Instituto de formación técnica superior N° 18

Programación Aplicada - apuntes

Curso: 2° Año

Profesor: Bonini Juan Ignacio

Ciclo lectivo: 2023

Carga horaria: 5 hs cátedra/semana

Régimen: Cuatrimestral

Introducción a la programación orientada a objetos (POO)

El paradigma de programación orientada a objetos se centra en la creación y manipulación de objetos como bloques fundamentales para construir software. Con este enfoque, los programas se diseñan modelando entidades del mundo real o abstracto como objetos, que combinan datos (atributos) y comportamientos (métodos) que están estrechamente vinculados y forman un conjunto lógico y cohesivo de funcionalidades.

Cuando se diseñan clases y objetos en la programación orientada a objetos, se busca agrupar características y comportamientos relacionados en una misma entidad. Esto se hace para que los datos y las acciones que pertenecen juntos estén organizados de manera ordenada y se puedan manipular de manera conjunta.

La coherencia en el diseño de clases y objetos es importante para crear un código más legible, mantenible y comprensible. Cuando los elementos relacionados están agrupados de manera lógica, es más fácil entender cómo interactúan entre sí y qué responsabilidades tiene cada parte del código.

Clase

Una clase es una plantilla o un plano para crear objetos. Define las propiedades (atributos) y comportamientos (métodos) que los objetos creados a partir de esa clase tendrán. En otras palabras, una clase es una representación abstracta de un concepto, entidad o tipo de datos. Una clase y un objeto son conceptos fundamentales que permiten organizar y estructurar el código de manera más eficiente y modular. Las clases agrupan los atributos y métodos relacionados en una sola unidad, lo que fomenta la reutilización de código y la organización eficiente de la lógica del programa.

Objeto

Un objeto es una instancia concreta de una clase. Se crea utilizando la plantilla proporcionada por la clase y contiene valores específicos para los atributos definidos en la clase. Los objetos son las unidades básicas con las que trabajamos en la POO y pueden interactuar entre sí a través de métodos y propiedades.

Imaginemos que estamos construyendo una ciudad virtual en nuestra computadora. Una clase es como un plano o diseño para crear edificios similares. Tiene todas las instrucciones sobre cómo deben ser los edificios: cuántos pisos, qué colores, qué puertas y ventanas.

Ahora, un objeto es un edificio real hecho basándose en ese plano. Cada edificio puede ser único, pero sigue las reglas del plano de la clase. Por ejemplo, si la clase es "Edificio", los objetos podrían ser "Casa", "Oficina" o "Escuela", todos construidos a partir de la misma plantilla pero con detalles diferentes.

Atributos

Los atributos son las características o propiedades que definen el estado de un objeto. Representan datos asociados a un objeto y pueden ser variables que almacenan información específica. Los atributos son esenciales para describir las características únicas de cada instancia de una clase.

Supongamos que tenemos una clase "Auto". Los atributos de esta clase podrían ser "marca", "color" y "rodado". Estos atributos definen las características de un auto en el contexto del programa.

Propiedades

Las propiedades son métodos especiales que se utilizan para controlar el acceso y la modificación de los atributos de una clase. Permiten definir un comportamiento personalizado al obtener o establecer el valor de un atributo, en lugar de acceder directamente a él. Las propiedades son útiles para implementar lógica adicional al interactuar con los datos de un objeto.

Siguiendo el ejemplo previo, si deseamos transformar el atributo "color" de la clase Auto en una propiedad, el proceso se inicia renombrando la variable a "_color". Este enfoque de nomenclatura con un guión bajo indica que la variable es privada y no debe accederse directamente. Posteriormente, creamos la propiedad "color", que está diseñada para recuperar el color del auto. Sin embargo, esta propiedad no permitirá que el valor del color sea modificado directamente. En cambio, se proporciona una interfaz controlada para obtener información sobre el color del automóvil mientras se mantiene la integridad de los datos internos.

Métodos

Un método es una función definida dentro de una clase que opera en instancias de esa clase. Los métodos son acciones o comportamientos asociados a un objeto y se utilizan para realizar diversas operaciones y manipulaciones en los datos contenidos en ese objeto.

Un método en Python se declara dentro de una clase y generalmente toma self como su primer parámetro, que hace referencia a la instancia del objeto en sí mismo. Los métodos pueden acceder a los atributos de la instancia y realizar tareas específicas relacionadas con la clase.

Si crearamos el método "cambiar_color" en la clase "Auto" sería una función definida dentro de la clase que permitiría modificar el color del automóvil. Este método tomaría un nuevo color como parámetro y, después de validar si el color es válido, cambiaría el color actual del auto al nuevo color proporcionado.

