Day 44

Operator Overloading in Python:

Why Operator Overloading?

Built-in types already overload operators:

```
print(2 + 3)       # 5
print("Hi" + "!")  # Hi!
print([1, 2] + [3]) # [1, 2, 3]
```

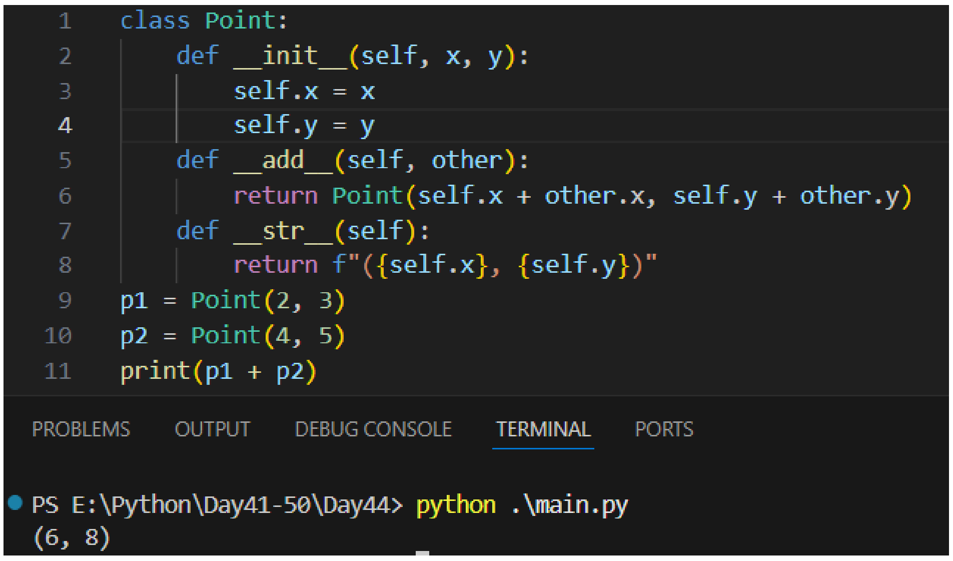
Same + operator, different behaviour. Operator overloading lets your own objects behave like built-in types.

Basic Idea

Operators call special methods internally.

Operator	Method
+	__add__()
-	__sub__()
*	__mul__()
/	__truediv__()
==	__eq__()
<	__lt__()
>	__gt__()

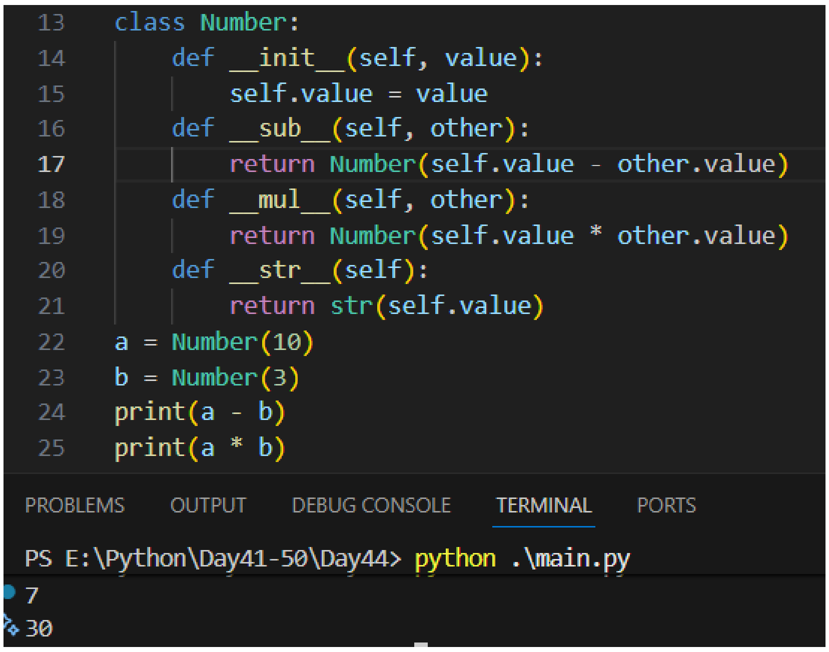
Example: overloading the + operator.



```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
    def __str__(self):
        return f"({self.x}, {self.y})"
p1 = Point(2, 3)
p2 = Point(4, 5)
print(p1 + p2)
```

```
PS E:\Python\Day41-50\Day44> python .\main.py
(6, 8)
```

Example: - and *



```python
class Number:
    def __init__(self, value):
        self.value = value
    def __sub__(self, other):
        return Number(self.value - other.value)
    def __mul__(self, other):
        return Number(self.value * other.value)
    def __str__(self):
        return str(self.value)
a = Number(10)
b = Number(3)
print(a - b)
print(a * b)
```

```
PS E:\Python\Day41-50\Day44> python .\main.py
7
30
```

Summary

- Operator overloading lets objects behave like built-in types
- Achieved using magic methods
- Makes code cleaner and more expressive
- Should be used thoughtfully

--The End--