

Programación Orientada a

Objetos



python

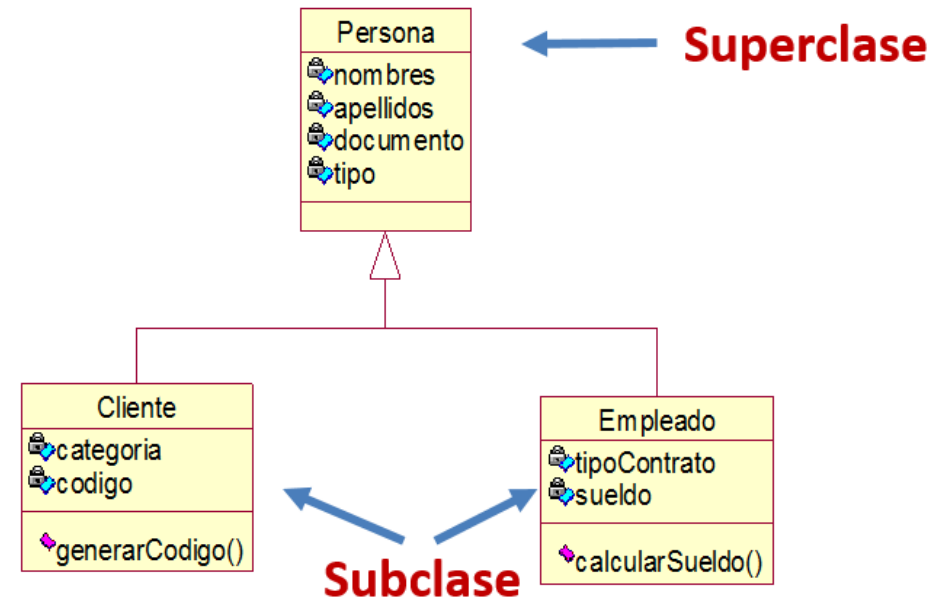
03

Herencia



Contenido

- Sobre el video anterior
- Herencia



Sobre el Video anterior

Más métodos especiales

- `X = len(obj)` # `__len__()`

Comparison Operator Overloading in Python

Operator	Expression	Internally
Less than	<code>p1 < p2</code>	<code>p1.__lt__(p2)</code>
Less than or equal to	<code>p1 <= p2</code>	<code>p1.__le__(p2)</code>
Equal to	<code>p1 == p2</code>	<code>p1.__eq__(p2)</code>
Not equal to	<code>p1 != p2</code>	<code>p1.__ne__(p2)</code>
Greater than	<code>p1 > p2</code>	<code>p1.__gt__(p2)</code>
Greater than or equal to	<code>p1 >= p2</code>	<code>p1.__ge__(p2)</code>

