

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

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

Assignment Operators	
<code>__iadd__()</code>	equivalent to <code>a += b</code>
<code>__isub__()</code>	equivalent to <code>a -= b</code>

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