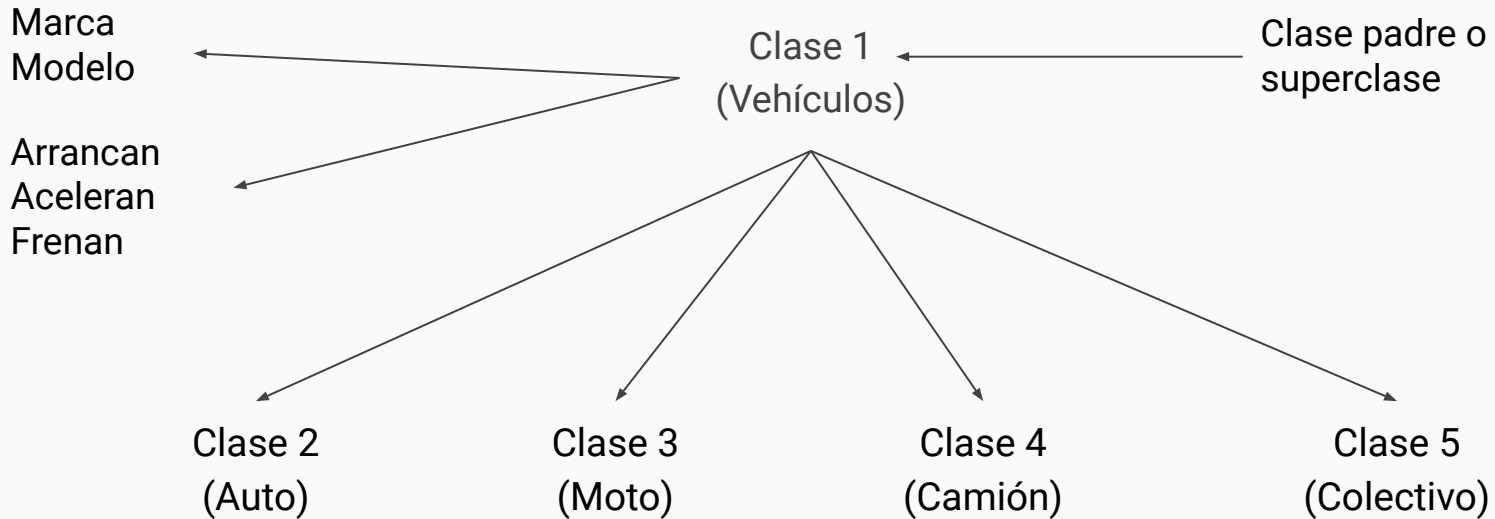


P00: Programación orientada a objetos (Parte 4)

Programación 2



Herencia



Herencia: Superclase

```
class Vehiculos():  
    def __init__(self, marca, modelo):  
        self.marca=marca  
        self.modelo=modelo  
        self.enmarcha=False  
        self.acelera=False  
        self.frena=False
```

Herencia: Superclase

```
class Vehiculos():  
    def __init__(self, marca, modelo):  
        self.marca=marca  
        self.modelo=modelo  
        self.enmarcha=False  
        self.acelera=False  
        self.frena=False  
  
    def arrancar(self):  
        self.enmarcha=True  
  
    def acelerar(self):  
        self.acelera=True  
  
    def frenar(self):  
        self.frena=True  
  
    def estado(self):  
        print("Marca: ", self.marca, "y Modelo: ", self.modelo)
```

Herencia: Subclases

```
class Vehiculos():
    def __init__(self, marca, modelo):
        self.marca=marca
        self.modelo=modelo
        self.enmarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):
        self.enmarcha=True

    def acelerar(self):
        self.acelera=True

    def frenar(self):
        self.frena=True

    def estado(self):
        print("Marca: ", self.marca, "y Modelo: ",
self.modelo)
```

```
class Moto(Vehiculos):
    (...)
```

```
miMoto = Moto("Honda", "CBR")
miMoto.estado()
```

