Stack operator in Python

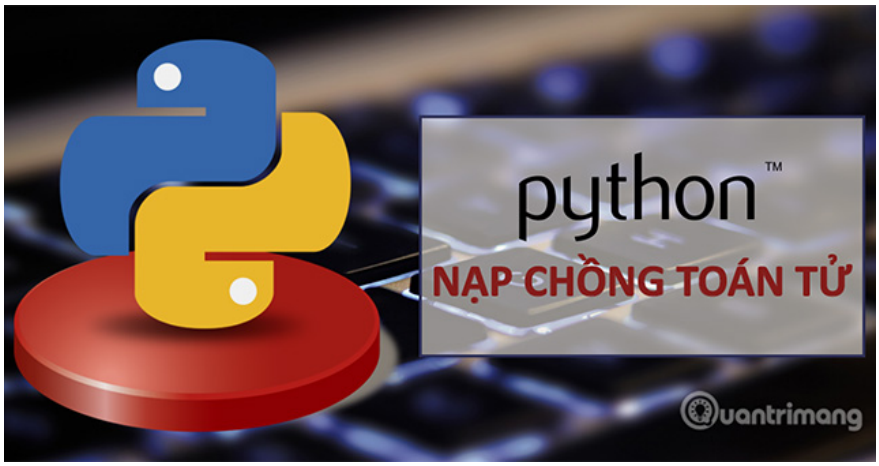
You can change the meaning of the operator in Python depending on the operand used and we call that operator overloading. Quantrimang will learn more about this content through the article. Invites you to read the track.

You can change the meaning of the operator in Python depending on the operand used and we call that operator overloading.

What is operator overloading in Python?

Python operators work with built-in functions, but an operator can be used to perform many different operations. For example with the '+' operator, you can add arithmetic to two numbers together, can combine two lists, or join two different strings .

This feature in Python is called operator overloading, allowing the same operator to be used differently depending on the context.



So what happens when we use operator overloading with the object of a class declared by the user? Follow the example of simulating a point in the following two-dimensional coordinate system:

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

We run the program and enter the points:

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

The program immediately reports a *TypeError* error because Python cannot receive two *Point* objects at the same time.

To handle this problem, we will use operator overloading.

First, learn some of the following special functions.

Special functions in Python

The function in Class begins with two consecutive underscores *(__) which* are special functions, which have special meanings.

There are many special functions in Python, one of which is the *__init __ ()* function that Quantrimang introduced earlier in the Class and Object lesson. This function is called whenever an object is initialized, a new variable in the class.

The purpose of using these special functions is to make our functions compatible with Python built-in functions.

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

You should declare the *__str __ () method* in the class to control how the results are printed.

```
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)
```

And now try again with the *print ()* function .

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

Using *__str __ ()* results in a more standard display. Also you can print the same result using Python's built-in function str () or format ().

```
>>> str(p1)
'(2,3)'

>>> format(p1)
```

```
'(2,3)'
```

When using str () and format (), Python executes the call p1 .__ str __ (), so the result is returned similarly.

Overload '+' operator in Python

To overload the '+' operator, we will use the *__add __ ()* function in the class. We can deploy many jobs using this function, such as adding two coordinates in the example above.

```
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)
```

We run the program and enter the points:

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

In this program, when you execute *p1 + p2* , Python will call *p1 .__ add __ (p2).*

Similarly, you can overload many other operators. Quantrimang would like to introduce some special functions for operator overloading in the table below:

MATHEMATICS OF ACTIVITIES P1 + p2 p1 .__ add __ (p2) Subtraction p1 - p2 p1 .__ sub __ (p2) Multiplication p1 * p2 p1 .__ mul __ (p2) Excess p1 ** p2 p1 .__ pow __ (p2) Division p1 / p2 p1 .__ truediv __ (p2) Partial division division p1 // p2 p1 .__ floordiv __ (p2) Balance (modulo) p1% p2 p1 .__ mod __ (p2) Operation on bit: left translation p1 p2 p1 .__ lshift __ (p2) Bit manipulation: right shift p1 >> p2 p1 .__ rshift __ (p2) Bit manipulation: permission AND p1 & p2 p1 .__ and __ (p2) Operation on bit: enable OR p1 | p2 p1 .__ or __ (p2) Bit operation: XOR p1 ^ p2 p1 .__ xor __ (p2) Bit operation: NOT ~ p1 p1 .__ invert __ ()

Overload comparison operator in Python

Python is not only limited to overloading math operators, but also allowing users to overload comparison operators.

There are many comparison operators supported by Python, such as: ,>, =,> =, ==, .

You use this operator overload when you want to compare objects in the class together.

For example, if you want to compare points in a class point , compare the magnitude of these points starting from the origin, doing the following:

```
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
```

We run the program and enter the points and operators used to compare:

```
>>> Point(1,1)  Point(-2,-3)
True

>>> Point(1,1)  Point(0.5,-0.2)
False

>>> Point(1,1)  Point(1,1)
False
```

Similarly, you can overload many other comparison operators. Quantrimang would like to introduce some special functions for comparison operator overloading in the table below:

OPERATING ACTIVITIES Smaller than p1 = p2 p1 .__ le __ (p2) Equal to p1 == p2
p1 .__ eq __ (p2) Other p1! = p2 p1 .__ ne __ (p2) Larger than p1> p2 p1 .__ gt __ (p2) Greater than or equal to p1> = p2 p1 .__ ge __ (p2)

See more:

1. Operator overloading in C #
2. Overload relational operator in C ++
3. Object-oriented programming in Python

Previous article: Multiple Inheritance in Python