Programación Orientada a

Objetos

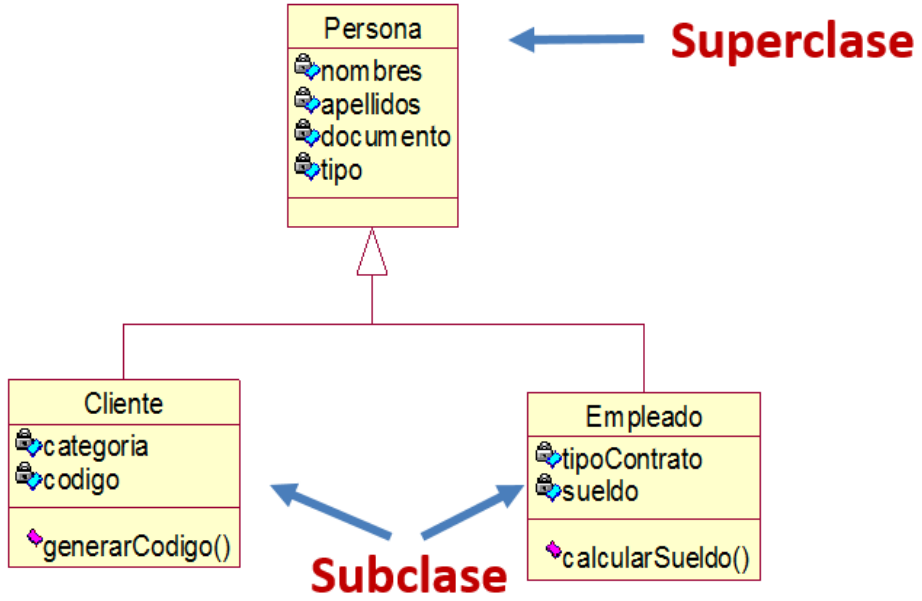
python

03

Herencia

Contenido

- Sobre el video anterior
- Herencia



Sobre el Video anterior
Más métodos especiales

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

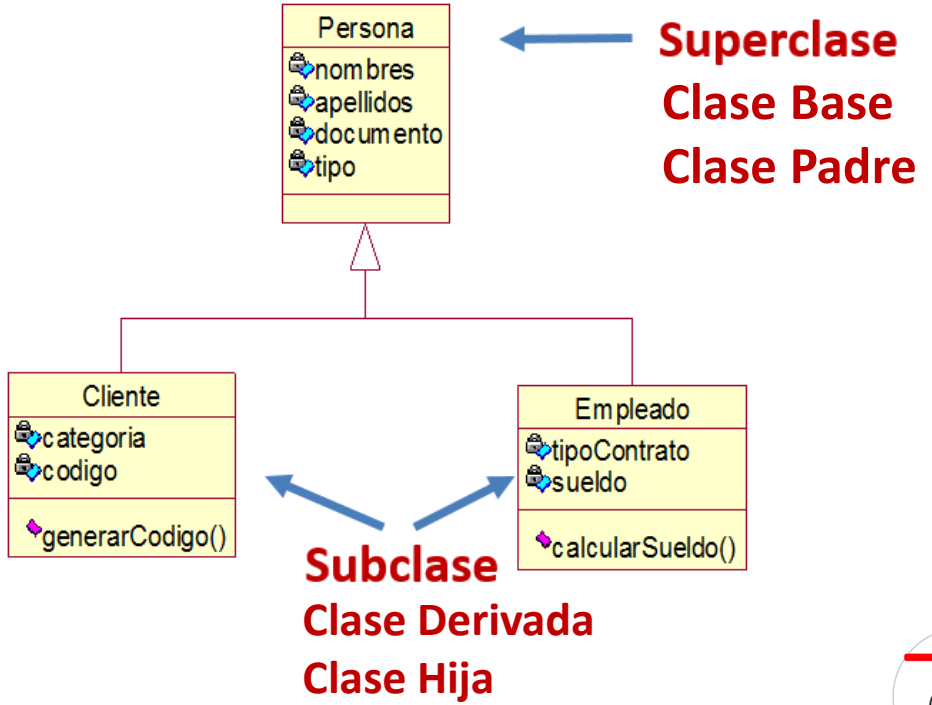
Comparision Operator Overloading in Python		
Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