Herencia: Subclases

```
class Vehiculos():
    def __init__(self, marca, modelo):
        self.marca=marca
        self.modelo=modelo
        self.enmarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):
        self.enmarcha=True

    def acelerar(self):
        self.acelera=True

    def frenar(self):
        self.frena=True

    def estado(self):
        print("Marca: ", self.marca, "y Modelo: ",
self.modelo)
```

```
class Moto(Vehiculos):
    hwheelie = ""

    def wheelie(self):
        hwheelie = "Haciendo Wheelie"
```

Herencia: Sobrescribir un método

```
class Vehiculos():
    def __init__(self, marca, modelo):
        self.marca=marca
        self.modelo=modelo
        self.enmarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):
        self.enmarcha=True

    def acelerar(self):
        self.acelera=True

    def frenar(self):
        self.frena=True

    def estado(self):
        print("Marca: ", self.marca, "y Modelo: ", self.modelo)
```

```
class Moto(Vehiculos):
    hwheelie = ""

    def wheelie(self):
        hwheelie = "Haciendo Wheelie"

    def estado(self):
        ↑ print("Marca: ", self.marca, "y Modelo: ", self.modelo)
        print(self.hwheelie)
```

Método sobrescrito

Herencia: Sobrescribir un método

```
class Vehiculos():  
    def __init__(self, marca, modelo):  
        self.marca=marca  
        self.modelo=modelo  
        self.enmarcha=False  
        self.acelera=False  
        self.frena=False
```

```
    def arrancar(self):  
        self.enmarcha=True
```

```
    def acelerar(self):  
        self.acelera=True
```

```
    def frenar(self):  
        self.frena=True
```

```
    def estado(self):  
        print("Marca: ", self.marca, "y Modelo: ", self.modelo)
```

```
class Moto(Vehiculos):  
    hwheelie = ""
```

```
    def wheelie(self):  
        self.hwheelie = "Haciendo Wheelie"
```

```
    def estado(self):  
        print("Marca: ", self.marca, "y Modelo: ", self.modelo)  
        print(self.hwheelie)
```

```
miMoto = Moto("Honda", "CBR")
```

```
miMoto.estado()
```



Herencia: Otro ejemplo

```
class Vehiculos():
    def __init__(self, marca, modelo):
        self.marca=marca
        self.modelo=modelo
        self.enmarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):
        self.enmarcha=True

    def acelerar(self):
        self.acelera=True

    def frenar(self):
        self.frena=True

    def estado(self):
        print("Marca: ", self.marca, "y Modelo: ",
self.modelo)
```

```
class Moto(Vehiculos):
    hwheelie = ""

    def wheelie(self):
        self.hwheelie = "Haciendo Wheelie"

    def estado(self):
        print("Marca: ", self.marca, "y Modelo: ",self.modelo)
        print(self.hwheelie)
```

```
class Camion(Vehiculo):

    def carga(self, cargar):
        self.cargado = cargar
        if(self.cargado):
            return "El camión está cargado"
        else:
            return "El camión no está cargado"

miCamion = Camion("Mack", "cv713")
miCamion.arrancar()
miCamion.estado()
miCamion.carga(True)
```

Herencia múltiple

```
class VElectricos():  
    def __init__(self):  
        self.autonomia=100  
  
    def cargarEnergia(self):  
        self.cargando=True
```

```
class BicicletaElectrica(VElectricos, Vehiculos):  
    (...)
```

Herencia múltiple

```
class VElectricos():  
  
    def __init__(self):  
        self.autonomia=100  
  
    def cargarEnergia(self):  
        self.cargando=True
```

```
class BicicletaElectrica(VElectricos, Vehiculos):  
    (...)
```

```
miBici=BicicletaElectrica()
```



¿Qué constructor hereda? ¿Cual es la forma correcta?



```
miBici=BicicletaElectrica("marca","modelo")  
o  
miBici=BicicletaElectrica()
```

