Universidad Católica

San Pablo

Programación Orientada a Objetos - POO

Programación Básica con Python

Graciela Meza Lovón, Yessenia Yari Ramos, Alvaro Mamani Aliaga

Concepto : Objeto

- Los objetos son la clave para entender la **Programación Orientada a Objetos**.
- En nuestro alrededor hay un sin fin de objetos: perro, escritorio, televisor, bicicleta, etc.

Concepto: Atributos

- Los atributos o propiedades de los objetos son las características que puede tener un objeto:

Objeto : Perro

Atributos :

- tamaño,
- edad,
- color,
- raza,

Concepto: Método

- Los métodos son la acción o función que realiza un objeto.

Objeto : Perro

Métodos:

- caminar,
- ladrar,
- saltar,
- dormir,
- etc...

Concepto: Instancia

- **Recordar:**
  - Una clase es una estructura general del objeto.
  - Por ejemplo, podemos decir que la clase Perro necesita tener un nombre, tamaño, edad, color, raza,
- **Una instancia es** una copia específica de la clase con todo su contenido.
- **Ejemplo**: Pimpon = Perro (“Pequeño, “4”, “marrón”, “Schnauzer”)

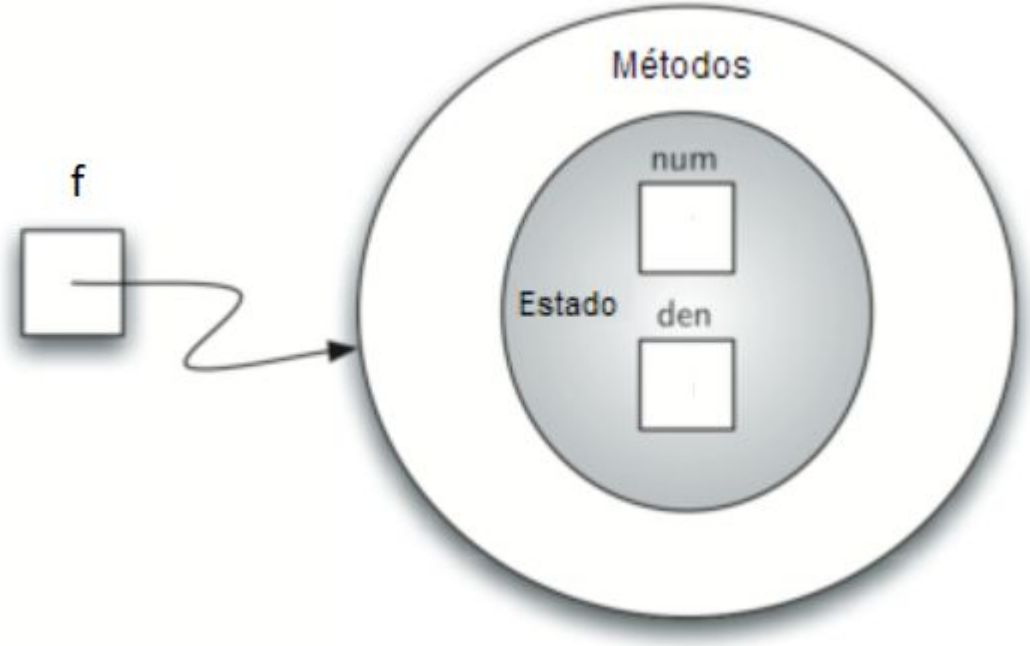
Concepto : Clase

- Una **clase es una plantilla genérica de un objeto**.
- La clase proporciona variables iniciales de estado (donde se guardan los atributos) e implementaciones de comportamiento (métodos)

Ejemplo

```
1   class fraccion:
2       def __init__(self,arriba,abajo):
3           self.num = arriba
4           self.den = abajo
5
6       def mostrar(self):
7           print(self.num, "/", self.den)
8
9   f=fraccion(7,3)
10  f.mostrar()
11
```

Ejemplo



Ejemplo

- Implementemos la clase Punto
- Un punto tiene coordenadas x, y.

```
1   class punto:
2       def __init__(self,x,y):
3           self.x=x
4           self.y=y
5
6       def mostrar(self):
7           print(self.x, ",", self.y)
8
```

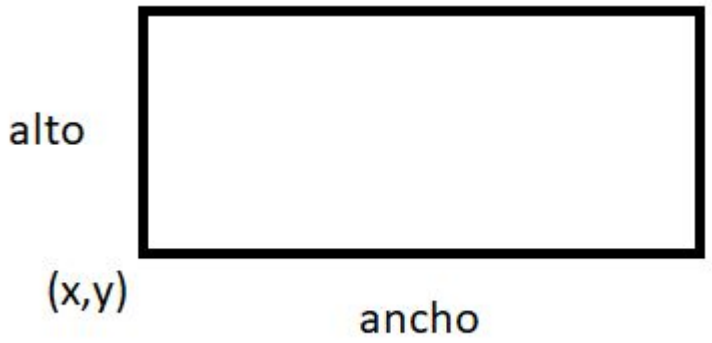
- def ___init___ (self,x,y):
  - constructor siempre se llama ___init___
  - self, llama al mismo objeto.
  - x,y son atributos del objeto

```
25  def main():
26      esq = punto(3,4)
27      esq.mostrar()
```

```
PS C:\Users\Yessi> & C:/Users/Yessi/.conda/envs/python3/python.exe d:/EjerciciosPython/classes2.py
3 , 4
```

Ejemplo

- Implementemos la clase Rectángulo
- Rectángulo tiene como atributos un punto inicial, ancho y el alto.



