

Programación Orientada a Objetos (POO)

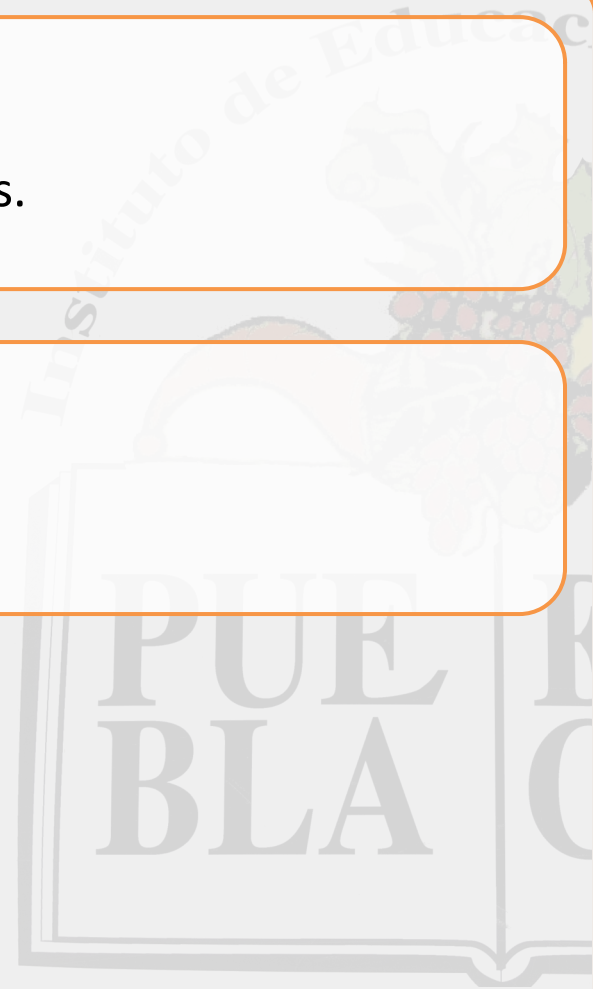
Programación de Inteligencia Artificial



POO. Clases:

- Introducción.
- Atributos.
- Constructor.
- Métodos.
- Métodos especiales.

Herencia:



Introducción

Atributos

Constructor

Métodos

Métodos Especiales

- La POO es un paradigma de programación en la que se programa entorno a los objetos.
- Un objeto es una entidad que permite guardar una serie de datos (atributos) y que suele tener una serie de funciones (métodos), que nos permite interactuar con dichos valores.
- Un objeto se crea a partir de una clase que hemos definido previamente. Para ello se usa la palabra reservada *class*.
- El nombre de las clases comienzan por mayúscula.
- La organización dentro de una clase es la misma que en otros lenguajes: primero los atributos, a continuación el constructor y por último los métodos.
- Al igual que con las funciones, se puede documentar una clase para ofrecer información a los programadores que vayan a utilizarla. Para ello, se pone un comentario con triple comillas a continuación de la declaración de la misma.

```
class Persona():  
    """Clase que almacena los datos relativos a una persona"""
```

Introducción

Atributos 1/2

Constructor

Métodos

Métodos
Especiales

- Son las variables generales de una clase que nos permite guardar la información relativa a un objeto.
- En Python, si no tienen un valor predefinido no es necesario declararlo con anterioridad. Se realizaría en el constructor (lo veremos a continuación) y son atributos propios del objeto.
- Si los declaramos al inicio de la clase con un valor predeterminado, ese atributo es un *atributo de clase*, es decir, es común para todos los objetos que creamos a partir de dicha clase.
- Cuando un atributo empieza por 1 guion bajo, queremos decir que es “privado” y deberíamos acceder a él mediante los métodos de la clase. En la práctica se puede seguir accediendo a su contenido. De igual modo, para indicar un atributo oculto se ponen 2 guiones bajos e igualmente se puede acceder.

