### Example 1: Complex Numbers

In Python, the "+" operator for adding two objects is associated with the \_\_add\_\_ method. Let's create a simple Complex class and overload the "+" operator:

class ComplexNumbers:

def \_\_init\_\_(self,r,i):

self.real=r

self.imaginary=i

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

new\_real=self.real+other.real

new\_imaginary=self.imaginary+other.imaginary

return f"{new\_real} + {new\_imaginary}i"

def \_\_mul\_\_(self,other):

new\_real=self.real\*other.real

new\_imaginary=self.imaginary\*other.imaginary

return f"{new\_real} \* {new\_imaginary}i"

c1=ComplexNumbers(2,9)

c2=ComplexNumbers(5,8)

print(c1+c2)

print(c1\*c2)

In this example, the \_\_add\_\_ method is called when you use the "+" operator with two Complex instances. It specifies how to add the real and imaginary parts of the two complex numbers, and the result is a new Complex instance representing the sum.

### Example 2: Person Class

Now, let's create a Person class and overload the ">" operator to compare who pays the bill:

class Person:

def \_\_init\_\_(self, name, money):

self.name = name

self.money = money

def \_\_gt\_\_(self, other\_person):

# Define how to compare two persons based on their money

return self.money > other\_person.money

# Create two persons

person1 = Person("Alice", 100)

person2 = Person("Bob", 150)

# Use the ">" operator to compare who has more money

if person1 > person2:

print(f"{person1.name} will pay the bill.")

else:

print(f"{person2.name} will pay the bill.")

In this example, the \_\_gt\_\_ method is called when you use the ">" operator with two Person instances. It specifies how to compare persons based on their money, and the result determines who will pay the bill.

Operator overloading allows you to use familiar operators in a way that makes sense for your custom classes, making your code more intuitive and expressive.