Operator Overloading

Operator Overloading

- می توان عملکرد بعضی عملگرها را با توجه به عملوندهایش تغییر داد. به این کار operator overloading گفته می شود.
- بعنوان مثال عملگر + اگر عملوندهایش عدد باشند، مفهوم جمع را پیاده سازی می کند و اگر رشته باشند مفهوم concatenation را پیاده سازی می کند.
- مثال

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

>>> p1 = Point(2,3)
>>> p2 = Point(-1,2)
>>> p1 + p2
Traceback (most recent call last):
...
TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

Special Function (توابع ویژه)

- به توابع کلاس که با __ شروع می شوند، توابع ویژه گفته می شود.
- بعنوان مثال __init__() که بعنوان سازنده مورد استفاده قرار می گیرد، یک تابع ویژه می باشد.
- مثال

```
>>> p1 = Point(2,3)
>>> print(p1)
<__main__.Point object at 0x00000000031F8CC0>
```

Special Function (توابع ویژه) - ادامه

- می توان با استفاده از تابع ویژه __str__() خروجی بهتری ارائه کرد.

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
```

```
>>> p1 = Point(2,3)
>>> print(p1)
(2,3)
```

- بنابراین زمانیکه از توابع str(p1) و یا format(p1) استفاده می کنیم، در واقع __str__() p1. اجرا می شود.

سربارگزاری عملگر +

- برای سربارگزاری عملگر + ما نیاز به پیاده سازی تابع ویژه __add__() داریم.

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x,y)

>>> p1 = Point(2,3)
>>> p2 = Point(-1,2)
>>> print(p1 + p2)
(1,5)
```

توابع ویژه Operator Overloading

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

سربارگزاری عملگر مقایسه ای <

- برای سربارگزاری عملگر < ما نیاز به پیاده سازی تابع ویژه __lt__() داریم.

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

    def __lt__(self,other):
        self_mag = (self.x ** 2) + (self.y ** 2)
        other_mag = (other.x ** 2) + (other.y ** 2)
        return self_mag < other_mag
```

```
>>> Point(1,1) < Point(-2,-3)
True
>>> Point(1,1) < Point(0.5,-0.2)
False
>>> Point(1,1) < Point(1,1)
False
```

توابع ویژه Operator Overloading

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)