

Programación Orientada a Objetos (POO)

Encapsulación

El encapsulamiento es un principio fundamental en la programación orientada a objetos (POO) que consiste en agrupar datos (atributos) y el comportamiento (métodos) que opera sobre esos datos dentro de una misma unidad, llamada clase.

El siguiente código muestra cómo definir una clase en Python con atributos que se pueden modificar después de la creación de una instancia.

1. Definición de la Clase Triangulo:

- La clase `Triangulo` incluye un constructor `__init__` para inicializar los atributos `base` y `altura`.

2. Creación de una Instancia y Modificación de Atributos:

- Se crea una instancia de `Triangulo` y se imprimen los valores de `base` y `altura`.
- Luego, se modifican los atributos `base` y `altura` de la instancia y se imprimen nuevamente.

Explicación

• Constructor `__init__`:

- El constructor inicializa los atributos `base` y `altura` con los valores proporcionados al crear una instancia de `Triangulo`.

• Modificación de Atributos:

- Después de crear la instancia `tri` con `base` igual a 1 y `altura` igual a 2, los valores de los atributos se imprimen.
- Los atributos `base` y `altura` se modifican directamente asignando nuevos valores (-3 para `base` y 5 para `altura`).
- Los nuevos valores de los atributos se imprimen después de la modificación.

```

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

tri = Triangulo(1, 2)
print(f"La base es: {tri.base}")
print(f"La altura es: {tri.altura}")
tri.base = -3
tri.altura = 5
print(f"La base es: {tri.base}")
print(f"La altura es: {tri.altura}")

```

```

La base es: 1
La altura es: 2
La base es: -3
La altura es: 5

```

En el siguiente código, se están utilizando nombres de atributos con doble guion bajo (`__base` y `__altura`). En Python, esto activa el *name mangling*, que cambia el nombre del atributo para hacerlo más difícil de acceder desde fuera de la clase.

En el siguiente código se muestra cómo usar el *name mangling* para encapsular atributos en una clase en Python.

1. Definición de la Clase Triangulo con Atributos Encapsulados:

- Los atributos `__base` y `__altura` están precedidos por dos guiones bajos, lo que indica que son privados.

2. Intento de Acceso a Atributos Privados:

- Se crea una instancia de `Triangulo` con `base` igual a 1 y `altura` igual a 2.
- Se intenta acceder al atributo privado `__base` directamente desde fuera de la clase.

Explicación

• Name Mangling:

- Los atributos que comienzan con dos guiones bajos (`__base`, `__altura`) están diseñados para ser privados y no deben ser accesibles directamente desde fuera de la clase. Python realiza un *name mangling*, que cambia el nombre del atributo para incluir el nombre de la clase, haciendo más difícil el acceso directo.
- Por ejemplo, el atributo `__base` se convierte en `_Triangulo__base`.

- **Acceso a Atributos Privados:**

- Al intentar acceder a `tri.__base` directamente, se produce un `AttributeError` porque el atributo está encapsulado.
- Para acceder a estos atributos, deberías usar métodos públicos de la clase, como getters y setters, para proporcionar una interfaz segura.

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

tri = Triangulo(1, 2)
print(tri.__base)
```

```
AttributeError: 'Triangulo' object has no attribute '__base'
[1;31m-----[0m
[1;31mAttributeError[0m                                Traceback (most recent call last)
Cell [1;32mIn[72], line 7[0m
[0;32m      4[0m          [38;5;28mself[39m[38;5;241m.[39m__altura [38;5;241m=[39m altura
[0;32m      6[0m tri [38;5;241m=[39m Triangulo([38;5;241m1[39m, [38;5;241m2[39m)
[1;32m----> 7[0m [38;5;28mprint[39m(tri[38;5;241m.[39m__base)

[1;31mAttributeError[0m: 'Triangulo' object has no attribute '__base'
```

Python cambia el nombre del atributo `__base` a `_Triangulo__base` y el nombre del atributo `__altura` a `_Triangulo__altura`. Esto se hace para evitar conflictos de nombres en clases derivadas y para mantener la encapsulación. Por lo tanto, podemos acceder a estos atributos de la siguiente manera:

```
print(tri._Triangulo__base)
print(tri._Triangulo__altura)
```

1
2