



TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

AN ASSIGNMENT ON
Operator Overloading in Python

SUBMITTED BY:
SURENDEA TIRUWA (081BEL091)

SUBMITTED TO:
PRAKASH CHANDRA PRASAD SIR,
DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

OPERATOR OVERLOADING IN PYTHON

Objective:

The objective of this assignment is to implement operator overloading for all arithmetic and relational operators in Python.

Theory:

Operator overloading in Python allows developers to redefine the behavior of built-in operators for user-defined classes. This enables intuitive interaction with objects of custom classes using standard operators like +, -, *, /, ==, <, etc.

Code:

```
class Number:
    def __init__(self, value):
        self.value = value

    # Arithmetic operators
    def __add__(self, other):
        return Number(self.value + other.value)

    def __sub__(self, other):
        return Number(self.value - other.value)

    def __mul__(self, other):
        return Number(self.value * other.value)

    def __truediv__(self, other):
        return Number(self.value / other.value)

    def __floordiv__(self, other):
        return Number(self.value // other.value)

    def __mod__(self, other):
        return Number(self.value % other.value)

    def __pow__(self, other):
        return Number(self.value ** other.value)
```

```
# Relational operators
def __eq__(self, other):
    return self.value == other.value

def __ne__(self, other):
    return self.value != other.value

def __lt__(self, other):
    return self.value < other.value

def __le__(self, other):
    return self.value <= other.value

def __gt__(self, other):
    return self.value > other.value

def __ge__(self, other):
    return self.value >= other.value

def __str__(self):
    return str(self.value)
```

```
# Example Usage
```

```
a = Number(10)
```

```
b = Number(5)
```

```
print("Addition:", a + b)
```

```
print("Subtraction:", a - b)
```

```
print("Multiplication:", a * b)
```

```
print("Division:", a / b)
```

```
print("Floor Division:", a // b)
```

```
print("Modulus:", a % b)
```

```
print("Power:", a ** b)
```

```
print("Equal:", a == b)
```

```
print("Not Equal:", a != b)
```

```
print("Less Than:", a < b)
```

```
print("Less Than or Equal:", a <= b)
```

```
print("Greater Than:", a > b)
```

```
print("Greater Than or Equal:", a >= b)
```

Output:

```
Addition: 15  
Subtraction: 5  
Multiplication: 50  
Division: 2.0  
Floor Division: 2  
Modulus: 0  
Power: 100000  
Equal: False  
Not Equal: True  
Less Than: False  
Less Than or Equal: False  
Greater Than: True  
Greater Than or Equal: True
```

Conclusion:

In this assignment, operator overloading was demonstrated for all arithmetic and relational operators in Python. This allows user-defined classes to interact with built-in operators seamlessly, enhancing code readability and flexibility.