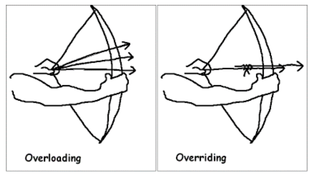
Operator and Method Overloading



Operator Overloading berarti memberikan makna yang diperluas di luar makna operasional yang telah ditentukan sebelumnya. Misalnya operator + digunakan untuk menambahkan dua bilangan bulat serta menggabungkan dua string dan menggabungkan dua daftar. Ini dapat dicapai karena operator '+' kelebihan beban oleh kelas int dan str. Anda mungkin telah memperhatikan bahwa operator atau fungsi bawaan yang sama menunjukkan perilaku yang berbeda untuk objek dari kelas yang berbeda, ini disebut Operator Overloading.

In [1]:
```
def product(a, b):
    p = a * b
    print(p)
def product(a, b, c):
    p = a * b * c
    print(p)
product(4, 5, 5)
```

```
100
```

Dalam kode di atas telah menetapkan dua metode produk, tetapi hanya dapat menggunakan metode produk kedua, karena python tidak mendukung metode kelebihan muatan. Anda dapat mendefinisikan banyak metode dengan nama yang sama dan argumen yang berbeda, tetapi saya hanya dapat menggunakan metode yang didefinisikan terbaru. Memanggil metode lain akan menghasilkan kesalahan. Seperti di sini, memanggil produk (4, 5) akan menghasilkan kesalahan karena metode produk yang didefinisikan terbaru membutuhkan tiga argumen.

Namun,Anda dapat menggunakan implementasi lain dalam python untuk membuat fungsi yang sama bekerja secara berbeda yaitu sesuai argumen.

In [2]:
```
def add(datatype, *args):
    if datatype == 'int':
        answer = 0
    if datatype == 'str':
        answer = ''
    for x in args:
        answer = answer + x
    print(answer)
add('int', 5, 6)
add('str', 'Hi ', 'Geeks')
```

```
11
Hi Geeks
```

In [3]:
```
class Human:
    def sayHello(self, name=None):
        if name is not None:
            print('Hello ' + name)
        else:
            print('Hello ')
obj = Human()
obj.sayHello()
obj.sayHello('Guido')
```

```
Hello
Hello Guido
```

Objek dibuat berdasarkan kelas, dan saya memanggil metode menggunakan nol dan satu parameter. Untuk mengklarifikasi metode overloading, sekarang kita dapat memanggil metode sayHello () dengan dua cara: obj.sayHello () obj.sayHello ('Guido')

Magic Method

Magic Method merupakan sebuah metode untuk menggunakan operator dalam overloading.

Binary Operators

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__truediv__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)
^	object.__xor__(self, other)
|	object.__or__(self, other)

Unary Operators

Operator	Method
-	object.__neg__(self)
+	object.__pos__(self)
abs()	object.__abs__(self)
~	object.__invert__(self)
complex()	object.__complex__(self)
int()	object.__int__(self)
long()	object.__long__(self)
float()	object.__float__(self)
oct()	object.__oct__(self)
hex()	object.__hex__(self

Comparison Operators

Operator	Method
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)

Extended Assignments

Operator	Method
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other)
**=	object.__ipow__(self, other[, modulo])
<<=	object.__ilshift__(self, other)
>>=	object.__irshift__(self, other)
&=	object.__iand__(self, other)
^=	object.__ixor__(self, other)
|=	object.__ior__(self, other)

In [5]:
```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def __str__(self):
        return "Point object is at: (" + str(self.x) + ", " + str(self.y) + ")"
    def __sub__(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)
```

```
Point object is at: (3, 1)
```

In [1]:
```
class Fraction:
    def __init__(self,top,bottom):
        self.num = top
        self.den = bottom
    def __str__(self):
        return str(self.num)+"/"+str(self.den)
    def __add__(self,otherfraction):
        newnum = self.num*otherfraction.den + self.den*otherfraction.num
        newden = self.den * otherfraction.den
        return Fraction(newnum,newden)
f1 = Fraction(1,4)
f2 = Fraction(1,2)
print(f1+f2)
```

```
6/8
```