Ejemplo

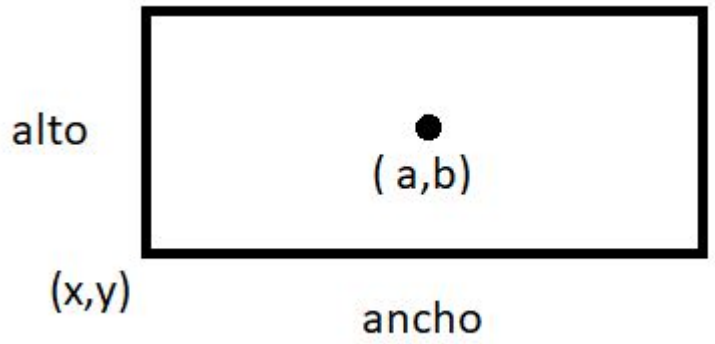
```
 9   class rectangulo:
10       def __init__(self,a,h,p):
11           self.ancho=a
12           self.alto=h
13           self.esquina=p
14
15       def mostrar(self):
16           print(self.ancho,self.alto, self.esquina.x, self.esquina.y)
```

```
25   def main():
26       esq = punto(3,4)
27       esq.mostrar()
28
29       rect = rectangulo(10,20,esq)
30       rect.mostrar()
```

```
PS C:\Users\Yessi> & C:/Users/Yessi/.conda/envs/python3/python.exe d:/EjerciciosPython/classes2.py
3 , 4
10 20 3 4
```

Ejemplo

Crear una función para la clase rectángulo que devuelva el centro del rectángulo.



Ejemplo

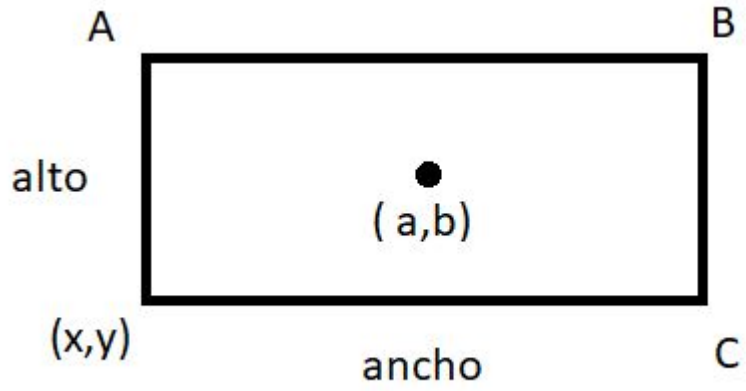
```
19      def centro(self):
20          p = punto(0,0)
21          p.x = self.esquina.x + self.ancho/2
22          p.y = self.esquina.y + self.alto/2
23          return p
```

```
25  def main():
26      esq = punto(3,4)
27      rect = rectangulo(10,20,esq)
28
29      centro = rect.centro()
30      centro.mostrar()
```

```
PS C:\Users\Yessi> & C:/Users/Yessi/.conda/envs/python3/python.exe d:/EjerciciosPython/classes2.py
8.0 , 14.0
```

Ejemplo

Encuentre los puntos punto A,B,C del rectángulo (incorpore funciones a la clase rectángulo).



Mismidad

- Si dos Puntos son el mismo, ¿significa que contienen los mismos datos (coordenadas) o que son de verdad el mismo objeto?
- Para averiguar si dos referencias se refieren al mismo objeto, se utiliza el operador ==. ejemplo:

```
p1 = punto(3,4)
p2 = punto(3,4)
print(p1==p2)      : Imprime en pantalla FALSE, por qué?
```

p1 y p2 contienen las mismas coordenadas, no son el mismo objeto (dirección memoria)

Mismidad

p3 = p2

print(p3==p2) … Imprime en pantalla TRUE, por qué? (alias)

- Igualdad superficial: sólo compara las referencias, pero no el contenido de los objetos.
- Igualdad profunda : Compara los contenidos de los objetos

Copiar

- Copiar un objeto es, muchas veces, una alternativa a la creación de un alias. El módulo copy contiene una función llamada copy que puede duplicar cualquier objeto:

```
p4 = copy.copy(p1)
print(p1==p2)   ... Imprime FALSE
```

