMeta, abstractmethod
class Persona:
    __metaclass__ = ABCMeta
    def __init__(self, nombre):
        self.nombre = nombre
    def hablar(self, frase):
        print self.nombre, ': ', frase
    @abstractmethod
    def ejercitar(self): pass
class Deportista(Persona):
    def __init__(self, edad, nombre, deporte):
        self.edad = edad
        self.nombre = nombre
        self.deporte = deporte
    def competir(self):
        print self.nombre, ": voy a comeptir en", self.deporte
    def hablar(self, *frases):
        for frase in frases:
            print self.nombre, ': ', frase
    def ejercitar(self):
        print self.nombre, ": voy a practicar", self.deporte
luis = Deportista(18, "Luis", "natacion")
luis.hablar("Hola, estoy hablando", "Este soy yo")
luis.competir()
luis.ejercitar()
```

a) Tipo: Abstracta

b) ¿Qué clase es la implementación?

Deportista

c) ¿Qué método es el que se está implementando?

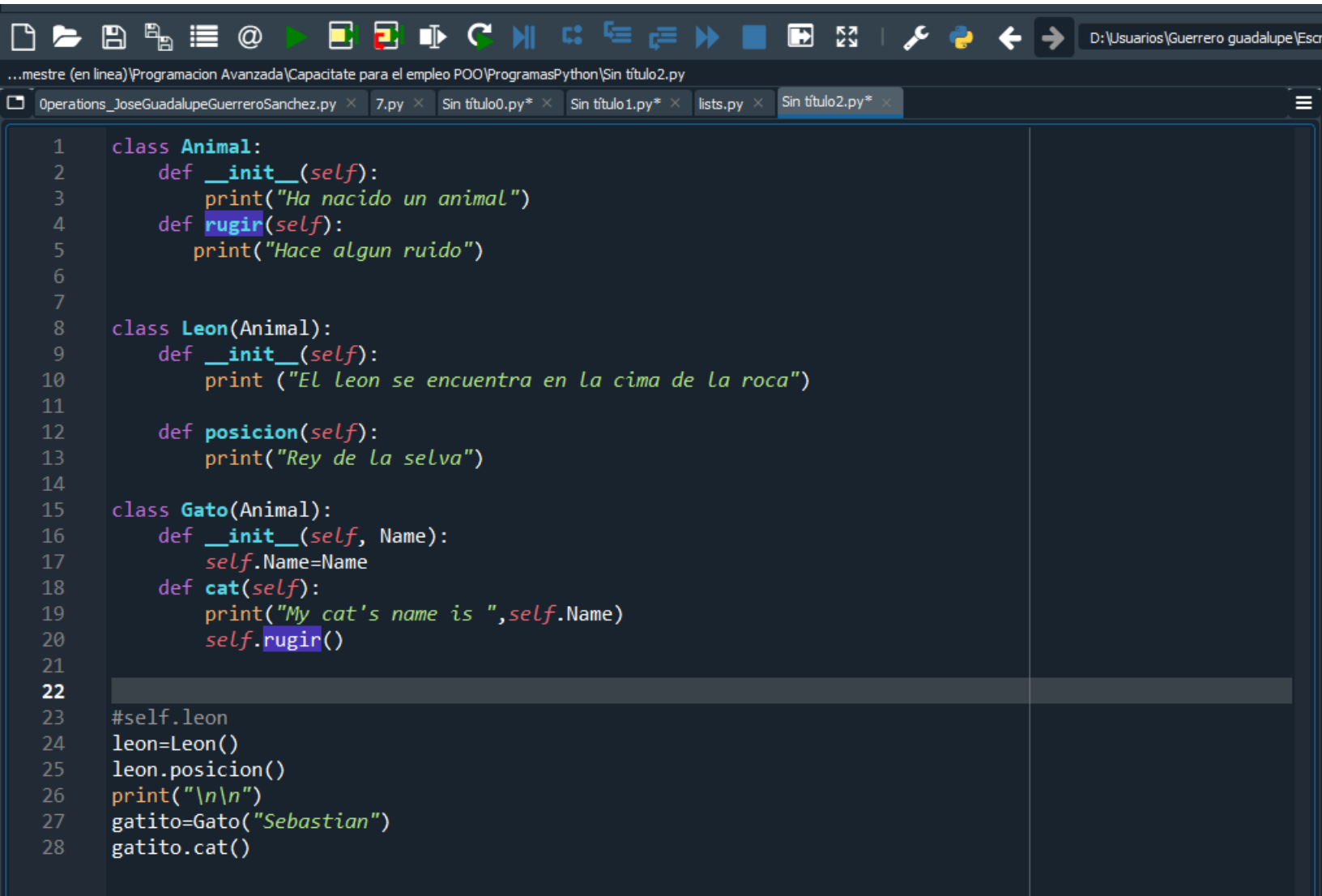
hablar

III. Usa tu IDE para crear dos subclases de la siguiente clase :

```
class Animal:
    def __init__(self):
        print "Ha nacido un animal"

    def rugir(self):
        print "Hace algun ruido"
```

### Practica lo que aprendiste (III)



```
1 class Animal:
2     def __init__(self):
3         print("Ha nacido un animal")
4     def rugir(self):
5         print("Hace algun ruido")
6
7
8 class Leon(Animal):
9     def __init__(self):
10        print ("El leon se encuentra en la cima de la roca")
11
12    def posicion(self):
13        print("Rey de la selva")
14
15 class Gato(Animal):
16     def __init__(self, Name):
17         self.Name=Name
18     def cat(self):
19         print("My cat's name is ",self.Name)
20         self.rugir()
21
22
23 #self.leon
24 leon=Leon()
25 leon.posicion()
26 print("\n\n")
27 gatito=Gato("Sebastian")
28 gatito.cat()
```