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

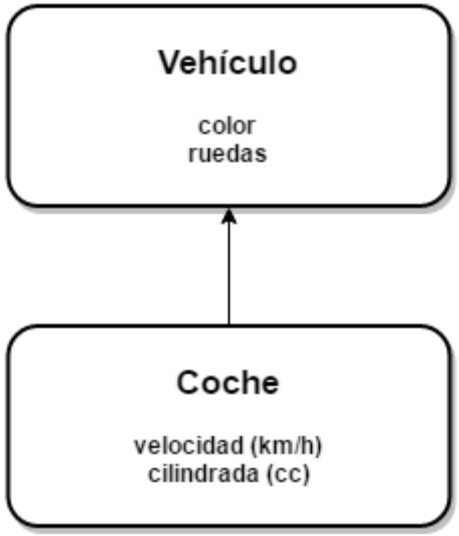
Herencia

- 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



clase Padre

clase Hija

```
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

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

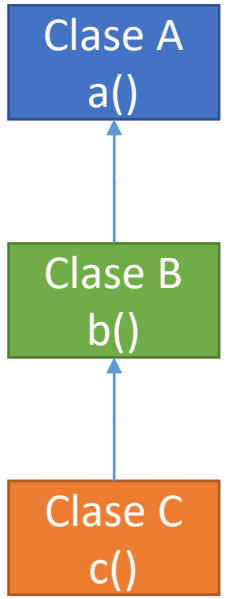
```
class Vehiculo():

    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
```

Fracciones mixtas

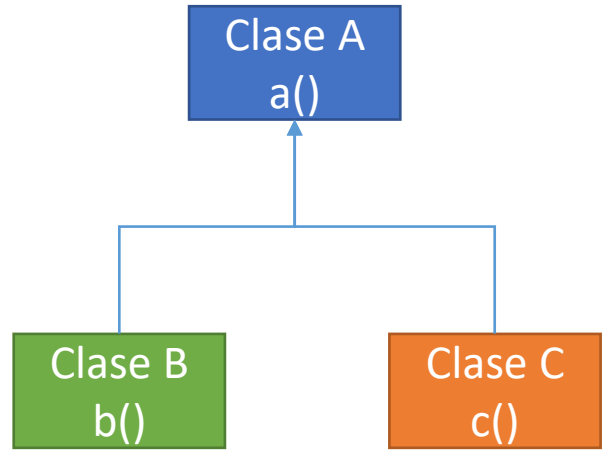
$$6\frac{3}{4} + 3\frac{1}{2} + 2\frac{5}{6}$$

Turbo Código
https://www.youtube.com/user/juarezefren

Herencia
(situaciones)



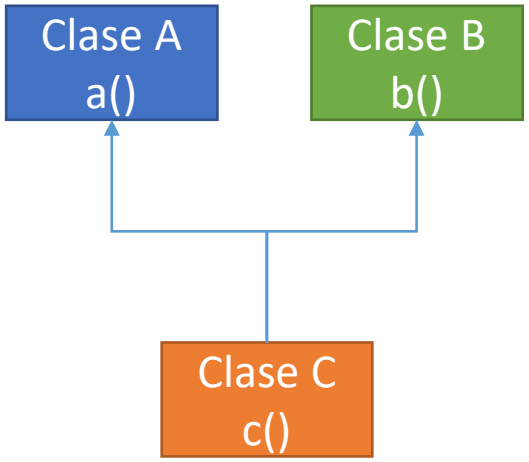

```
class A:
    def __init__(self):
        print("Soy de clase A")
    def a(self):
        print("Este método lo heredo de A")

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

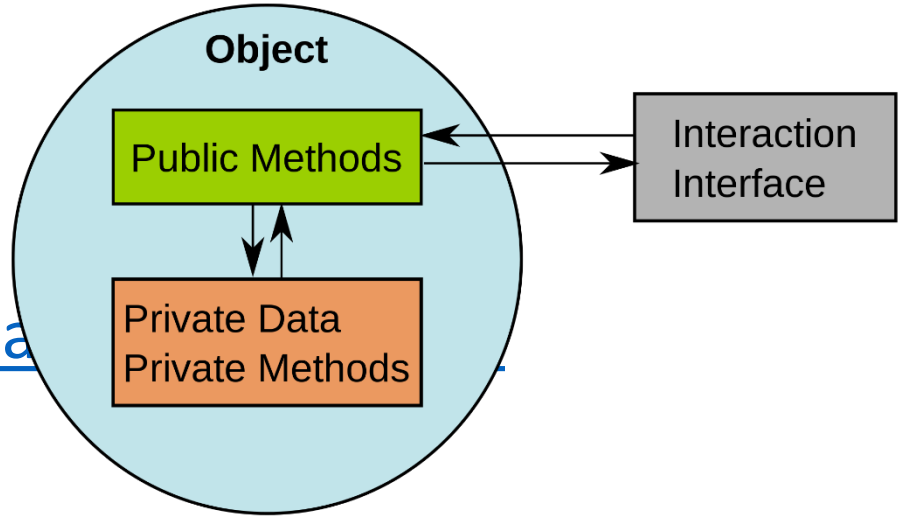
Objetos

python

03

Herencia

Encapsulamiento
(encapsulación)



- https://www.geeksforgeeks.org/encapsula
- https://pythonspot.com/encapsulation/
- https://www.programiz.com/python-programming/property