Operator Overloading in Python

The operator overloading in Python means provide extended meaning beyond their predefined operational meaning. Such as, we use the "+" operator for adding two integers as well as joining two strings or merging two lists. We can achieve this as the "+" operator is overloaded by the "int" class and "str" class. The user can notice that the same inbuilt operator or function is showing different behaviour for objects of different classes. This process is known as operator overloading.

Example:

```
print (14 + 32)
# Now, we will concatenate the two strings
print ("HELLO" + "CSE 3RD")

# We will check the product of two numbers
print (23 * 14)

# Here, we will try to repeat the String
print ("X Y Z " * 3)
```

How to Overload the Operators in Python?

Suppose the user has two objects which are the physical representation of a user-defined data type class. The user has to add two objects using the "+" operator, and it gives an error.

This is because the compiler does not know how to add two objects. So, the user has to define the function for using the operator, and that process is known as "operator overloading". The user can overload all the existing operators by they cannot create any new operator.

Python provides some special functions, or we can say magic functions for performing operator overloading, which is automatically invoked when it is associated with that operator. Such as, when the user uses the "+" operator, the magic function __add__ will automatically invoke in the command where the "+" operator will be defined.

How to Perform Binary "+" Operator in Python:

When the user uses the operator on the user-defined data types of class, then a magic function that is associated with the operator will be invoked automatically. The process of changing the behaviour of the operator is as simple as the behaviour of the function or method defined.

The user define methods or functions in the class and the operator works according to that behaviour defined in the functions. When the user uses the "+" operator, it will change the code of a magic function, and the user has an extra meaning of the "+" operator.

Program 1: Simply adding two objects.

Python program for simply using the overloading operator for adding two objects.

Example:

```
class example:
    def __init__(self, X):
        self.X = X

    # adding two objects
    def __add__(self, U):
        return self.X + U.X
object_1 = example( int( input( print ("Please enter the value: "))))
object_2 = example( int( input( print ("Please enter the value: "))))
print (": ", object_1 + object_2)
object_3 = example(str( input( print ("Please enter the value: "))))
object_4 = example(str( input( print ("Please enter the value: "))))
print (": ", object_3 + object_4)
```

Program 2: defining Overloading operator in another object

Python program for defining the overloading operator inside another object.

Example:

```
class complex_1:
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y

    # Now, we will add the two objects
    def __add__(self, U):
        return self.X + U.X, self.Y + U.Y

Object_1 = complex_1(23, 12)
Object_2 = complex_1(21, 22)
Object_3 = Object_1 + Object_2
print (Object_3)
```

Output:

```
(44, 34)
```

Program 3: Overloading comparison operators in Python

Python program for overloading comparison operators.

Example:

```
class example_1:
    def __init__(self, X):
        self.X = X
    def __gt__(self, U):
        if(self.X > U.X):
            return True
        else:
            return False
object_1 = example_1(int( input( print ("Please enter the value: "))))
object_2 = example_1(int (input( print("Please enter the value: "))))
if(object_1 > object_2):
    print ("The object_1 is greater than object_2")
else:
    print ("The object_2 is greater than object_1")
```

Output:

Case 1:

```
Please enter the value: 23
Please enter the value: 12
The object_1 is greater than object_2
```

Case 2:

```
Please enter the value: 20
Please enter the value: 31
The object_2 is greater than object_1
```

Program 4: Overloading equality and less than operators

Python Program for overloading equality and less than operators:

Example:

```
class E_1:
    def __init__(self, X):
        self.X = X
    def __lt__(self, U):
        if(self.X < U.X):
            return "object_1 is less than object_2"
```

```
            else:
                return "object_2 is less than object_1"
        def __eq__(self, U):
            if(self.X == U.X):
                return "Both the objects are equal"
            else:
                return "Objects are not equal"

    object_1 = E_1(int( input( print ("Please enter the value: "))))
    object_2 = E_1(int( input( print ("Please enter the value: "))))
    print (": ", object_1 < object_2)

    object_3 = E_1(int( input( print ("Please enter the value: "))))
    object_4 = E_1(int( input( print ("Please enter the value: "))))
    print (": ", object_3 == object_4)
```

Output:

Case 1:

```
Please enter the value: 12
Please enter the value: 23
:  object_1 is less than object_2
Please enter the value: 2
Please enter the value: 2
:  Both the objects are equal
```

Case 2:

```
Please enter the value: 26
Please enter the value: 3
: object_2 is less than object_1
Please enter the value: 2
Please enter the value: 5
: Objects are not equal
```

Python magic functions used for operator overloading:

Binary Operators:

Operator	Magic Function
+	__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 Function
<	__LT__(SELF, OTHER)
>	__GT__(SELF, OTHER)
<=	__LE__(SELF, OTHER)
>=	__GE__(SELF, OTHER)
==	__EQ__(SELF, OTHER)
!=	__NE__(SELF, OTHER)

Assignment Operators:

Operator	Magic Function
-=	__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 Operator:

Operator	Magic Function
-	__NEG__(SELF, OTHER)
+	__POS__(SELF, OTHER)
~	__INVERT__(SELF, OTHER)

Conclusion

In this tutorial, we have discussed overloading operators in Python and how to use them to perform various operators.