



*Departamento de Física Médica - Centro atómico Bariloche - IB*

# Introducción a la POO en Python

Ariel Hernán Curiale  
[ariel.curiale@cab.cnea.gov.ar](mailto:ariel.curiale@cab.cnea.gov.ar)



**UNCUYO**  
 UNIVERSIDAD  
 NACIONAL DE CUYO



---

# Paradigmas de programación

---

- ❖ Paradigmas de programación: Un paradigma de programación consiste en un método para llevar a cabo cálculos.
- ❖ 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

```
In [13]: class animal():
...:     def correr(self):
...:         print('estoy corriendo')
...:
In [14]: class pato(animal):
...:     def correr(self):
...:         super().correr()
...:         print('medio lento prefiero volar')
...:
In [15]: class perro(animal):
...:     def correr(self):
...:         super().correr()
...:         print('muy rápido')
...:
In [16]: p = perro()
In [17]: pa = pato()
In [18]: p.correr()
estoy corriendo
muy rápido
In [19]: pa.correr()
estoy corriendo
medio lento prefiero volar
```

---

# Programación Orientada a Objetos (POO)

---

¿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

# Programación Orientada a Objetos (POO)

¿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.



---

# 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.



# Programación Orientada a Objetos (POO)

```
class Humano(object):  
    pass
```

Clase vacía

Definimos un método

```
class Humano(object):  
    def hablar(self):  
        print('Hola soy un humano')
```

```
class Humano(object):
```

Un método muy especial: Constructor

```
def __init__(self, sexo=None, pelo=None, nombre=None):  
    self.sexo = sexo  
    self.pelo = pelo  
    self.nombre = nombre  
    self.organos = creacion_organos()  
    self._ordenar_orgamos()
```

Atributos

```
def _ordenar_organos(self):  
    pass
```

Convención de nombre  
para método privado

```
def hablar(self):  
    print('Hola, mi nombre es {}'.format(self.nombre))
```



# Orden de `__init__`(self)

```
In [4]: 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(Second, Third):
...:         def __init__(self):
...:             super().__init__()
...:             print("that's it")
...:
```

```
In [5]: a = Fourth()
second
that's it
```

```
In [4]: 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):
...:             super().__init__()
...:             print("that's it")
...:
```

```
In [5]: a = Fourth()
third
that's it
```

# Orden de `__init__`(self)

```
In [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
```

# Programación Orientada a Objetos (POO)

## ❖ Ejemplo

```
class Humano(object):  
    def hablar(self):  
        print('Hola soy un humano')
```

```
class Humano(object):  
  
    def __init__(self, sexo=None, pelo=None, nombre=None):  
        self.sexo = sexo  
        self.pelo = pelo  
        self.nombre = nombre  
  
    def hablar(self):  
        print('Hola, mi nombre es {}'.format(self.nombre))
```

```
class Pez(object):
```

```
    def __init__(self, cola=None):  
        self.cola = cola
```

```
    def nadar(self):  
        print('Puedo nadar')
```

```
class Sirena(Humano, Pez):  
    especie = 'Disney'  
  
    def __init__(self, nombre, pelo, parientes):  
        super().__init__(sexo='F', pelo=pelo, nombre=nombre)  
        self.parientes = parientes
```

Herencia múltiple

Atributo de la clase

Inicialización clase padre

Lista de Sirenos y Sirenas

# Programación Orientada a Objetos (POO)

## ❖ Interfaz en Python (con y sin abc):

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



```
    def __call__(self, y_true, y_pred):
        raise NotImplementedError( "Should have implemented this" )
```

```
@abc.abstractmethod
```

```
def gradient(self, y_true, y_pred):
    raise NotImplementedError( "Should have implemented this" )
```

```
class MSE(Loss):
```

```
    __name__ = 'mse' ← ¿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