Herencia múltiple

```
class VElectricos():  
    def __init__(self):  
        self.autonomia=100  
  
    def cargarEnergia(self):  
        self.cargando=True
```

```
class BicicletaElectrica(VElectricos, Vehiculos):  
    (...)
```

```
miBici=BicicletaElectrica("marca","modelo") ← Forma incorrecta  
o  
miBici=BicicletaElectrica() ← Forma correcta de crear el objeto
```

Herencia múltiple

```
class VElectricos():  
    def __init__(self):  
        self.autonomia=100  
  
    def cargarEnergia(self):  
        self.cargando=True
```

```
class BicicletaElectrica(Vehiculos, VElectricos):  
    (...)
```

```
miBici=BicicletaElectrica("marca","modelo") ← Forma correcta  
o  
miBici=BicicletaElectrica() ← Forma correcta de crear el objeto
```

Herencia: función super()

```
class Persona():
```

```
    def __init__(self, nombre, edad, lugar_residencia):  
        self.nombre=nombre  
        self.edad=edad  
        self.lugar_residencia=lugar_residencia
```

```
    def descripcion(self):  
        print("Nombre: ", self.nombre)  
        print("Edad: ", self.edad)  
        print("Residencia: ", self.lugar_residencia)
```

```
class Empleado():
```

```
    def __init__(self, salario, antigüedad):  
        self.salario=salario  
        self.antigüedad=antigüedad
```

Herencia: función super()

```
class Persona():
```

```
    def __init__(self, nombre, edad, lugar_residencia):  
        self.nombre=nombre  
        self.edad=edad  
        self.lugar_residencia=lugar_residencia
```

```
    def descripcion(self):  
        print("Nombre: ", self.nombre)  
        print("Edad: ", self.edad)  
        print("Residencia: ", self.lugar_residencia)
```

```
class Empleado(Persona):
```

```
    def __init__(self, salario, antigüedad):  
        self.salario=salario  
        self.antigüedad=antigüedad
```

```
Gabriel = Persona("Gabriel", 22, "Argentina")
```

```
Lucas=Empleado(100000, 10)  
Lucas.descripcion()
```

Herencia: función super()

```
class Persona():
```

```
    def __init__(self, nombre, edad, lugar_residencia):  
        self.nombre=nombre  
        self.edad=edad  
        self.lugar_residencia=lugar_residencia
```

```
    def descripcion(self):  
        print("Nombre: ", self.nombre)  
        print("Edad: ", self.edad)  
        print("Residencia: ", self.lugar_residencia)
```

```
class Empleado(Persona):
```

```
    def __init__(self, salario, antigüedad, nombre, edad, lugar_residencia):  
        super().__init__(nombre, edad, lugar_residencia)  
        self.salario=salario  
        self.antigüedad=antigüedad
```




Diagram illustrating inheritance: Three arrows point from the `Empleado` class definition to the `Persona` class definition, indicating that `Empleado` inherits from `Persona`.

```
Lucas=Empleado(100000, 10, "Lucas", 25, "Argentina")  
Lucas.descripcion()
```




Diagram illustrating object creation: An arrow points from the `Lucas=Empleado(...)` line to the `Empleado` class definition, indicating that the `Lucas` object is an instance of the `Empleado` class.

Herencia: función super()

```
class Persona():
```

```
    def __init__(self, nombre, edad, lugar_residencia):  
        self.nombre=nombre  
        self.edad=edad  
        self.lugar_residencia=lugar_residencia
```

```
    def descripcion(self):  
        print("Nombre: ", self.nombre)  
        print("Edad: ", self.edad)  
        print("Residencia: ", self.lugar_residencia)
```

```
class Empleado(Persona):
```

```
    def __init__(self, salario, antigüedad, nombre, edad, lugar_residencia):  
        super().__init__(nombre, edad, lugar_residencia)  
        self.salario=salario  
        self.antigüedad=antigüedad
```

```
    def descripción(self):  
        super().descripción()  
        print("salario: ", self.salario)  
        print("antigüedad: ", self.antigüedad)
```

```
Lucas=Empleado(100000, 10, "Lucas", 25, "Argentina")  
Lucas.descripcion()
```

