

Departamento de Física Médica - Centro atómico Bariloche - IB

Introducción a la POO en Python

Ariel Hernán Curiale ariel.curiale@cab.cnea.gov.ar

CONICET









Paradigmas de programación

- Paradigmas de programación: Un paradigma de programación consiste en un método para llevar a cabo cómputos.
 - * Procedural: se definen procedimientos que operan sobre estructuras de datos. Pensar en una receta de cocina (C, fortran)
 - * Objetos: se definen objetos que encapsula estados y su comportamiento. La interacción entre ellos es a través del pasaje de mensajes, y su colaboración es la que resuelve el problema
 - * Orientada a Objetos: se basa en el paradigma procedural donde el estado y su funcionalidad se encapsula en los objetos (C++, Java, Python)
 - * Funcional: Se definen funciones y casos base o condiciones de parada. Se pueden crear listas infinitas (Haskell)
 - Lógico: basado en la definición de relaciones lógicas (Prolog)

Diseño orientado a objetos

- * Encapsulamiento: Se define una interfaz pública y privada con el objetivo de establecer cómo se interactúa con el objeto. ¿Porque es útil? ¿Ejemplos?
- * Polimorfismo: la capacidad de sobre-escribir (overload) el comportamiento predefinido de operaciones ¿Porque es útil? ¿Ejemplos?
- * Herencia: la habilidad de crear subclases que contengan una especialización de sus padres. Permite heredar comportamientos. ¿Porque es útil? ¿Ejemplos?

Diseño orientado a objetos

```
[13]: class animal():
   ...: def correr(self):
              print('estoy corriendo')
In [14]: class pato(animal):
   ...: def correr(self):
              super().correr()
              print('medio lento prefiero volar')
[n [15]: class perro(animal):
  ...: def correr(self):
   super().correr()
   print('muy rápido')
[17]: pa = pato()
[n [18]: p.correr()
estoy corriendo
muy rápido
In [19]: pa.correr()
estoy corriendo
medio lento prefiero volar
```

¿Qué es un objeto en este paradigma?

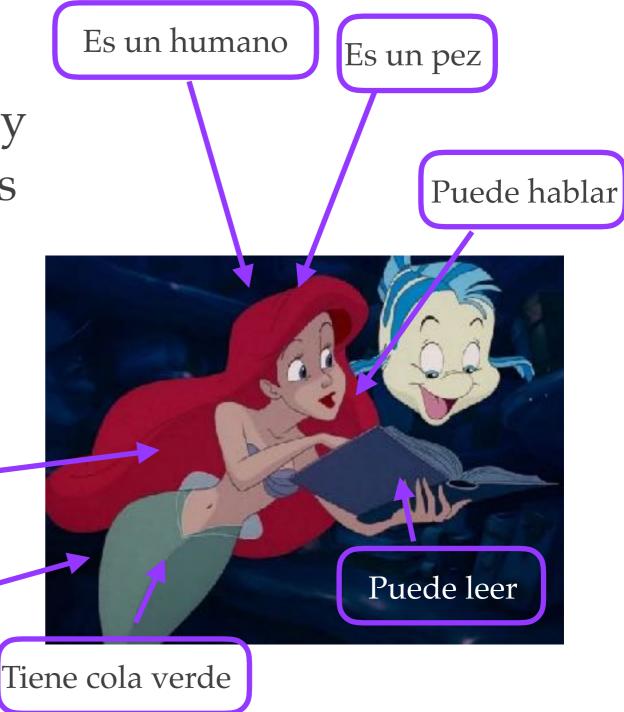
- * Es un código que tiene variables y métodos (funciones):
 - Las variables codifican el estado del objeto
 - Los métodos definen su comportamiento

¿Qué es un clase?

La clase define la funcionalidad y sus atributos de todos los objetos de la clase. Las clases son variables de tipo.

Tiene pelo rojo

Puede nadar



Clases en Python

- Una clase es un objeto con diferentes características:
 - * Se puede llamar como a una función y retorna un objeto de dicha clase
 - * Tiene una cantidad arbitraria de atributos
 - * Los atributos pueden ser datos o métodos (funciones)
 - Una clase puede heredar de otras clases
- * Convención de nombres: clases mayúscula, métodos, atributos y funciones minúscula.

```
class Humano(object):
                                                      Definimos un método
        pass
                              class Humano(object):
                                    def hablar(self):
  Clase vacía
                                         print('Hola soy un humano')
                             Un método muy especial: Constructor
class Humano(object):
   def init (self, sexo=None, pelo=None, nombre=None):
        self.sexo = sexo
       self.pelo = pelo
                                                   Atributos
        self.nombre = nombre
        self.organos = creacion organos()
        self. ordenar orgamos()
                                           Convención de nombre
   def ordenar organos(self):
                                            para método privado
      pass
   def hablar(self):
       print ('Hola, mi nombre es {}'.format(self.nombre))
```

