In [1]:
```
a = '10'
b = '10'
print(a + b)
print(a.__add__(b))
```

```
1010
1010
```

In [2]:
```
# Python program to show use of + operator for different purposes.

print(1 + 2)

# concatenate two strings
print("Geeks"+"For")

# Product two numbers
print(3 * 4)

# Repeat the String
print("Geeks"*4)
```

```
3
GeeksFor
12
GeeksGeeksGeeksGeeks
```

In [7]:
```
class Skul:
    def __init__(self,math, physics):
        self.math = math
        self.physics = physics

    def __gt__(self, other):
        student1 = self.math + self.physics
        student2 = other.math + other.physics
        if student1 > student2:
            return True
        else:
            return False

    def __add__(self, other):
        math = self.math +self.physics
        physics =  other.math  + other.physics
        return (math, physics)

s1 = Skul(90, 95)
s2 = Skul(92, 92)
print(s1 + s2)

if s1>s2:
    print("Student 1 wins")
else:
    print("Student 2 wins")
```

```
(185, 184)
Student 1 wins
```

In [23]:
```
# Python Program illustrate how to overload an binary + operator

class A:
    def __init__(self, a):
        self.a = a

    # adding two objects
    def __add__(self, o):
        print(self.a)
        print(o.a)
        return self.a + o.a

ob1 = A(1)
ob2 = A(2)

ob3 = A("Geeks")
ob4 = A("For")

print('sum : ' + str(ob1 + ob2))
print('\n')
print('concate : ' + str(ob3 + ob4))
```

```
1
2
sum : 3


Geeks
For
concate : GeeksFor
```

In [26]:
```
# Python Program to perform addition of two complex numbers using binary + operator overloading.

class complex:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    # adding two objects
    def __add__(self, other):
        return self.a + other.a, self.b + other.b

Ob1 = complex(1, 2)
Ob2 = complex(3, 4)

Ob3 = Ob1 + Ob2
print(Ob3)
```

```
(4, 6)
```

In [33]:
```
# Python program to overload equality and less than operators

class A:
    def __init__(self, a):
        self.a = a
    def __lt__(self, other):
        if(self.a<other.a):
            return "ob1 is less than ob2"
        else:
            return "ob2 is less than ob1"
    def __eq__(self, other):
        if(self.a == other.a):
            return "Both are equal"
        else:
            return "Not equal"

ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)
print(ob1 > ob2)

print('\n')

ob3 = A(4)
ob4 = A(4)
print(ob1 == ob2)
print(ob3 == ob4)
```

```
ob1 is less than ob2
ob2 is less than ob1


Not equal
Both are equal
```

In [ ]:
```
# Operator      Magic Method

# +             __add__(self, other)
# -             __sub__(self, other)
# *             __mul__(self, other)
# /             __truediv__(self, other)
# //    __floordiv__(self, other)
# %             __mod__(self, other)
# **    __pow__(self, other)
# >>    __rshift__(self, other)
# <<    __lshift__(self, other)
# &             __and__(self, other)
# |             __or__(self, other)
# ^             __xor__(self, other)

# Comparison Operators :
# Operator      Magic Method
# <             __LT__(SELF, OTHER)
# >             __GT__(SELF, OTHER)
# <=    __LE__(SELF, OTHER)
# >=    __GE__(SELF, OTHER)
# ==    __EQ__(SELF, OTHER)
# !=    __NE__(SELF, OTHER)

# Assignment Operators :
# Operator      Magic Method
# -=    __ISUB__(SELF, OTHER)
# +=    __IADD__(SELF, OTHER)
# *=    __IMUL__(SELF, OTHER)
# /=    __IDIV__(SELF, OTHER)
# //=   __IFLOORDIV__(SELF, OTHER)
# %=    __IMOD__(SELF, OTHER)
# **=   __IPOW__(SELF, OTHER)
# >>=   __IRSHIFT__(SELF, OTHER)
# <<=   __ILSHIFT__(SELF, OTHER)
# &=    __IAND__(SELF, OTHER)
# |=    __IOR__(SELF, OTHER)
# ^=    __IXOR__(SELF, OTHER)

# Unary Operators :
# Operator      Magic Method
# -             __NEG__(SELF, OTHER)
# +             __POS__(SELF, OTHER)
# ~             __INVERT__(SELF, OTHER)
```