



Operator Overloading in Python

Difficulty Level : Medium • Last Updated : 19 Feb, 2023

[Read](#)

[Discuss](#)

[Courses](#)

[Practice](#)

[Video](#)

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called *Operator Overloading*.

Example

Python3

```
# Python program to show use of  
# + operator for different purposes.
```

```
print(1 + 2)
```

```
# concatenate two strings  
print("Geeks"+"For")
```

```
# Product two numbers  
print(3 * 4)
```

```
# Repeat the String  
print("Geeks"*4)
```

Output

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Got It !

How to overload the operators in Python?

Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary '+' operator it throws an error, because compiler don't know how to add two objects. So we define a method for an operator and that process is called operator overloading. We can overload all existing operators but we can't create a new operator. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method `__add__` is automatically invoked in which the operation for + operator is defined.

Overloading binary + operator in Python:

When we use an operator on user-defined data types then automatically a special function or magic function associated with that operator is invoked. Changing the behavior of operator is as simple as changing the behavior of a method or function. You define methods in your class and operators work according to that behavior defined in methods. When we use + operator, the magic method `__add__` is automatically invoked in which the operation for + operator is defined. Thereby changing this magic method's code, we can give extra meaning to the + operator.

Start Your Coding Journey Now!

How Does the Operator Overloading Actually work?

Whenever you change the behavior of the **existing operator** through operator overloading, you have to redefine the special function that is invoked automatically when the operator is used with the objects.

For Example:

Code 1:

Python3

```
# Python Program illustrate how
# to overload an binary + operator
# And how it actually works

class A:
    def __init__(self, a):
        self.a = a

    # adding two objects
    def __add__(self, o):
        return self.a + o.a

ob1 = A(1)
ob2 = A(2)
ob3 = A("Geeks")
ob4 = A("For")

print(ob1 + ob2)
print(ob3 + ob4)
# Actual working when Binary Operator is used.
print(A.__add__(ob1 , ob2))
print(A.__add__(ob3,ob4))
#And can also be Understand as :
print(ob1.__add__(ob2))
print(ob3.__add__(ob4))
```

Start Your Coding Journey Now!

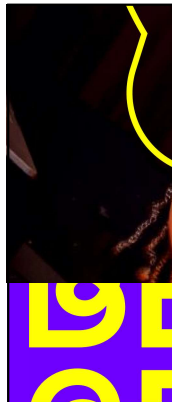
3

GeeksFor

3

GeeksFor

Here, We defined the special function "**__add__()**" and when the objects *ob1 and ob2* are coded as "**ob1 + ob2**", the special function is automatically called as **ob1.__add__(ob2)** which simply means that ob1 calls the **__add__()** function with ob2 as an Argument and It actually means **A.__add__(ob1, ob2)**. Hence, when the Binary operator is overloaded, the object before the operator calls the respective function with object after operator as parameter.



Code 2:

Python3

```
# Python Program to perform addition  
# of two complex numbers using binary  
# + operator overloading.
```

```
class complex:  
    def __init__(self, a, b):  
        self.a = a  
        self.b = b
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

```
print(ob3)
```

Output

```
(3, 5)
```

Overloading comparison operators in Python :

Python3

```
# Python program to overload
# a comparison operators

class A:
    def __init__(self, a):
        self.a = a
    def __gt__(self, other):
        if(self.a>other.a):
            return True
        else:
            return False

ob1 = A(2)
ob2 = A(3)
if(ob1>ob2):
    print("ob1 is greater than ob2")
else:
    print("ob2 is greater than ob1")
```

Output:

```
ob2 is greater than ob1
```

Overloading equality and less than operators:

Python3

```
# Python program to overload equality
# and less than operators
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

```
        return "ob2 is less than ob1"
def __eq__(self, other):
    if(self.a == other.a):
        return "Both are equal"
    else:
        return "Not equal"

ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)

ob3 = A(4)
ob4 = A(4)
print(ob1 == ob2)
```

Output:

```
ob1 is less than ob2
Not equal
```

Python magic methods or special functions for operator overloading

Binary Operators:

Operator	Magic Method
+	<code>__add__(self, other)</code>
-	<code>__sub__(self, other)</code>
*	<code>__mul__(self, other)</code>
/	<code>__truediv__(self, other)</code>
//	<code>__floordiv__(self, other)</code>
%	<code>__mod__(self, other)</code>
**	<code>__pow__(self, other)</code>
>>	<code>__rshift__(self, other)</code>
<<	<code>lshift (self, other)</code>

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

Operator	Magic Method
<	<code>__lt__(self, other)</code>
>	<code>__gt__(self, other)</code>
<=	<code>__le__(self, other)</code>
>=	<code>__ge__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>