Herencia: función super()

```
class VElectricos(Vehiculos):  
  
    def __init__(self,marca,modelo):  
        super().__init__(marca,modelo)  
        self.autonomia=100  
  
    def cargarEnergia(self):  
        self.cargando=True
```

```
class BicicletaElectrica(VElectricos, Vehiculos):  
    (...)
```

```
miBici=BicicletaElectrica("marca","modelo")
```



Polimorfismo

Indica que un objeto puede cambiar dependiendo el contexto en que se lo utilice y de esta manera también cambia su comportamiento.

Herramienta muy potente en la programación orientada a objetos.

Veamos un ejemplo...

Polimorfismo

```
class Auto():
```

```
    def desplazamiento(self):  
        print("me desplazo utilizando 4 ruedas")
```

```
class Moto():
```

```
    def desplazamiento(self):  
        print("me desplazo utilizando 2 ruedas")
```

```
class Camion():
```

```
    def desplazamiento(self):  
        print("me desplazo utilizando 6 ruedas")
```

Polimorfismo

```
class Auto():  
    def desplazamiento(self):  
        print("me desplazo utilizando 4 ruedas")
```

```
class Moto():  
    def desplazamiento(self):  
        print("me desplazo utilizando 2 ruedas")
```

```
class Camion():  
    def desplazamiento(self):  
        print("me desplazo utilizando 6 ruedas")
```

```
mivehiculo=moto()  
mivehiculo.desplazamiento()  
#me desplazo utilizando 2 ruedas"
```

```
mivehiculo2=auto()  
mivehiculo2.desplazamiento()  
#me desplazo utilizando 4 ruedas"
```

```
mivehiculo3=camion()  
mivehiculo3.desplazamiento()  
#me desplazo utilizando 6 ruedas"
```

Polimorfismo

```
class Auto():  
  
    def desplazamiento(self):  
        print("me desplazo utilizando 4 ruedas")
```

```
class Moto():  
  
    def desplazamiento(self):  
        print("me desplazo utilizando 2 ruedas")
```

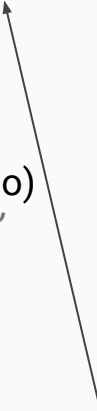
```
class Camion():  
  
    def desplazamiento(self):  
        print("me desplazo utilizando 6 ruedas")
```

```
def desplazamientoVehiculo(vehiculo):  
    vehículo.desplazamiento()
```

Polimorfismo

```
class Auto():  
    def desplazamiento(self):  
        print("me desplazo utilizando 4 ruedas")  
  
class Moto():  
    def desplazamiento(self):  
        print("me desplazo utilizando 2 ruedas")  
  
class Camion():  
    def desplazamiento(self):  
        print("me desplazo utilizando 6 ruedas")
```

```
def desplazamientoVehiculo(vehiculo):  
    vehiculo.desplazamiento()  
  
miVehiculo=Camion()  
  
desplazamientoVehiculo(miVehiculo)  
#me desplazo utilizando 6 ruedas"
```

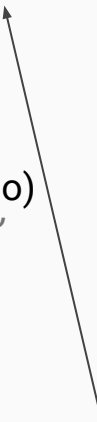


Aquí ocurre el polimorfismo

Polimorfismo

```
class Auto():  
    def desplazamiento(self):  
        print("me desplazo utilizando 4 ruedas")  
  
class Moto():  
    def desplazamiento(self):  
        print("me desplazo utilizando 2 ruedas")  
  
class Camion():  
    def desplazamiento(self):  
        print("me desplazo utilizando 6 ruedas")
```

```
def desplazamientoVehiculo(vehiculo):  
    vehiculo.desplazamiento()  
  
miVehiculo=Auto()  
  
desplazamientoVehiculo(miVehiculo)  
#me desplazo utilizando 4 ruedas"
```



Aquí ocurre el polimorfismo