Chapter 6 - Oprator Overlodading

Theory:

This chapter covers the topic of Oprator Overlodading.

Code Example:

```
# What is Operator Overloading?
# Jab hum Python ke built-in operators (jaise +, -, *, ==)
# ko apni class ke objects ke liye customize karte hain,
# use operator overloading kehte hain.
class Box:
def __init__(self, weight = 0):
self.weight = weight
self.box = [10, 20, 30, 40]
def __add__(self, other):
return Box(self.weight + other.weight) # ■ +
def __sub__(self, other):
return Box(self.weight - other.weight) # ■ -
def __mul__(self, other):
return Box(self.weight * other.weight) # ■ *
def __truediv__(self, other):
return Box(self.weight / other.weight) # ■ /
def __eq__(self, other):
return self.weight == other.weight # ■ ==
def __lt__(self, other):
return self.weight < other.weight # ■ <
def __gt__(self, other):
return self.weight > other.weight # ■ >
def __str__(self):
return f"Box with weight: {self.weight}kg" # ■ print()
def __len__(self):
return len(self.box) # ■ Returns an integer — required by __len__
b1 = Box(10)
b2 = Box(5)
b = Box()
print("Add (+):", b1 + b2) # → 15
print("Sub (-):", b1 - b2) # → 5
print("Mul (*):", b1 * b2) # → 50
print("Div (/):", b1 / b2) # → 2.0
print("Equal (==):", b1 == b2) # → False
print("Less than (<):", b1 < b2) # → False
print("Greater (>):", b1 > b2) # → True
print("Length of box:", len(b))
#mycode
class Number:
def __init__(self , n):
self.n = n
def __add__(self, num):
return self.n + num.n
def __sub__(self , num):
return self.n - num.n
n = Number(int(input("Enter a number : ")))
m = Number(int(input("Enter a number : ")))
print(n+m)
```

```
print(n-m)
```