Sobrecarga de Operadores

- Algunos lenguajes hacen posible cambiar la definición de los operadores primitivos cuando se aplican sobre tipos definidos por el programador.
- Esta característica se denomina sobrecarga de operadores. Es especialmente útil para definir tipos de datos matemáticos.
- Por ejemplo, para sobrecargar el operador suma, +, proporcionamos un método denominado add :

```
def __add__(self, t1):
```

Ejemplo

```
3   class punto:
4       def __init__(self,x,y):
5           self.x=x
6           self.y=y
7
8       def __str__(self):
9           return '({0}, {1})'.format(self.x, self.y)
10
11      def __add__(self, other):
12          return punto(self.x + other.x, self.y + other.y)
13
14      def mostrar(self):
15          print("Punto: ", self.x, ",", self.y)
```

```
33  def main():
34      p1 = punto(3,4)
35      p2 = punto(7,9)
36      p3 = p1 +p2
37      print(p3)
```

Constructor

Convierte en cadena al obj. punto

Sobrecarga

Función Mostrar

Ejemplo: Reloj

- Genere una clase reloj que tenga:
  - Atributos: hora, minuto, segundo.
  - Métodos : SumarHora, mostrar, convertirSegundos,

Ejemplo: Reloj

```python
class reloj:
    def __init__(self,h,min,seg):
        self.hora=h
        self.minuto=min
        self.segundo=seg

    def __str__(self):
        return str(self.hora) + ":" + str(self.minuto)+ ":" + str(self.segundo)

    def mostrar(self):
        print("Reloj =",self.hora,":",self.minuto,":",self.segundo)

    def __add__(self, t1):
        hh = self.hora + t1.hora
        mm = self.minuto + t1.minuto
        ss = self.segundo + t1.segundo
        if ss >= 60:
            ss = ss - 60
            mm = mm + 1
        if mm >= 60:
            mm = mm - 60
            hh = hh + 1
        return reloj(hh,mm,ss)
```

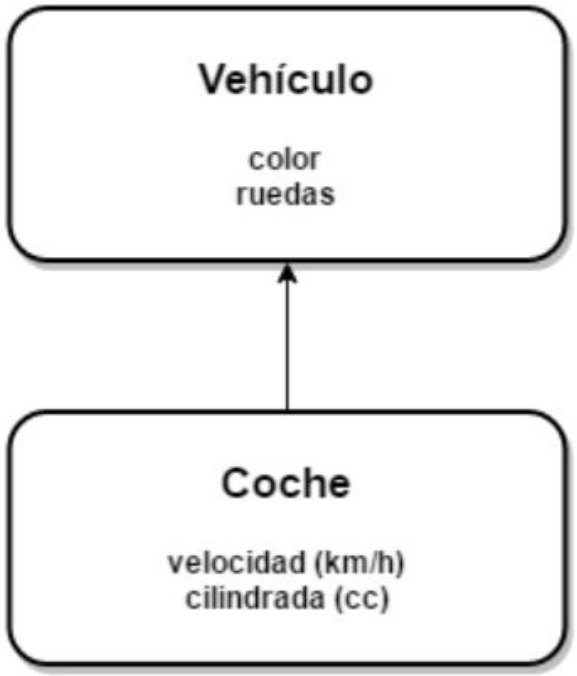
Herencia

```python
class Vehiculo():

    def __init__(self, color, ruedas):
        self.color = color
        self.ruedas = ruedas

    def __str__(self):
        return "Color {}, {} ruedas".format( self.color, self.ruedas )

class Coche(Vehiculo):

    def __init__(self, color, ruedas, velocidad, cilindrada):
        self.color = color
        self.ruedas = ruedas
        self.velocidad = velocidad
        self.cilindrada = cilindrada

    def __str__(self):
        return "color {}, {} km/h, {} ruedas, {} cc".format( self.color, self.velocidad, self.ruedas, self.cilindrada )
```

Herencia

- La característica mas asociada con la programación orientada a objetos quizás sea la herencia.
- Esta es la capacidad de definir una nueva clase que constituye una versión modificada de una clase preexistente.
- La principal ventaja de la herencia consiste en que se pueden agregar nuevos métodos a una clase sin modificarla. El nombre “herencia” se usa porque la nueva clase hereda todos los métodos de la clase existente.
- Extendiendo esta metáfora, la clase preexistente se denomina la clase madre . La nueva clase puede llamarse clase hija o, “subclase.”

Herencia



Herencia

```
2   class Vehiculo():
3
4       def __init__(self, color, ruedas):
5           self.color = color
6           self.ruedas = ruedas
7
8       def __str__(self):
9           return "Color {}, {} ruedas".format( self.color, self.ruedas )
10
11  class Coche(Vehiculo):
12
13      def __init__(self, color, ruedas, velocidad, cilindrada):
14          Vehiculo.__init__(self, color, ruedas)
15          self.velocidad = velocidad
16          self.cilindrada = cilindrada
17
18      def __str__(self):
19          return Vehiculo.__str__(self) + ", {} km/h, {} cc".format(self.velocidad, self.cilindrada)
```

¡Gracias!