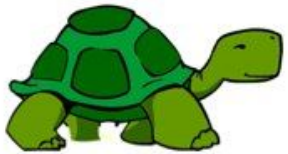
Introducción a la programación con Python
Herencia

Alexis Rodríguez
Marcel Morán C

Esquema

- Mi libro de animales
- ¿Que es herencia?
- Sintaxis de herencia
- Mi libro de animales con herencia
- ¿Que es una clase abstracta?
- Sintaxis de clase abstracta

Un programa de animales



Un libro de animales

```
class Perro:
  def __init__(self, edad, nombre, sonido):
    self.edad = edad
    self.nombre = nombre
    self.sonido = sonido


  def describete(self):
    print("Hola me llamo", self.nombre, "tengo", self.edad, "años y",
self.sonido)


max = Perro(10, "Max", "ladr@",)
max.describete()
>>> Hola me llamo Max tengo 10 años y ladr@
```

Un libro de animales

```
class Gato:
  def __init__(self, edad, nombre, sonido):
    self.edad = edad
    self.nombre = nombre
    self.sonido = sonido


  def describete(self):
    print("Hola me llamo", self.nombre, "tengo", self.edad, "años y",
self.sonido)


gato = Gato(15, "Mineta", "maull@",)
gato.describete()
>>> Hola me llamo Mineta tengo 15 años y maull@
```

Un libro de animales

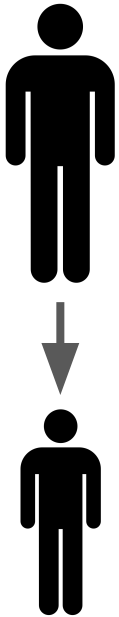
```
class Cuervo:
  def __init__(self, edad, nombre, sonido):
    self.edad = edad
    self.nombre = nombre
    self.sonido = sonido


  def describete(self):
    print("Hola me llamo", self.nombre, "tengo", self.edad, "años y",
self.sonido)


cuervo = Cuervo(5, "Itachi", "grazn@",)
cuervo.describete()
>>> Hola me llamo Itachi tengo 5 años y grazn@
```

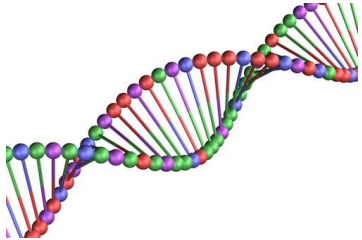
Un libro de animales

```
class Caballo:
  def __init__(self, edad, nombre, sonido):
    self.edad = edad
    self.nombre = nombre
    self.sonido = sonido


  def describete(self):
    print("Hola me llamo", self.nombre, "tengo", self.edad, "años y",
self.sonido)


caballo = Caballo(8, "Spirit", " ladr@")
caballo.describete()
>>> Hola me llamo Spirit tengo 8 años y relinch@
```

Programación Orientada a Objetos

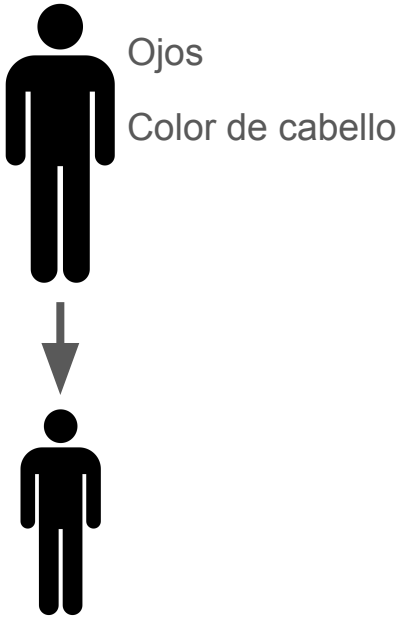
- Es un derivado de una clase que hereda propiedades de su clase
- La clase padre hereda sus propiedades a sus hijos

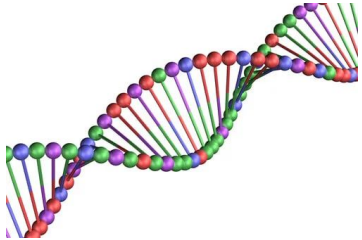




Programación Orientada a Objetos

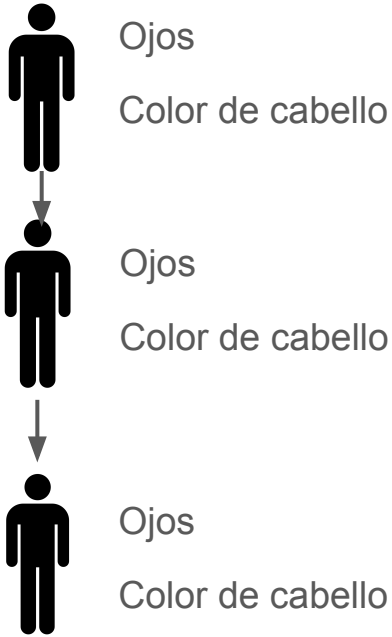
- Es un derivado de una clase que hereda propiedades de su clase
- La clase padre hereda sus propiedades a sus hijos