```
In [104]: runfile('D:/Usuarios/Guerrero guadalupe/Escritorio/ITQ/Quinto Semestre (en linea)/Programacion Avanzada/
Capacitate para el empleo P00/ProgramasPython/Sin título2.py', wdir='D:/Usuarios/Guerrero guadalupe/Escritorio/ITQ/Quinto
Semestre (en linea)/Programacion Avanzada/Capacitate para el empleo P00/ProgramasPython')
El leon se encuentra en la cima de la roca
Rey de la selva
```

```
My cat's name is Sebastian
Hace algun ruido
```

```
In [105]:
```



- IV. Crea dos subclases de "FiguraRegular" que definan el método para calcular su área respectiva según el número de lados, y almacenarla en el atributo "area". Crea una instancia de cada una.

```
class FiguraRegular:
    def __init__(self, lado = 3):
        self.lado = lado
        self.__area = 0
    def verArea(self):
        return self.__area
```

- V. Crea dos implementaciones de la siguiente clase abstracta:

```
from abc import ABCMeta, abstractmethod
class Transporte:
    __metaclass__ = ABCMeta

    def __init__(self, medio):
        self.medio = medio
        #Usar el atributo "medio" para definir como
        avanza

    @abstractmethod
    def avanzar(self, frase): pass

    def giraIzquierda(self):
        print "Gira a la izquierda"

    def giraDerecha(self):
        print "Gira a la derecha"

    #De acuerdo al "medio" especificar que hace para
    frenar
    @abstractmethod
    def detener(self): pass
```

## Practica lo que aprendiste (IV)

```
1 import math
2
3 class FiguraRegular:
4     def __init__(self, lado = 3):
5         self.lado = lado
6         self.__area = 0
7     def verArea(self):
8         return self.__area
9     def setArea(self, A):
10        self.__area=A
11
12 class TriangleEq(FiguraRegular):
13     def area(self):
14         b=math.sqrt((self.lado**2)-((self.lado)/2)**2)
15         Ar=((self.lado/2)*b)/2
16         self.setArea(Ar)
17
18 class Square(FiguraRegular):
19     def area(self):
20         Area=self.lado*self.lado;
21         self.setArea(Area)
22
23 Equilateral=TriangleEq(8)
24 Equilateral.area()
25 print("Equilateral triangle area: ",Equilateral.verArea())
26
27 square=Square(6)
28 square.area()
29 print("Square area: ",square.verArea())
30
```

```
Terminal 1/A x
In [120]: runfile('D:/Usuarios/Guerrero guadalupe/Escritorio/ITQ/Quinto Semestre (en linea)/Programacion Avanzada/ProgramasPython/Sin titulo3.py', wdir='D:/Usuarios/Guerrero guadalupe/Escritorio/ITQ/Quinto Semestre (en linea)/Programacion Avanzada/ProgramasPython')
Equilateral triangle area:  13.856406460551018
Square area:  36

In [121]:
```

## Practica lo que aprendiste (V)



```
1 #Crea dos implementaciones de la siguiente clase abstracta:
2
3 from abc import ABCMeta, abstractmethod
4 class Transporte:
5     __metaclass__ = ABCMeta
6     def __init__(self, medio):
7         self.medio = medio
8     #Usar el atributo "medio" para definir como avanza
9     @abstractmethod
10    def avanzar(self, frase): pass
11    def giraIzquierda(self):
12        print("Gira a La izquierda")
13    def giraDerecha(self): print("Gira a La derecha")
14    #De acuerdo al "medio" especificar que hace para frenar
15    @abstractmethod
16    def detener(self): pass
17
18 class Public(Transporte):
19     def avanzar(self,*frase):
20         for frase in frase:
21             print(self.medio,frase)
22
23     def detener(self,*word):
24         for word in word:
25             print(self.medio,word)
26
27 class Private(Transporte):
28     def avanzar(self, *frase):
29         for frase in frase:
30             print(self.medio,frase)
31
32     def detener(self, *commentary):
33         for commentary in commentary:
34             print(self.medio,commentary)
35
36 publico=Public("Autobus")
37 publico.avanzar("se esta evanzando por el pavimento")
38 publico.giraIzquierda()
39 publico.giraDerecha()
40 publico.detener("se ha detenido el autobus")
41
42 private=Private("Coche")
43 private.avanzar("avanzamos hacia el destino")
44 private.giraDerecha()
45 private.giraIzquierda()
46 private.detener("Nos detendremos en La gasolineria")
47
```



```
In [129]: runfile('D:/Usuarios/Guerrero guadalupe/Escritorio/ITQ/Quinto Semestre (en linea)/Programacion Avanzada/
ProgramasPython/Sin título4.py', wdir='D:/Usuarios/Guerrero guadalupe/Escritorio/ITQ/Quinto Semestre (en linea)/
Programacion Avanzada/ProgramasPython')
Autobus se esta evanzando por el pavimento
Gira a la izquierda
Gira a la derecha
Autobus se ha detenido el autobus
Coche avanzamos hacia el destino
Gira a la derecha
Gira a la izquierda
Coche Nos detendremos en la gasolineria
```