

Programación Orientada a Objetos (POO)

Propiedades

Las propiedades permiten controlar cómo se accede y modifica la información dentro de una clase. Así, se asegura de que los datos siempre se mantengan correctos y no se cambien de manera inapropiada.

- Usar propiedades es como usar atributos normales (como `objeto.nombre`). Esto hace que el código sea más fácil de leer y trabajar.
- Puedes añadir reglas para verificar que los datos sean correctos antes de guardarlos. Por ejemplo, asegurarte de que la edad de una persona siempre sea un número positivo.
- Las propiedades hacen que el código sea más limpio y más fácil de entender, ya que se accede a los datos de forma más directa y natural.

Para usar propiedades en Python, se utilizan los decoradores `@property` y `@<nombre>.setter`. La sintaxis básica y los pasos para definir y usar propiedades en una clase es la siguiente:

1. Definir la Propiedad: Primero, se debe definir un método de instancia que actuará como la propiedad. Se debe usar el decorador `@property` para indicar que este método debe ser tratado como una propiedad. Esto convierte el método en una propiedad de solo lectura.
2. Definir el Setter (opcional): Si se necesita permitir que la propiedad sea modificable, se define un método adicional con el decorador `@<nombre>.setter`, donde `<nombre>` es el nombre de la propiedad. Esto permitirá definir cómo se asignan los valores a la propiedad.

Ejemplo: Uso de propiedades

Clase Triangulo

- Esta clase modela un triángulo con atributos de base y altura, y calcula su área.

Atributos Privados

- `__base`: Atributo privado para almacenar la base del triángulo.

- `__altura`: Atributo privado para almacenar la altura del triángulo.

Métodos property

- **@property para base:**
 - **Propósito:** Permite acceder al valor de `__base` como si fuera un atributo público.
 - **Getter:** Devuelve el valor actual de `__base`.
- **@base.setter:**
 - **Propósito:** Permite modificar el valor de `__base`.
 - **Setter:** Establece el valor de `__base` solo si es positivo. Si el valor es menor o igual a cero, lanza una excepción `ValueError`.
- **@property para altura:**
 - **Propósito:** Permite acceder al valor de `__altura` como si fuera un atributo público.
 - **Getter:** Devuelve el valor actual de `__altura`.
- **@altura.setter:**
 - **Propósito:** Permite modificar el valor de `__altura`.
 - **Setter:** Establece el valor de `__altura` solo si es positivo. Si el valor es menor o igual a cero, lanza una excepción `ValueError`.
- **@property para area:**
 - **Propósito:** Calcula y devuelve el área del triángulo utilizando los valores actuales de `base` y `altura`.
 - **Getter:** Devuelve el resultado de la fórmula del área del triángulo: `base * altura / 2`.

```
class Triangulo:
    def __init__(self, base, altura):
        self.__base = base
        self.__altura = altura

    @property
    def base(self):
        return self.__base

    @base.setter
    def base(self, valor):
        if valor <= 0:
            raise ValueError("La base debe ser un valor positivo")
        self.__base = valor
```

```

@property
def altura(self):
    return self.__altura

@altura.setter
def altura(self, valor):
    if valor <= 0:
        raise ValueError("La altura debe ser un valor positivo")
    self.__altura = valor

@property
def area(self):
    return self.base * self.altura / 2

```

Si intentamos construir un objeto con un valor inválido, nos dará error

```
tri = Triangulo(10, -5)
```

```
NameError: name 'Triangulo' is not defined
```

```
[1;31m-----[0m
```

```
[1;31mNameError[0m                                Traceback (most recent call last)
```

```
Cell [1;32mIn[9], line 1[0m
```

```
[1;32m----> 1[0m tri [38;5;241m=[39m Triangulo([38;5;241m10[39m, [38;5;241m-[39m[38;5;241m5[39m
```

```
[1;31mNameError[0m: name 'Triangulo' is not defined
```