Programación Orientada a Objetos (POO)

Programación de Inteligencia Artificial

Junta de Andalucía

FPA
FORMACIÓN PROFESIONAL ANDALUZA

Fondo Social Europeo

Índice | POO. Clases | Polimorfismo | Herencia | Interfaces

POO. Clases:

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

Polimorfismo:

- Introducción.

Herencia:

- Introducción.
- Codificación.

Interfaces:

- Introducción.

- 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"""
```

- 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 creemos 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.

```python
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()
```

- 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.
- Python no tiene la posibilidad de tener constructores múltiples.

```
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()
```

- 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*.

```python
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()
```

- 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* seguidade 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()
```

- 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()
```

- 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))
```

Introducción

- El polimorfismo indica que un objeto puede tomar formas diferentes (*poly: muchas, morfo: formas*).
- Esto nos permite tener varias clases distintas de objetos y poder usarlos de forma indistinta.

```
class Perro:
    def hablar(self):
        print("GUAU")

class Gato:
    def hablar(self):
        print("MIAU")

def escuchar(animal):
    animal.hablar()

def main():
    p = Perro(); g = Gato()
    escuchar(g); escuchar(p)

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

- Esto, por ejemplo, lo hemos estado aplicando con el for al recorrer un rango, lista, cadena de caracteres, etc.

- La herencia es una de las características de los lenguajes POO.
- Nos permite crear clases a partir de otras, *herendando* su funcionalidad y añadiendo una nueva.
- En definitiva, podemos acceder a los atributos y métodos de la clase madre.
- Decimos que podemos añadir nueva funcionalidades ya que podemos añadir nuevos atributos y métodos en las clases hijas y sobrescribir alguna de las clases madre.
- Python permite la herencia múltiple, es decir, heredar de varias clases.
- Normalmente, se suele decir que se realiza una especialización.

Introducción

Codificación

- Para indicar que una clase hereda de otra se pone entre paréntesis la clase (o clases) madre de las que se quiere heredar.

```
class Alumno(Persona):
```

- Para acceder a los atributos y métodos de la clase madre se usa la función *super()*.
- Es necesario llamar al constructor de la clase madre e inicializar los atributos de esta.

```
    def __init__(self, nombre, apellidos, edad=0):
        super().__init__(nombre, apellidos, edad)
```

- Como se ha comentado, una clase hija puede tener nuevos métodos, atributos y sobrescribir métodos de la clase madre.

```
class Alumno(Persona):
    def __init__(self, nombre, apellidos, edad=0, materias = {}):
        super().__init__(nombre, apellidos,edad)
        self._materias = materias
```

```
@property
def materias(self):
    return self._materias
@materias.setter
def materias(self, materias):
    if len(materias) > 0:
        self._materias = materias
```

```
def __str__(self):
    contenido = "\nMODULOS. "
    for k, v in self.materias.items():
        contenido += k + ": " + str(v) + ", "
    return super().__str__() + contenido[:-2]
```

Introducción

- Las interfaces están presentes en la mayoría de los lenguajes con POO.
- En una interfaz se definen un conjunto de especificaciones que otras clases han de implementar, es decir, métodos que deben de tener.
- Una clase abstracta es igual que una interfaz pero puede tener métodos implementados.
- Python incorpora esta segunda opción.
- Para su uso es necesario importar la clase *ABCMeta* del módulo *abc*. La interfaz tiene que heredar esta clase. También es necesario importar *abstractmethod*.

```
from abc import ABCMeta, abstractmethod
class Reproductor(metaclass = ABCMeta):
    @abstractmethod
    def play(self):
        pass
    @abstractmethod
    def stop(self):
        pass
    @abstractmethod
    def pause(self):
        pass
```

```
class Video(Reproductor):
    def play(self):
        print("Reproduciendo vídeo")
    def stop(self):
        print("Vídeo parado")
    def pause(self):
        print("Vídeo pausado")
```

Instituto de Educación Secundaria PUEBLA DE VICAR

FPA
FORMACIÓN PROFESIONAL ANDALUZA

Fondo Social Europeo

Imágenes: Flaticon.com, IES La Puebla, Junta de Andalucía