Programación Orientada a Objetos (POO).

Curso de POO y algoritmos con Python.

Programación orientada a objetos en Python.

Uno de los elementos más importantes de los lenguajes de programación es la utilización de clases para organizar programas en módulos y abstracciones de datos.

Las clases se pueden utilizar de diversas maneras. Pero en este artículo hablaremos de cómo utilizarlas en el contexto de la programación orientada a objetos. La clave para entender la programación orientada a objetos es pensar en objetos como agrupaciones de datos y los métodos que operan en dichos datos.

Por ejemplo, podemos representar a una persona con propiedades como nombre, edad, género, etc. y los comportamientos de dicha persona como caminar, cantar, comer, etc. De la misma manera podemos representar unos audífonos con propiedades como su marca, tamaño, color, etc. y sus comportamientos como reproducir música, pausar y avanzar a la siguiente canción.

Puesto de otra manera, la programación orientada a objetos nos permite modelar cosas reales y concretas del mundo y sus relaciones con otros objetos.

Las ideas detrás de la programación orientada a objetos tienen más de 50 años y han sido ampliamente aceptadas y practicadas en los últimos treinta. A mediados de la década de los setenta se comenzaron a escribir artículos académicos explicando los beneficios de esta aproximación a la programación. También durante esos años se comenzaron a escribir los primeros lenguajes de programación que incorporaban estas ideas (como Smalltalk y CLU). Pero no fue hasta la llegada de Java y C++ que la idea consiguió un número importante de seguidores.

Hasta ahora, en el curso previo de esta serie hemos utilizado programación orientada a objetos de manera implícita. Cuando decimos “Los objetos son las principales cosas que un programa de Python manipula. Cada objeto tiene un tipo que define qué cosas puede realizar un programa con dicho objeto,” nos estamos refiriendo a las ideas principales de la programación orientada a objetos. Hemos utilizado los tipos lista y diccionario, entre muchos otros, así como los métodos asociados a dichos tipos.

Así como los creadores de un lenguaje de programación solo pueden diseñar una fracción muy pequeña de todas las funciones útiles (como `abs`, `float`, `type`, etc.), también pueden escribir una fracción muy pequeña de los tipos útiles (`int`, `str`, `dict`, `list`, etc.). Ya sabemos los mecanismos que nos permiten crear funciones, ahora veremos los mecanismos que nos permiten crear nuevos tipos (o clases).

Clases en Python.

Las estructuras primitivas con las que hemos trabajado hasta ahora nos permiten definir cosas sencillas, como el precio de algo, el nombre de un usuario, las veces que debe ejecutarse un bucle, etc. Sin embargo, existen ocasiones cuando necesitamos definir estructuras más complejas, por ejemplo, un hotel. Podríamos utilizar dos listas: una para definir los cuartos y una segunda para definir si el cuarto se encuentra ocupado o no.

```
cuartos_de_hotel = [101, 102, 103, 201, 202, 203]
cuarto_ocupado = [True, False, False, True, True, False]
```

Sin embargo, este tipo de organización rápidamente se sale de control. ¿Qué tal si quisiéramos añadir más propiedades, cómo si el cuarto ya fue limpiado o no? ¿Si el cuarto tiene cama doble o sencilla? Esto nos lleva a un problema de organización y es justamente esto lo que justifica la existencia de clases.

Las clases nos permiten crear nuevos tipos que contienen información arbitraria sobre un objeto. En el caso del hotel, podríamos crear dos clases `Hotel ()` y `Cuarto ()` que nos permitieran dar seguimiento a las propiedades como número de cuartos, ocupación, limpieza, tipo de habitación, etc.

Es importante resaltar que las clases sólo proveen estructura. Son un molde con el cual podemos construir objetos específicos. La clase señala las propiedades que los hoteles que modelemos tendrán, pero no es ningún hotel específico. Para eso necesitamos las instancias.

Instancias.

Mientras que las clases proveen la estructura, las instancias son los objetos reales que creamos en nuestro programa: un hotel llamado PlatziHotel o Hilton. Otra forma de pensarlo es que las clases son como un formulario y una vez que se llena cada copia del formulario tenemos las instancias que pertenecen a dicha clase. Cada copia puede tener datos distintos, al igual que cada instancia es distinta de las demás (aunque todas pertenezcan a una misma clase).

Para definir una clase, simplemente utilizamos la palabra reservada `class`. Por ejemplo:

```
class Hotel:
    pass
```

Una vez que tenemos una clase llamada `Hotel` podemos generar una instancia llamando al constructor de la clase.

```
hotel = Hotel()
```

Atributos de la instancia.

Todas las clases crean objetos y todos los objetos tienen atributos. Utilizamos el método especial `__init__` para definir el estado inicial de nuestra instancia. Recibe como primer parámetro obligatorio el `self` (que es simplemente una referencia a la propia instancia).

```
class Hotel:

    def __init__(self, numero_maximo_de_huespedes, lugares_de_estacionamiento):
        self.numero_maximo_de_huespedes = numero_maximo_de_huespedes
        self.lugares_de_estacionamiento = lugares_de_estacionamiento
        self.huespedes = 0


hotel = Hotel(numero_maximo_de_huespedes=50, lugares_de_estacionamiento=20)
print(hotel.lugares_de_estacionamiento) # 20
```

Métodos de instancia.

Mientras que los atributos de la instancia describen lo que representa el objeto, los métodos de instancia nos indican qué podemos hacer con las instancias de dicha clase y normalmente operan en los mencionados atributos. Los métodos son equivalentes a funciones pero declarados dentro de la clase y todos ellos reciben `self` como primer argumento.

```
class Hotel:

    ...

    def anadir_huespedes(self, cantidad_de_huespedes):
        self.huespedes += cantidad_de_huespedes

    def checkout(self, cantidad_de_huespedes):
        self.huespedes -= cantidad_de_huespedes

    def ocupacion_total(self):
        return self.huespedes


hotel = Hotel(50, 20)
hotel.anadir_huespedes(3)
hotel.checkout(1)
hotel.ocupacion_total() # 2
```