Operator Overloading

- Operator Overloading is where operators **behave differently** in different situations
- Operator Overloading is possible in normal DataType also
- Using operators we can **achieve polymorphism**
- Lets see this with an example
- We can add 2 rational numbers using operator overloading

```
class Rational:
    def __init__(self, p=1, q=1):
        self.p=p
        self.q=q

    def __add__(self, other):
        s = Rational()
        s.p = self.p * other.q + self.q * other.p
        s.q = self.q * other.q
        return s

r1 = Rational(2,3)
r2 = Rational(2,5)

sum = r1+r2
print(sum.p, sum.q)
```

we are writing a constructor and method through this way we add 2 rational numbers

```
16 15

Process finished with exit code 0
```

- You can also overload for **subtraction , multiplication , power , division** etc.

Binary Operators	
__add__()	Addition
__sub__()	Subtraction
__mul__()	Multiplication
__pow__()	Power
__truediv__()	Division
__floordiv__()	Floor Division
__mod__()	Remainder (modulo)
__lshift__()	Bitwise Left Shift
__rshift__()	Bitwise Right Shift
__and__()	Bitwise AND
__or__()	Bitwise OR
__xor__()	Bitwise XOR

- Operator Overloading also works well for **comparison operators and assignment operators.**

Assignment Operators	
__iadd__()	equivalent to a += b
__isub__()	equivalent to a -= b
__imul__()	equivalent to a *= b
__idiv__()	equivalent to a /= b
__ifloordiv__()	equivalent to a //= b
__imod__()	equivalent to a %= b
__ipow__()	equivalent to a **= b
__ilshift__()	equivalent to a <<= b
__irshift__()	equivalent to a >>= b
__iand__()	equivalent to a &= b
__ior__()	equivalent to a |= b
__ixor__()	equivalent to a ^= b

Comparison Operators	
__lt__()	Less than
__le__()	Less than or equal to
__eq__()	Equal to
__ne__()	Not equal to
__gt__()	Greater than
__ge__()	Greater than or equal to