**SobreCarga de Operadores en Python**

La palabra clave **operator** declara una función especificando el significado del *símbolo de operador* cuando se aplica a las instancias de una clase. Esto proporciona al operador más de un significado, o lo "sobrecarga". El compilador distingue entre los diferentes significados de un operador examinando los tipos de sus operandos.

<https://learn.microsoft.com/es-es/cpp/cpp/operator-overloading?view=msvc-170>

La sobrecarga de operadores en Python permite (extender, modificar y/o redefinir) el comportamiento de los operadores cuando se aplican a objetos de una clase que definimos nosotros.

+, -, \*, ==, etc, operadores que utilizamos para hacer operacinoes comunes en python.

Por ejemplo, si tenemos una clase llamada Punto para representar coordenadas, podemos usar la **sobrecarga de operadores** para que el operador “+” combine dos objetos de tipo Punto y devuelva otro Punto con la suma de sus coordenadas:

class Punto:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, otro):

return Punto(self.x + otro.x, self.y + otro.y)

p1 = Punto(1, 2)

p2 = Punto(3, 4)

resultado = p1 + p2

En este ejemplo se llama a \_\_add\_\_ y devolverá un nuevo Punto(4, 6)

Aquí, cuando escribimos p1 + p2, en lugar de dar un error, Python entiende que debe sumar p1 y p2 como puntos y devuelve un nuevo punto con las coordenadas sumadas. La sobrecarga hace que el uso de los objetos sea más intuitivo.

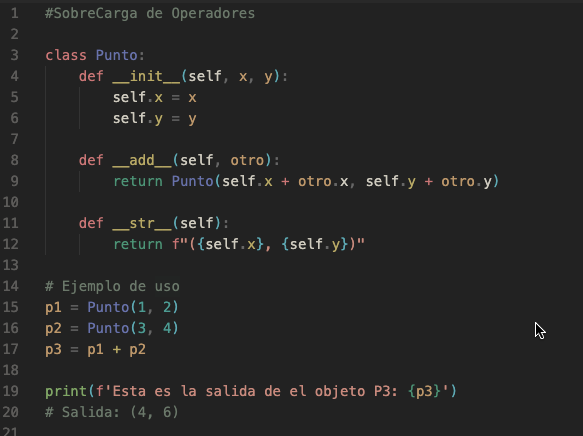
La funcion \_\_add\_\_ en pyhton es un metodo que devuelve un nuevo objeto

<https://www.geeksforgeeks.org/python-__add__-magic-method/>

En Python, la sobrecarga de operadores específicamente requiere el uso de métodos especiales (como \_\_add\_\_, \_\_sub\_\_, \_\_mul\_\_, etc.) que Python interpreta para cada operador.

Sin estos métodos especiales, Python no puede reconocer que el operador debe tener un comportamiento personalizado para los objetos de la clase.

* **\_\_add\_\_(self, other)**: Se llama cuando se usa +.
* **\_\_sub\_\_(self, other)**: Se llama cuando se usa -.
* **\_\_mul\_\_(self, other)**: Se llama cuando se usa \*.
* **\_\_truediv\_\_(self, other)**: Se llama cuando se usa / (división real).
* **\_\_floordiv\_\_(self, other)**: Se llama cuando se usa // (división entera).
* **\_\_mod\_\_(self, other)**: Se llama cuando se usa % (módulo).
* **\_\_pow\_\_(self, other)**: Se llama cuando se usa \*\* (potencia).
* Reutilizar código
* Aumentar la claridad del código
* Reducir la complejidad del código
* Hacer que el código sea más conciso y fácil de entender
* Comparar directamente los objetos de las clases
* Realizar operaciones aritméticas con los objetos de las clases



Imagina que tienes una clase para representar un **Punto** en el espacio 2D (con coordenadas x e y). Gracias a la sobrecarga de operadores, podemos hacer que el operador + sume dos puntos, en lugar de tener que escribir una función específica para eso.

Las **funciones mágicas** en Python, también conocidas como *métodos especiales* o *métodos dunder* (porque comienzan y terminan con doble guion bajo, como \_\_init\_\_), permiten que los objetos de una clase se comporten de forma especial. Estas funciones hacen que los objetos puedan usar operadores (+, -, \*), o responder a funciones integradas como len(), str(), y más. Usarlas es sencillo una vez que sabes cuáles necesitas y cómo implementarlas.

El metodo \_\_init\_\_ :

class Persona:

def \_\_init\_\_(self, nombre):

self.nombre = nombre

El metodo \_\_str\_\_ :

class Persona:

def \_\_init\_\_(self, nombre):

self.nombre = nombre

def \_\_str\_\_(self):

return f"Persona llamada {self.nombre}"

p = Persona("Ana")

print(p) # Muestra: Persona llamada Ana

El metodo \_\_len\_\_ :

class Grupo:

def \_\_init\_\_(self, miembros):

self.miembros = miembros

def \_\_len\_\_(self):

return len(self.miembros)

grupo = Grupo(["Ana", "Luis", "Carlos"])

print(len(grupo)) # Muestra: 3

El metodo \_\_add\_\_ :

class Numero:

def \_\_init\_\_(self, valor):

self.valor = valor

def \_\_add\_\_(self, otro):

return Numero(self.valor + otro.valor)

n1 = Numero(10)

n2 = Numero(20)

suma = n1 + n2

print(suma.valor) # Muestra: 30

**Uso básico**

Para usar estas funciones mágicas:

1. Define la clase.
2. Agrega el método mágico (como \_\_str\_\_, \_\_add\_\_, etc.) dentro de la clase.
3. Luego, puedes usar el operador o función como si estuviera en un objeto común.

**Consejos**

* No tienes que definir todas las funciones mágicas; solo las que necesites.
* Las funciones mágicas hacen que el código sea más limpio e intuitivo, ya que puedes usar operadores y funciones integradas en Python directamente.