Operator Overloading Special Functions in Python

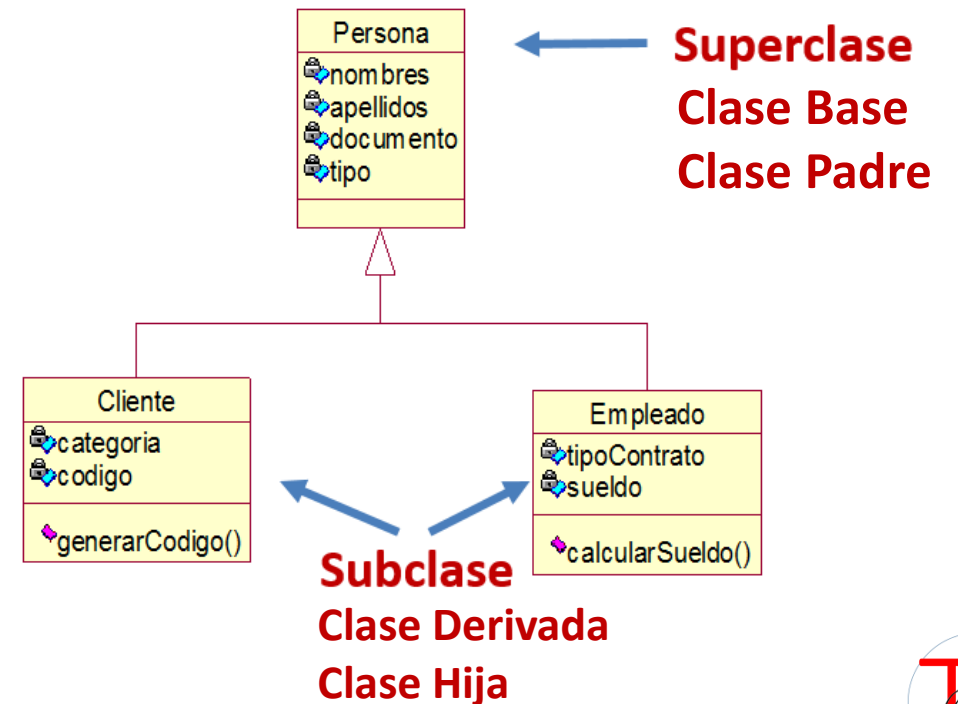
Operator	Expression	Internally
Power	<code>p1 ** p2</code>	<code>p1.__pow__(p2)</code>
Floor Division	<code>p1 // p2</code>	<code>p1.__floordiv__(p2)</code>
Remainder (modulo)	<code>p1 % p2</code>	<code>p1.__mod__(p2)</code>
Bitwise Left Shift	<code>p1 << p2</code>	<code>p1.__lshift__(p2)</code>
Bitwise Right Shift	<code>p1 >> p2</code>	<code>p1.__rshift__(p2)</code>
Bitwise AND	<code>p1 & p2</code>	<code>p1.__and__(p2)</code>
Bitwise OR	<code>p1 p2</code>	<code>p1.__or__(p2)</code>
Bitwise XOR	<code>p1 ^ p2</code>	<code>p1.__xor__(p2)</code>
Bitwise NOT	<code>-p1</code>	<code>p1.__invert__()</code>



Herencia

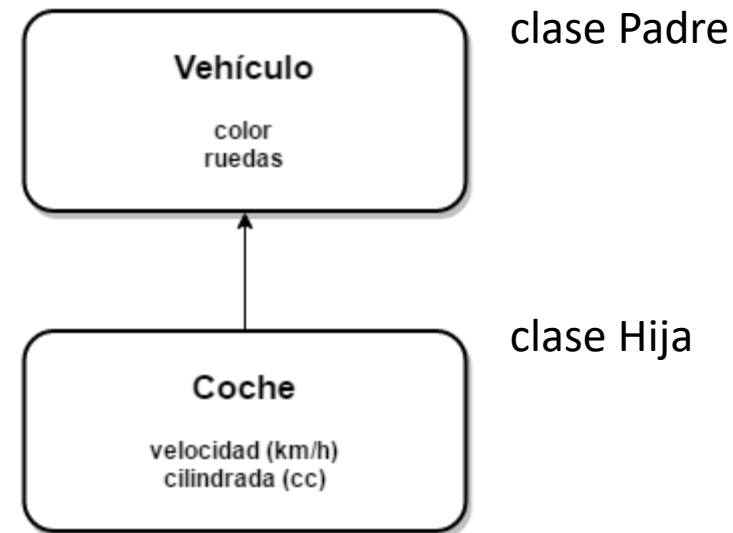
- La herencia es la capacidad que tiene una clase de heredar los atributos y métodos de otra, algo que nos permite reutilizar código y hacer programar mucho más óptimos.

```
class BaseClass:  
    Body of base class  
class DerivedClass(BaseClass):  
    Body of derived class
```



Herencia ejemplo

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



Herencia

Uso del nombre de la clase Padre

- Para acceder a los miembros de la clase padre
- Es necesario cuando en la clase hija existen miembros con el mismo nombre

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



Herencia super()