Assignment Operators:

Operator	Magic Method
<code>--</code>	<code>__isub__(self, other)</code>
<code>+=</code>	<code>__iadd__(self, other)</code>
<code>*=</code>	<code>__imul__(self, other)</code>
<code>/=</code>	<code>__idiv__(self, other)</code>
<code>//=</code>	<code>__ifloordiv__(self, other)</code>
<code>%=</code>	<code>__imod__(self, other)</code>

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

<<=	__lshift__(self, other)
&=	__iand__(self, other)
=	__ior__(self, other)
^=	__ixor__(self, other)

Unary Operators:

Operator	Magic Method
-	__neg__(self)
+	__pos__(self)
~	__invert__(self)

Note: It is not possible to change the number of operands of an operator. For example: If we can not overload a unary operator as a binary operator. The following code will throw a syntax error.

Python3

```
# Python program which attempts to
# overload ~ operator as binary operator

class A:
    def __init__(self, a):
        self.a = a

    # Overloading ~ operator, but with two operands
    def __invert__(self):
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

Output

This is the ~ operator, overloaded as binary operator.

operator overloading on Boolean values:

In Python, you can overload the Boolean operators and, or, and not by defining the `__and__`, `__or__`, and `__not__` special methods in your class.

Here's an example of how to overload the and operator for a custom class:

Python

```
class MyClass:
    def __init__(self, value):
        self.value = value

    def __and__(self, other):
        return MyClass(self.value and other.value)

a = MyClass(True)
b = MyClass(False)
c = a & b  # c.value is False
```

Explanation:

In this example, we define a `MyClass` that has a single attribute `value`, which is a boolean. We then overload the `&` operator by defining the `__and__` method to perform a logical and operation on the `value` attribute of two `MyClass` instances.

When we call `a & b`, the `__and__` method is called with `a` as the first argument and `b` as the second argument. The method returns a new instance of `MyClass` with a `value` attribute that is the logical and of `a.value` and `b.value`.

Note that Python also provides built-in boolean operators that can be used with any

Start Your Coding Journey Now!

Advantages:

Overloading boolean operators in a custom class can provide several advantages, including:

1. Improved readability: By overloading boolean operators, you can provide a more natural syntax and semantics for your class that makes it easier to read and understand.
2. Consistency with built-in types: Overloading boolean operators can make your class behave more like built-in types in Python, which can make it easier to use and integrate with other code.
3. Operator overloading: Overloading boolean operators is an example of operator overloading in Python, which can make your code more concise and expressive by allowing you to use familiar operators to perform custom operations on your objects.
4. Custom behavior: Overloading boolean operators can allow you to define custom behavior for your class that is not available in built-in types or other classes.
5. Enhanced functionality: By overloading boolean operators, you can add new functionality to your class that was not available before, such as the ability to perform logical and or or operations on instances of your class.

Overall, overloading boolean operators in a custom class can make your code more readable, consistent, concise, expressive, and functional. However, it's important to use operator overloading judiciously and only when it makes sense for the semantics of your class.

Related Articles

1. Concatenate two strings using Operator Overloading in Python

2. Benefits of Double Division Operator over Single Division Operator in Python

3. Python | Method Overloading

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

6. Operator Functions in Python | Set 2

7. Operator Functions in Python | Set 1

8. Python | Operator.countOf

9. Python NOT EQUAL operator

10. Walrus Operator in Python 3.8

Like 206

Previous

Next

Article Contributed By :



bestharadhakrishna
@bestharadhakrishna

Vote for difficulty

Current difficulty : [Medium](#)

Easy

Normal

Medium

Hard

Expert

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

[Improve Article](#)

[Report Issue](#)



A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

[About Us](#)
[Careers](#)
[In Media](#)
[Contact Us](#)
[Privacy Policy](#)
[Copyright Policy](#)
[Advertise with us](#)

News

[Top News](#)
[Technology](#)
[Work & Career](#)
[Business](#)
[Finance](#)
[Lifestyle](#)

Learn

[DSA](#)
[Algorithms](#)
[Data Structures](#)
[SDE Cheat Sheet](#)
[Machine learning](#)
[CS Subjects](#)
[Video Tutorials](#)
[Courses](#)

Languages

[Python](#)
[Java](#)
[CPP](#)
[Golang](#)
[C#](#)
[SQL](#)

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Start Your Coding Journey Now!

HTML

JavaScript

Bootstrap

ReactJS

NodeJS

Pick Topics to Write

Write Interview Experience

Internships

Video Internship

@geeksforgeeks , Some rights reserved