El método "cambiar_color" en la clase "Auto" sería responsable de cambiar el color del automóvil a otro color válido en función de los criterios de validación establecidos.

Métodos de clase

Un método de clase en Python es una función definida dentro de una clase que opera en la misma clase, en lugar de en instancias individuales de la clase. Se denota como un método que lleva el decorador @classmethod justo encima de su definición. A diferencia de los métodos de instancia, los métodos de clase toman la clase como su primer argumento, generalmente llamado cls, en lugar de una instancia (self).

Los métodos de clase son útiles cuando se necesita realizar una operación que involucra a la clase en sí misma, en lugar de las instancias específicas de la clase. Un caso común de uso es cuando se necesita mantener datos o realizar operaciones que sean relevantes para la clase en general y no para instancias individuales.

Métodos estáticos

Un método estático en Python es una función definida dentro de una clase que no tiene acceso a las instancias de la clase ni a sus atributos. Se denota como un método que lleva el decorador @staticmethod justo encima de su definición. A diferencia de los métodos de instancia y de clase, los métodos estáticos no reciben automáticamente ningún argumento especial relacionado con la instancia o la clase.

Los métodos estáticos son útiles cuando se necesita encapsular una funcionalidad que está relacionada con la clase, pero no depende de los atributos de instancia o de la clase. No tienen acceso a los atributos de instancia ni pueden modificar el estado de la instancia. Se utilizan principalmente para agrupar funciones que están relacionadas con la clase, pero que no necesitan acceder a información específica de instancia.

Decoradores

Los decoradores en Python son funciones especiales que se utilizan para modificar o extender el comportamiento de otras funciones o métodos. Los decoradores permiten agregar funcionalidades adicionales a una función sin cambiar su código interno. Se aplican utilizando la sintaxis @nombre_del_decorador justo antes de la definición de una función.

Un decorador es esencialmente una función que toma otra función como argumento, realiza alguna operación y devuelve una nueva función con el comportamiento modificado. Esto permite encapsular lógica repetitiva o tareas comunes que se deben realizar antes o después de la ejecución de una función.

```python
class Auto:
    """Esta es una clase que representa un automóvil."""
    COLORES_VALIDOS: tuple = ("rojo", "verde", "azul")

    def __init__(self, marca: str, color: str, rodado: int):
        """Inicializa un objeto de la clase Auto."""
        self.marca = marca
        self._color = color
        self.rodado = rodado

    def cambiar_color(self, color: str) -> bool:
        """Cambia el color del automóvil si es un color válido."""
        se_cambio_el_color = False
        if self.validar_color(color):
            self._color = color
            se_cambio_el_color = True
        return se_cambio_el_color

    @classmethod
    def validar_color(cls, color: str) -> bool:
        """ si un color dado es válido.
        return color in cls.COLORES_VALIDOS

    @property
    def color(self) -> str:
        """Obtiene el color actual del automóvil."""
        return self._color

    @staticmethod
    def metodo_estatico():
        """Este es un método estático que no realiza ninguna acción."""
        pass
```

El código anterior define una clase llamada "Auto" que representa un automóvil. La clase tiene atributos como "marca", "color" y "rodado". Además, incluye métodos para cambiar el color del automóvil, validar si un color es válido, obtener el color actual y un método estático a modo ejemplo que no tiene implementación.

Ejercicios prácticos de clases

1. Crea una clase llamada "Estudiante" con los siguientes atributos: nombre, edad y _promedio. Implementa tres métodos: mostrar_informacion() para mostrar los detalles del estudiante, actualizar_promedio(nuevo_promedio) para cambiar el promedio del estudiante y incrementar_edad() para aumentar en 1 la edad del estudiante cada vez que se llama. Además, crea una propiedad llamada promedio para acceder y modificar el promedio de manera controlada.
2. Crea una clase llamada "Persona" con los atributos nombre, edad y altura. Implementa métodos para mostrar información, incrementar la edad y cambiar la altura.
3. Crea una clase llamada "Perro" con los atributos nombre, raza y edad. Crea métodos para mostrar información, cambiar la raza y aumentar la edad.
4. Crea una clase llamada "Cuenta" con los atributos titular, saldo y _numero_cuenta. Crea métodos para mostrar información, depositar y retirar dinero.
5. Crea una clase llamada "Libro" con los atributos titulo, autor y _paginas. Implementa métodos para mostrar información, cambiar el autor y agregar páginas.