Method Overriding in Python

Method overriding is a concept of object oriented programming that allows us to change the implementation of a function in the **child class** that is defined in the **parent class**. It is the ability of a child class to change the implementation of any method which is already provided by one of its parent class(ancestors).

Following conditions must be met for overriding a function:

1. **Inheritance** should be there. Function overriding cannot be done within a class. We need to derive a child class from a parent class.
2. The function that is redefined in the child class should have the same signature as in the parent class i.e. same **number of parameters**.

As we have already learned about the **concept of Inheritance**, we know that when a **child class** inherits a **parent class** it also get access to it public and protected(**access modifiers in python**) variables and methods, for example,

# parent class

class Parent:

# some random function

def anything(self):

print('Function defined in parent class!')

# child class

class Child(Parent):

# empty class definition

pass

obj2 = Child()

obj2.anything()

Function defined in parent class!

While the child class can access the parent class methods, it can also provide a new implementation to the parent class methods, which is called **method overriding**.

Python Method Overriding Example

Let's take a very cool example which we also had in the inheritance tutorial. There is a parent class named Animal:

class Animal:

# properties

multicellular = True

# Eukaryotic means Cells with Nucleus

eukaryotic = True

# function breathe

def breathe(self):

print("I breathe oxygen.")

# function feed

def feed(self):

print("I eat food.")

Let's create a child class Herbivorous which will extend the class Animal:

class Herbivorous(Animal):

# function feed

def feed(self):

print("I eat only plants. I am vegetarian.")

In the child class Herbivorous we have overridden the method feed().

So now when we create an object of the class Herbivorous and call the method feed() the overridden version will be executed.

herbi = Herbivorous()

herbi.feed()

# calling some other function

herbi.breathe()

I eat only plants. I am vegetarian.

I breathe oxygen.

Click on **Run** to see the code in action and feel free to make changesto it and run again.

# Operator overloading in Python

Operators are used in Python to perform specific operations on the given operands. The operation that any particular operator will perform on any predefined data type is already defined in Python.

Each operator can be used in a different way for different types of operands. For example, + operator is used for **adding two integers** to give an integer as a result but when we use it with **float operands**, then the result is a float value and when + is used with **string operands** then it concatenates the two operands provided.

This different behaviour of a single operator for different types of operands is called **Operator Overloading**. The use of + operator with different types of operands is shown below:

>>> x=10

>>> y=20

>>> x+y

30

>>> z=10.4

>>> x+z

20.4

>>> s1 = 'hello '

>>> s2 = 'world'

>>> s1+s2

'hello world'

### Can + Operator Add anything?

The answer is No, it cannot. Can you use the + operator to add two objects of a class. The + operator can add two integer values, two float values or can be used to concatenate two strings only because these behaviours have been defined in python.

So if you want to use the same operator to add two objects of some user defined class then you will have to defined that behaviour yourself and inform python about that.

If you are still not clear, let's create a class and try to use the + operator to add two objects of that class,

class Complex:

def \_\_init\_\_(self, r, i):

self.real = r

self.img = i

c1 = Complex(5,3)

c2 = Complex(2,4)

print("sum = ", c1+c2)

Traceback (most recent call last):

File "/tmp/sessions/1dfbe78bb701d99d/main.py", line 7, in

print("sum = ", c1+c2)

TypeError: unsupported operand type(s) for +: 'Complex' and 'Complex'

So we can see that the + operator is not supported in a user-defined class. But we can do the same by overloading the + operator for our class Complex. But how can we do that?

## Special Functions in Python

Special functions in python are the functions which are used to perform special tasks. These special functions have \_\_ as prefix and suffix to their name as we see in \_\_init\_\_() method which is also a special function. Some special functions used for overloading the operators are shown below:

### Mathematical Operator

Below we have the names of the special functions to overload the mathematical operators in python.

|  |  |  |
| --- | --- | --- |
| **Name** | **Symbol** | **Special Function** |
| Addition | + | \_\_add\_\_(self, other) |
| Subtraction | - | \_\_sub\_\_(self, other) |
| Division | / | \_\_truediv\_\_(self, other) |
| Floor Division | // | \_\_floordiv\_\_(self, other) |
| Modulus(or Remainder) | % | \_\_mod\_\_(self, other) |
| Power | \*\* | \_\_pow\_\_(self, other) |

### Assignment Operator

Below we have the names of the special functions to overload the assignment operators in python.

|  |  |  |
| --- | --- | --- |
| **Name** | **Symbol** | **Special Function** |
| Increment | += | \_\_iadd\_\_(self, other) |
| Decrement | -= | \_\_isub\_\_(self, other) |
| Product | \*= | \_\_imul\_\_(self, other) |
| Division | /= | \_\_idiv\_\_(self, other) |
| Modulus | %= | \_\_imod\_\_(self, other) |
| Power | \*\*= | \_\_ipow\_\_(self, other) |

### Relational Operator

Below we have the names of the special functions to overload the relational operators in python.

|  |  |  |
| --- | --- | --- |
| **Name** | **Symbol** | **Special Function** |
| Less than | < | \_\_lt\_\_(self, other) |
| Greater than | > | \_\_gt\_\_(self, other) |
| Equal to | == | \_\_eq\_\_(self, other) |
| Not equal | != | \_\_ne\_\_(self, other) |
| Less than or equal to | <= | \_\_le\_\_(self, other) |
| Greater than or equal to | > = | \_\_gt\_\_(self, other) |

It's time to see a few code examples where we actually use the above specified special functions and overload some operators.

### Overloading + operator

In the below code example we will overload the + operator for our class Complex,

class Complex:

# defining init method for class

def \_\_init\_\_(self, r, i):

self.real = r

self.img = i

# overloading the add operator using special function

def \_\_add\_\_(self, sec):

r = self.real + sec.real

i = self.img + sec.img

return complx(r,i)

# string function to print object of Complex class

def \_\_str\_\_(self):

return str(self.real)+' + '+str(self.img)+'i'

c1 = Complex(5,3)

c2 = Complex(2,4)

print("sum = ",c1+c2)

sum = 7 + 7i

In the program above, \_\_add\_\_() is used to overload the + operator i.e. when + operator is used with two Complex class objects then the function \_\_add\_\_() is called.

\_\_str\_\_() is another special function which is used to provide a format of the object that is suitable for printing.

### Overloading < operator

Now let's overload the less than operator so that we can easily compare two Complex class object's values by using the less than operaton <.

As we know now, for doing so, we have to define the \_\_lt\_\_ special function in our class.

Based on your requirement of comparing the class object, you can define the logic for the special functions for overriding an operator. In the code above, we have given precedence to the real part of the complex number, if that is less then the whole complex number is less, if that is equal then we check for the imaginary part.

### Conclusion

Overloading operators is easy in python using the special functions and is less confusion too.