Orden de __init__(self)

```
[4]: class First(object):
                                  [4]: class First(object):
  ...: def __init__(self):
                                  ...: def __init__(self):
  print ("first")
                                  print ("first")
                                  ...: class Second(First):
  ...: class Second(First):
  ...: def __init__(self):
                                  ...: def __init__(self):
  print ("second")
                                  print ("second")
  ...: class Third(First):
                                  ...: class Third(First):
  ...: def __init__(self):
                                  ...: def __init__(self):
  print ("third")
                                  print ("third")
                                  ...: class Fourth(Third, Second):
  ...: class Fourth(Second, Third):
  ...: def __init__(self):
                                  ...: def __init__(self):
  super().__init__()
                                  super().__init__()
  print("that's it")
                                  ...: print("that's it")
                                [n [5]: a = Fourth()
<mark>In [5]: a = Fourth()</mark>
                                third
second
that's it
                                that's it
```

Orden de __init__(self)

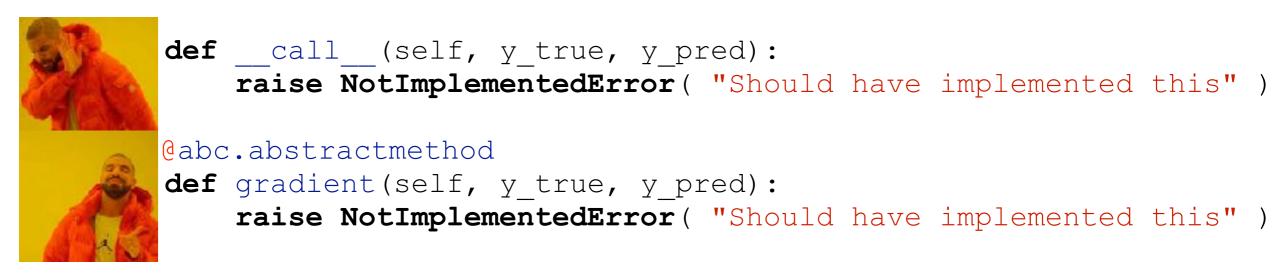
```
[16]: class First(object):
   ...: def __init__(self):
   ...: print ("first")
   ...: class Second(First):
   ...: def __init__(self):
   print ("second")
   ...: class Third(First):
   ...: def __init__(self):
   ...: print ("third")
   ...: class Fourth(Third, Second):
   ...: def __init__(self):
   Second. init (self)
   Third.__init__(self)
...: print("that's it")
In [17]: a = Fourth()
second
third
that's it
```

class Humano(object): def hablar(self): Ejemplo print('Hola soy un humano') class Humano(object): def init (self, sexo=None, pelo=None, nombre=None): self.sexo = sexoclass Pez(object): self.pelo = peloself.nombre = nombre def init (self, cola=None): **def** hablar(self): self.cola = colaprint ('Hola, mi nombre es {}'.format(self.nombre)) def nadar(self): print('Puedo nadar') Herencia múltiple Inicialización clase padre class Sirena(Humano, Pez): Atributo de la clase especie = 'Disney' **def** init (self, nombre, pelo, parientes): super(). init (sexo='F', pelo=pelo, nombre=nombre) self.parientes = parientes Lista de Sirenos y Sirenas

* Interfaz en Python (con y sin abc):

```
import abc
class Loss(object):
```

class MSE(Loss):



```
__name__ = 'mse' \leftarrow ¿Qué es esto?

def __call__(self, y_true, y_pred):
    return np.square(y_true - y_pred).sum(axis=-1).mean()

def gradient(self, y_true, y_pred):
    return -2*(y_true - y_pred)
```

Métodos especiales

- __init___: Se invoca al crearse una instancia de la clase
- __call___: Se invoca al ser llamado como función el objeto.
- str_: Se invoca cuando hace falta una representación en string de un objeto.
 Esta representación se obtiene con str(objeto) o con print(objeto).
- repr_: Se invoca con la función repr(). Es una representación imprimible del objeto
- * __getitem__: Para acceder a un objeto de forma secuencial o usando un subíndice tipo objeto[n]. Este método requiere 2 parámetros
- * __iter__: Permite definir la forma de ser iterado
- setitem_: Para asignarle el valor a una clave
- delitem_: Implementa el borrado de un objeto

Abrir el notebook 2 sección 2.7 que tengo algo de POO