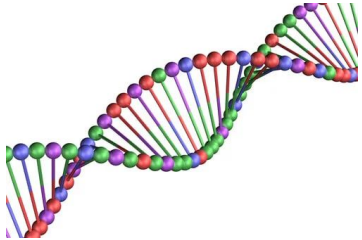




Programación Orientada a Objetos

- Es un derivado de una clase que hereda propiedades de su clase
- La clase padre hereda sus propiedades a sus hijos
- Clases superiores





Sintaxis de Herencia

- Asumiendo que tenemos nuestro clase Padre
- **class** Hijo(**Padre**)
- **super().__init__**(argumentos_n1, argumentos_n2)

Un libro de animales

```
class Animal:
  def __init__(self, edad, nombre, sonido):
    self.edad = edad
    self.nombre = nombre
    self.sonido = sonido


  def describete(self):
    print("Hola me llamo", self.nombre, "tengo", self.edad, "años y",
self.sonido)
```

Un libro de animales

```
class Perro(Animal):
  def __init__(self, edad, nombre, sonido):
      super().__init__(edad, nombre, sonido)


class Gato(Animal):
  def __init__(self, edad, nombre, sonido):
      super().__init__(edad, nombre, sonido)

class Cuervo(Animal):
  def __init__(self, edad, nombre, sonido):
      super().__init__(edad, nombre, sonido)

class Caballo(Animal):
  def __init__(self, edad, nombre, sonido):
      super().__init__(edad, nombre, sonido)
```

Un libro de animales

```
perro = Perro(10, "max", "ladr@")
gato = Gato(15, "Mineta", "maull@")
cuervo = Cuervo(5, "Itachi", "grazn@")
caballo = Caballo(8, "Spirit", "ladr@")

perro.describete()
gato.describete()
cuervo.describete()
caballo.describete()

>>>Hola me llamo max tengo 10 años y ladr@
>>>Hola me llamo Mineta tengo 15 años y maull@
>>>Hola me llamo Itachi tengo 5 años y grazn@
>>>Hola me llamo Spirit tengo 8 años y ladr@
```

Clases abstractas

- Una clase abstracta es aquella que implemente un método abstracto
- Permite a los hijos implementar definir los métodos abstractos
- Clases abstractas **no pueden ser instanciadas**

wuah

kro

jegi

miau

Sintaxis de clases abstractas

- from **abc** import **ABC, abstractmethod**
- decorador/decorator encima del método **@abstractmethod**
- **class nombre(ABC)**

wuah

kro

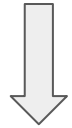
jegi

miau

Clases abstractas - El problema

```
class Animal:
 def   init__(self, edad, nombre,
sonido):
   self.edad = edad
   self.nombre = nombre
   self.sonido = sonido

def describete(self):
 print("Hola me llamo", self.nombre, "tengo",
self.edad, "años y", self.sonido)


def hablar(self):
  print("Hola me llamo", self.nombre, "tengo",
self.edad, "años y", self.sonido)
```

```
animal 1 = Animal(10, "max", "ladr@")
animal_2 = Animal(5, "Itachi", "grazn@")


animal 1.describete()
animal_2.describete()

animal 1.hablar()
animal_2.hablar()
```



```
Hola me ll          ños y ladr@
Hola me ll           años y grazn@
Hola me ll          ños y ladr@
Hola me ll           años y
grazn@
```

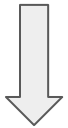
Clases abstractas - La solución

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    def __init__(self, edad, nombre, sonido):
        self.edad = edad
        self.nombre = nombre
        self.sonido = sonido
```

```python
@abstractmethod
def describete(self):
    pass
```

```python
@abstractmethod
def hablar(self):
    pass
```

```python
animal_1 = Animal(10, "max", "ladr@")
animal_2 = Animal(5, "Itachi", "grazn@")

animal_1.describete()
animal_2.describete()

animal_1.hablar()
animal_2.hablar()
```

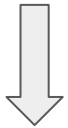


```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

~\AppData\Local\Temp/ipykernel_26580/1939502414.py in <module>
----> 1 animal_1 = Animal(10, "max", "ladr@")
      2 animal_2 = Animal(5, "Itachi", "grazn@")
      3
      4
      5 animal_1.describete()

TypeError: Can't instantiate abstract class Animal with abstract methods
describete, hablar
```

Clases abstractas - Subclases

```
class Gato(Animal):
    def   init  (self, edad, nombre, sonido):
          super().__init__(edad, nombre, sonido)


    def describete(self):
        print("Hola me llamo", self.nombre, "tengo",
self.edad, "años y", self.sonido)


    def hablar(self):
        print("Hola me llamo", self.nombre, "tengo",
self.edad, "años y", self.sonido)
```

```
gato = Gato(15, "Mineta", "maull@")
gato.describete()
gato.hablar()
```



```
Hola me llamo Mineta tengo 15
años y maull@
Hola me llamo Mineta tengo 15
años y maull@
```

Conclusión

- Uno de los principios OOP se aplica con la herencia
- Una clase hereda propiedades de otra clase o clase superior
- Sintaxis de python para heredar los atributos y métodos de otras class nombre (Clase Superior)
- Clases abstractas no pueden ser instanciadas pero permiten la implementación de sus clases
- from **abc** import **ABC, abstractmethod @abstractmethod**