The Python Guru

Become a better python developer

Python Operator Overloading

You have already seen you can use `+` operator for adding numbers and at the same time to concatenate strings. It is possible because `+` operator is overloaded by both `int` class and `str` class. The operators are actually methods defined in respective classes. Defining methods for operators is known as operator overloading. For e.g. To use `+` operator with custom objects you need to define a method called `__add__` .

Let's take an example to understand better

```
import math

class Circle:

    def __init__(self, radius):
        self.__radius = radius

    def setRadius(self, radius):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def area(self):
        return math.pi * self.__radius ** 2

    def __add__(self, another_circle):
        return Circle( self.__radius + another_circle.__radius )

c1 = Circle(4)
print(c1.getRadius())

c2 = Circle(5)
print(c2.getRadius())

c3 = c1 + c2 # This became possible because we have overloaded + operator by adding
print(c3.getRadius())
```

Expected Output:

```
4
5
9
```

In the above example we have added `__add__` method which allows use to use `+` operator to add two circle objects. Inside the `__add__` method we are creating a new object and returning it to the caller.

python has many other special methods like `__add__` , see the list below.

OPERATOR	FUNCTION	METHOD DESCRIPTION
`+`	`__add__(self, other)`	Addition
`*`	`__mul__(self, other)`	Multiplication
`-`	`__sub__(self, other)`	Subtraction
`%`	`__mod__(self, other)`	Remainder
`/`	`__truediv__(self, other)`	Division
`<`	`__lt__(self, other)`	Less than
`<=`	`__le__(self, other)`	Less than or equal to
`==`	`__eq__(self, other)`	Equal to
`!=`	`__ne__(self, other)`	Not equal to
`>`	`__gt__(self, other)`	Greater than
`>=`	`__ge__(self, other)`	Greater than or equal to
`[index]`	`__getitem__(self, index)`	Index operator
`in`	`__contains__(self, value)`	Check membership
`len`	`__len__(self)`	The number of elements
`str`	`__str__(self)`	The string representation

Program below is using some of the above mentioned functions to overload operators.

```
1  import math
2
3  class Circle:
4
5      def __init__(self, radius):
6          self.__radius = radius
7
8      def setRadius(self, radius):
9          self.__radius = radius
10
11     def getRadius(self):
12         return self.__radius
13
14     def area(self):
```

```
        return math.pi * self.__radius ** 2

    def __add__(self, another_circle):
        return Circle( self.__radius + another_circle.__radius )

    def __gt__(self, another_circle):
        return self.__radius > another_circle.__radius

    def __lt__(self, another_circle):
        return self.__radius < another_circle.__radius

    def __str__(self):
        return "Circle with radius " + str(self.__radius)

c1 = Circle(4)
print(c1.getRadius())

c2 = Circle(5)
print(c2.getRadius())

c3 = c1 + c2
print(c3.getRadius())

print( c3 > c2) # Became possible because we have added __gt__ method

print( c1 < c2) # Became possible because we have added __lt__ method

print(c3) # Became possible because we have added __str__ method
```

Expected Output:

```
4
5
9
True
True
Circle with radius 9
```

Next lesson is inheritance and polymorphism.

Other Tutorials

Related

Python Object and Classes
August 20, 2015
In "Python Basics"

Python inheritance and polymorphism
August 21, 2015

Python Functions
August 19, 2015
In "Python Basics"