- Para acceder a los miembros de la clase Padre.
- Se evita incluir **self**

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

`super().__init__(color, ruedas)`

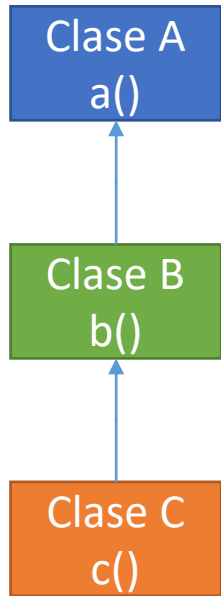


Fracciones mixtas

$$6\frac{3}{4} + 3\frac{1}{2} + 2\frac{5}{6}$$

Herencia

(situaciones)

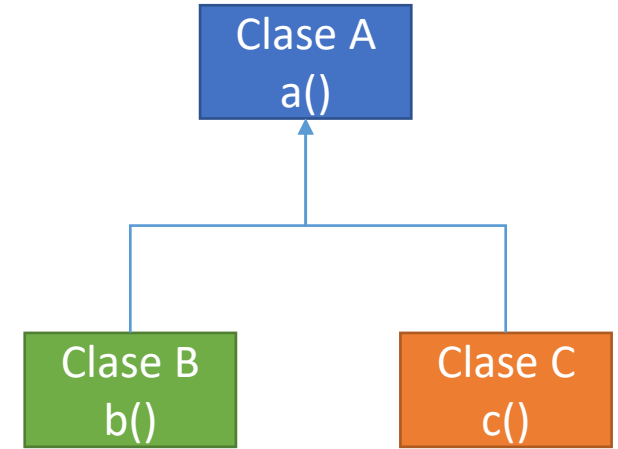


```
class A:
    def __init__(self):
        print("Soy de clase A")
    def a(self):
        print("Este método lo heredo de A")

class B(A):
    def __init__(self):
        print("Soy de clase B")
    def b(self):
        print("Este método lo heredo de B")

class C(B):
    def c(self):
        print("Este método es de C")

c = C()
c.a()
c.b()
c.c()
```



```
class A:
    def __init__(self):
        print("Soy de clase A")
    def a(self):
        print("Este método lo heredo de A")

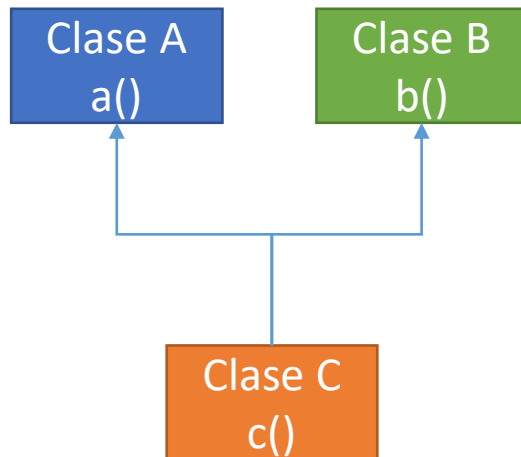
class B(A):
    def __init__(self):
        print("Soy de clase B")
    def b(self):
        print("Este método lo heredo de B")

class C(A):
    def c(self):
        print("Este método es de C")

oc = C()
oc.a()
oc.c()
```

Herencia múltiple

- Herencia múltiple: la capacidad de una subclase de heredar de **múltiples** superclases.



```
class A:
    def __init__(self):
        print("Soy de clase A")
    def a(self):
        print("Este método lo heredo de A")
```

```
class B:
    def __init__(self):
        print("Soy de clase B")
    def b(self):
        print("Este método lo heredo de B")
```

```
class C(B,A):
    def c(self):
        print("Este método es de C")
```

```
c = C()
c.a()
c.b()
c.c()
```



Herencia

Métodos importantes

- `isinstance(obj, int)`
- `issubclass(bool, int)`



Programación Orientada a

Objetos



python

03

Herencia