Introducción

Atributos 2/2

Constructor

Métodos

Métodos
Especiales

```
class Persona:
    """Clase que almacena los datos relativos a una persona"""
    nombre = "Nombre desconocido"
    _apellidos = "Apellidos privados"
    __edad = 0 # Atributo oculto

def main():
    print("Atributo de clase: " + Persona.nombre)
    p = Persona()
    print(p.nombre)
    print(p._apellidos)
    print(p._Persona__edad)

if __name__ == "__main__":
    main()
```

Introducción

Atributos

Constructor

Métodos

Métodos
Especiales

- El constructor en Python se llama `__init__` . Es el que nos permite crear los objetos de una clase e inicializar los atributos de la misma.
- Recibe como parámetros los atributos que queremos inicializar. Adicionalmente, el primer parámetro ha de ser *self* que se trata de una referencia al propio objeto.
- Los atributos se pueden “crear” e inicializar en el constructor si no los hemos definido previamente con un valor predeterminado.

```
class Persona:
    """Clase que almacena los datos relativos a una persona"""

    def __init__(self, nombre, apellidos, edad = 0) -> None:
        self._nombre = nombre
        self._apellidos = apellidos
        if edad > 0:
            self._edad = edad

def main():
    p = Persona("Nieves", "Concostrina", 61)
    print(p._nombre)
    print(p._apellidos)
    print(p._edad)

if __name__ == "__main__":
    main()
```

Introducción

Atributos

Constructor

Métodos 1/3

Métodos
Especiales

- Los métodos son funciones que nos permiten realizar diferentes operaciones con la información del objeto almacenada en sus atributos.
- Se utiliza *def* para indicar que se trata de un método. De igual modo, debe recibir como mínimo el parámetro *self* que hace referencia al objeto creado.
- Si se desea devolver alguna información se usa la palabra *return*.

```
class Persona:
    """Clase que almacena los datos relativos a una persona"""

    def __init__(self, nombre, apellidos, edad = 0) -> None:
        self._nombre = nombre
        self._apellidos = apellidos
        if edad > 0:
            self._edad = edad

    def nombre_completo(self):
        return self._nombre + " " + self._apellidos

def main():
    p = Persona("Nieves", "Concostrina", 61)
    print(p.nombre_completo())

if __name__ == "__main__":
    main()
```

Introducción

Atributos

Constructor

Métodos 2/3

Métodos
Especiales

- Al igual que en otros lenguajes, existen unos métodos llamados *getters* y *setters* que nos permite mostrar o modificar el contenido de un atributo y “encapsular” los datos.
- Normalmente, se pone la palabra *get* o *set* seguida de un guion bajo y del nombre del atributo.

```
class Persona:

    def __init__(self, nombre, apellidos,
                  edad = 0):
        self._nombre = nombre
        self._apellidos = apellidos
        self.set_edad(edad)

    def set_nombre(self, nombre):
        self._nombre = nombre
    def get_nombre(self):
        return self._nombre

    def set_edad(self, edad):
        if edad > 0:
            self._edad = edad
    def get_edad(self):
        return self._edad

    def nombre_completo(self):
        return self._nombre + " " +
               self._apellidos
```

```
def main():
    p = Persona("Nieves", "Concostrina", 61)
    print(p.get_nombre())
    p.set_edad(10); print(p.get_edad())

if __name__ == "__main__":
    main()
```


Introducción

Atributos

Constructor

Métodos 3/3

Métodos
Especiales

- Otra forma de encapsular los atributos es mediante las *propiedades*.
- Se trata de usar un método que se llame como el atributo al que se le antepone la etiqueta *@property* (es similar al *getter*).
- Para modificar un dato, anteponemos la etiqueta *@nombre.setter*, donde nombre es el nombre de nuestro atributo. De igual modo que en el caso anterior, el nombre del método será como el del atributo.

```
class Persona:
    def __init__(self, nombre, apellidos,
                 edad = 0):
        self._nombre = nombre
        self._apellidos = apellidos
        self._edad = edad

    @property
    def edad(self):
        return self._edad

    @edad.setter
    def edad(self, edad):
        if edad > 0:
            self._edad = edad

    def nombre_completo(self):
        return self._nombre + " " +
               self._apellidos
```

```
def main():
    p = Persona("Nieves", "Concostrina", 61)
    print(p.edad)
    p.edad = -4
    print(p.edad)

if __name__ == "__main__":
    main()
```

Introducción

Atributos

Constructor

Métodos

Métodos
Especiales

- Son métodos especiales que nos permiten realizar diferentes acciones con un objeto: mostrar la información almacenada, comparar 2 objetos, etc. Así como realizar una operación con un objeto.
- Estos métodos comienzan y acaban por dos guiones bajos `__str__` y podemos añadirlos a nuestra clase para que realicen una acción diferente.
- Por ejemplo, si en el método principal ponemos `print(p)` obtendremos de qué clase es el objeto y una dirección de memoria. Al sobrescribir el método como se indica a continuación, nos mostrará la información almacenada en sus atributos.

```
def __str__(self) -> str:  
    return ("nombre: " + self.nombre +  
           ", apellidos: " + self._apellidos +  
           ", edad: " + str(self.edad))
```



Fondo Social Europeo