Object Oriented Programming using Python (I)

Lecture(7)

Class Inheritance
Overriding methods

Prepared by.Ahmed Eskander Mezher
University of Information Technology and Communications
College of Business Informatics

Multi Inheritance (2)

There are 2 built-in functions in Python that are related to inheritance to check a relationships of two classes and instances.

They are:

1. **isinstance():** It checks the type of an object.

Its syntax is:

isinstance(object_name, class_name)

It would return **True** if the class of object_name is class_name else **False**.

```python
class a:
    def m(self):
        print("message from class a")
class b(a):
    def m(self):
        print("message from class b")
class c():
    def m(self):
        print("message from class c")
class d(b,c):
    def m(self):
        print("message from class d")
        a.m(self)
        b.m(self)
        c.m(self)
obj1=d()
obj1.m()
obj2=a()
print(isinstance(obj1,a))
print(isinstance(obj1,d))
print(isinstance(obj2,c))
print(isinstance(5, int))
```

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093,
1)] on win32
Type "copyright", "credits" or "l
>>>
 RESTART: C:\Users\Rula\AppData\L
nce.py
message from class d
message from class a
message from class b
message from class c
True
True
False
True
>>>
```

This is because 5 is an integer and hence belongs to the class of int.

NOTE: 'int' is both a type and a class in Python.

2. issubclass(): It checks whether a specific class is the child class of another class or not. Its syntax is:
issubclass(childclass_name, parentclass_name)
It would return **True** if the entered child class is actually derived from the entered parent class else, it returns **False**.
For example:

```
# Python code to demonstrate issubclass()
class A():
      def __init__(self, a):
            self.a = a
class B(A):
      def __init__(self, a, b):
            self.b = b
            A.__init__(self, a)

print(issubclass(B, A))
```

Output:

```
True
```

hybrid inhertance.py - C:\Users\Rula\AppData\Local\Programs\Python\Python37-32\hybrid inhertance.py (3.7.0)

File Edit Format Run Options Window Help

```python
class a:
    def m(self):
        print("message from class a")
class b(a):
    def m(self):
        print("message from class b")
class c():
    def m(self):
        print("message from class c")
class d(b,c):
    def m(self):
        print("message from class d")
        a.m(self)
        b.m(self)
        c.m(self)
obj1=d()
obj1.m()
obj2=a()
print(issubclass(d,b))
print(issubclass(c,a))
print(issubclass(a,b))
print(issubclass(b,a))
```

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc50
l)] on win32
Type "copyright", "credits" c
>>>
 RESTART: C:\Users\Rula\AppDa
nce.py
message from class d
message from class a
message from class b
message from class c
True
False
False
True
>>> |
```

Overriding Methods:

Overriding a method means redefining a method in the subclass when it has already been defined in some other class.

A method in the subclass would be called as overridden only when there exists another method with the same name and same set of parameters in the superclass.

For example:

```
# Base Class
class A(object):
        def __init__(self):
                constant1 = 1
        def method1(self):
                print('method1 of class A')

class B(A):
        def __init__(self):
                constant2 = 2
                self.calling1()
                A. __init__(self)
        def method1(self):
                print('method1 of class B')
        def calling1(self):
                self.method1()
                A.method1(self)
b = B()
```

Output:

```
method1 of class B
method1 of class A
```

For new-style classes, where the parent class inherits from the built-in 'object' class, there is another procedure for overriding methods.
The super() method helps us in overriding methods in new style classes. Its syntax is as follows:

super(class_name, instance_of_class).overridden_method_name()

Let us assume there are 3 classes A, B, and C. All 3 of them have a common function called 'method1'. Here comes the work of super().

```
class A(object):
        def function1(self):
                print 'function of class A'
class B(A):
        def function1(self):
                print 'function of class B'
                super(B, self).function1()
class C(B):
        def function1(self):
                print 'function of class C'
                super(C, self).function1()
j = C()
j.function1()
```

Output:

```
function of class C
function of class B
function of class A
```

The 'self' parameter within super function acts as the object of the parent class and hence invokes the function1 of the parent class.

polygon.py - C:\Users\Rula\AppData\Local\Programs\Python\Python37-32\polygon.py (3.7.0)

File Edit Format Run Options Window Help

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides

        self.sides=[]
    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
class Triangle(Polygon):
    def __init__(self):
        Polygon.__init__(self,3)
```

```
    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is ' ,area)
t = Triangle()
t.inputSides()
t.dispSides()



t.findArea()
```