Lección 19: Clases y objetos

Índice

Anterior

Clases y objetos

Python es un lenguaje de programación orientado a objetos. Todo en Python es un objeto, con sus propiedades y métodos. Un número, cadena, lista, diccionario, tupla, conjunto, etc. utilizado en un programa es un objeto de una clase incorporada correspondiente. Una clase es como un constructor de objetos o un "modelo" para crear objetos. Una instancia de una clase es un objeto. La clase define los atributos y el comportamiento del objeto, mientras que el objeto, por otro lado, es la materialización de una clase.

Hemos estado trabajando con clases y objetos desde el comienzo del curso.

> Nota: En Python es habitual que los nombres de clases comiencen por una letra mayúscula y que los nombres de objetos sean en minúsculas. Ej de nombres de clases: Persona, Session, Request Ej de nombres de objetos (instancias de clases): persona, persona1, session, r

Crear una clase

Python y su librería estándar tiene muchas clases con diversa funcionalidad que se pueden utilizar, pero también es posible crear nuevas clases con la palabra reservada *class*.

```
# pseudocódigo
class nombre_clase:
    'resto del código'
```

Por ejemplo:

```
class Persona:
    pass
print(Persona)
```

Creando un objeto

Puede crearse un objeto (una instancia de la clase) llamando al constructor, que en Python es siempre el nombre de la clase.

```
persona1 = Persona()
print(persona1)
```

Constructor de clases

Sin embargo, la clase *Persona* que acabamos de definir no tiene ningún constructor y por tanto es poco útil. Pero, al igual que la función constructora en Java o JavaScript, Python también tiene una función constructora *init()* incorporada . La función constructora *init* tiene un parámetro *self* que es una referencia a la instancia actual de la clase.

```
class Persona():
    version = '1.0.0' # Atributo de clase
    def __init__(self, nombre, apellido = None, edad = None, pais = None, ciudad = None):
        self.nombre = nombre        # Atributos de objeto
        self.apellido = apellido
        self.edad = edad
        self.pais = pais
        self.ciudad = ciudad

persona1 = Persona("Pedro")
print(persona1)
print(persona1.nombre)
print(persona1.version)
persona2 = Persona("Alicia")
print(persona2)
print(persona2.nombre)
print(persona2.version)
```

Métodos de objeto

Los objetos pueden tener métodos. Los métodos son funciones que pertenecen al objeto. Al igual que el constructor, siempre recibirán como primer parámetro *self* (Y no podemos olvidarnos de ponerlo)

```
class Persona():
    version = '1.0.0'
    def __init__(self, nombre, apellido = None, edad = None, pais = None, ciudad = None):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad
        self.pais = pais
        self.ciudad = ciudad
    def saluda(self):
        print(f"Hola, soy {self.nombre}. Encantado de saludarte.")
    def datos(self):
        return {
            'nombre': self.nombre,
            'apellido': self.apellido,
            'edad': self.edad,
```

```
            'pais': self.pais,
            'ciudad': self.ciudad
            }

persona1 = Persona("Pedro", apellido="Hernando", edad=54)
persona1.saluda()
print(persona1.datos())
```

Método para modificar atributos de objeto

```
class Persona():
    version = '1.0.0'
    def __init__(self, nombre, apellido = None, edad = None, pais = None, ciudad = None):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad
        self.pais = pais
        self.ciudad = ciudad
        self.idiomas = []
    def saluda(self):
        print(f"Hola, soy {self.nombre}. Encantado de saludarte.")
    def datos(self):
        return {
            'nombre': self.nombre,
            'apellido': self.apellido,
            'edad': self.edad,
            'pais': self.pais,
            'ciudad': self.ciudad,
            'idiomas': self.idiomas
            }
    def añadir_idioma(self, idioma):
        self.idiomas.append(idioma)

persona1 = Persona("Pedro", apellido="Hernando", edad=54)
persona1.añadir_idioma('Inglés')
persona1.añadir_idioma('Español')
print(persona1.datos())
```

Herencia

Usando la herencia se puede reutilizar el código de la clase principal. La herencia permite definir una clase que hereda todos los métodos y propiedades de la clase principal. La clase *principal* (o *superclase* o *base*) es la clase que proporciona todos los métodos y propiedades. La clase secundaria es la clase que hereda de otra clase o clase principal. Por ejemplo, se puede crear una clase *Estudiante* que herede de la clase *Persona*.

```python
class Estudiante(Persona):
    """Documentar"""
    # TODO: Documentar
    def __init__(self, nombre, apellido=None, edad=None, pais=None, ciudad=None):
        super().__init__(nombre, apellido, edad, pais, ciudad) # Si llamamos al constructor
        self.profesion = 'Estudiante'
    def saluda(self):
        # super().saluda() # Si no llamamos al método base, estamos anulando/sobreescribien
        print(f"Hola, soy {self.nombre}. Encantado de saludarte. Soy estudiante.")
    def datos(self):
        datos = super().datos()
        datos['profesion'] = 'Estudiante'
        return datos


persona1 = Estudiante("Pedro", apellido="Hernando", edad=54)
persona1.añadir_idioma('Inglés')
persona1.añadir_idioma('Español')
persona1.saluda()
print(persona1.datos())
```

Siguiente