Advanced Object Oriented Programming in Python

Session - 4

Agenda

- Muliple Inheritance
- Multilevel Inheritance
- Method Resolution Order (MRO)
- Method Overriding
- Methods Types

    Static Method

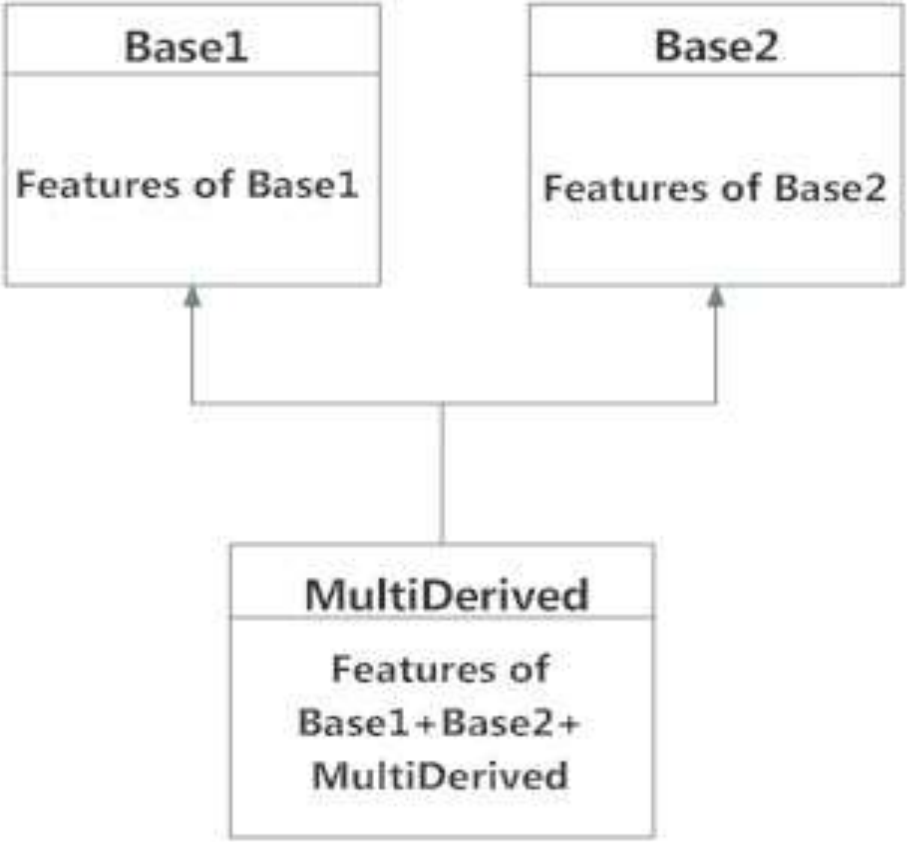
    Class Method

Multiple Inheritance

- Multiple inheritance is possible in Python.
- A class can be derived from more than one base classes. The syntax for multiple inheritance is similar to single inheritance.
- Here is an example of multiple inheritance.

Syntax:

```
class Base1:
        Statements
class Base2:
        Statements
class MultiDerived(Base1, Base2):
        Statements
```



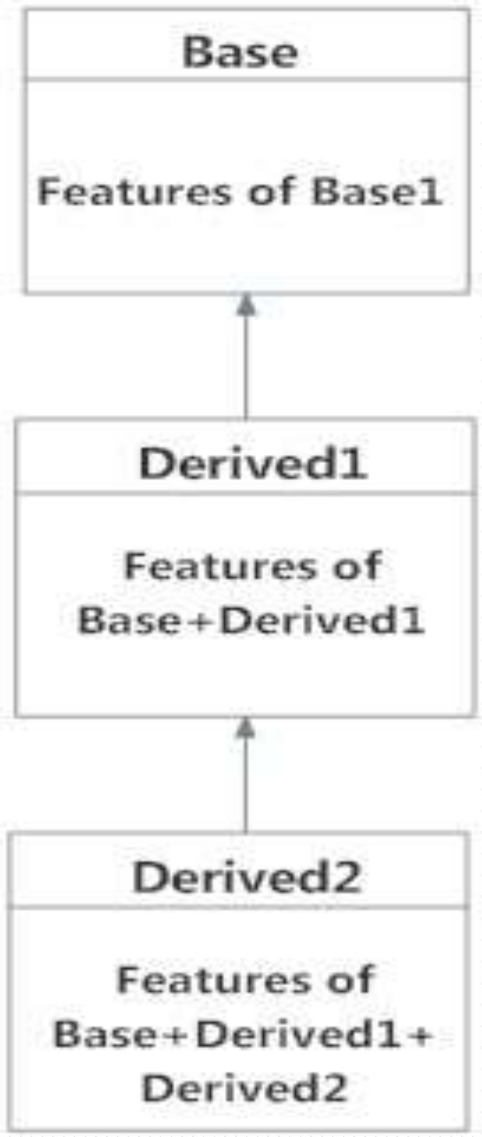
The class MultiDerived inherits from both Base1 and Base2

Multilevel Inheritance

- On the other hand, we can inherit form a derived class.
- This is also called multilevel inheritance.
- Multilevel inheritance can be of any depth in Python.
- An example with corresponding visualization is given below.

Syntax:

```
class Base:
        pass
class Derived1(Base):
        pass
class Derived2(Derived1):
        pass
```



The class Derivedd1 inherits from Base and Derivedd2 inherits from both Base as well as Derived1

Method Resolution Order in Python

- Every class in Python is derived from the class object.
- In the multiple inheritance scenario, any specified attribute is searched first in the current class. If not found, the search continues into parent classes in depth-first, left-right fashion without searching same class twice.
- So, in the above example of MultiDerived class the search order is [MultiDerived, Base1, Base2, object].
- This order is also called linearization of MultiDerived class and the set of rules used to find this order is called Method Resolution Order (MRO)

continue…

…continue

- MRO must prevent local precedence ordering and also provide monotonicity.
- It ensures that a class always appears before its parents and in case of multiple parents, the order is same as tuple of base classes.
- MRO of a class can be viewed as the __mro__ attribute or mro() method. The former returns a tuple while latter returns a list

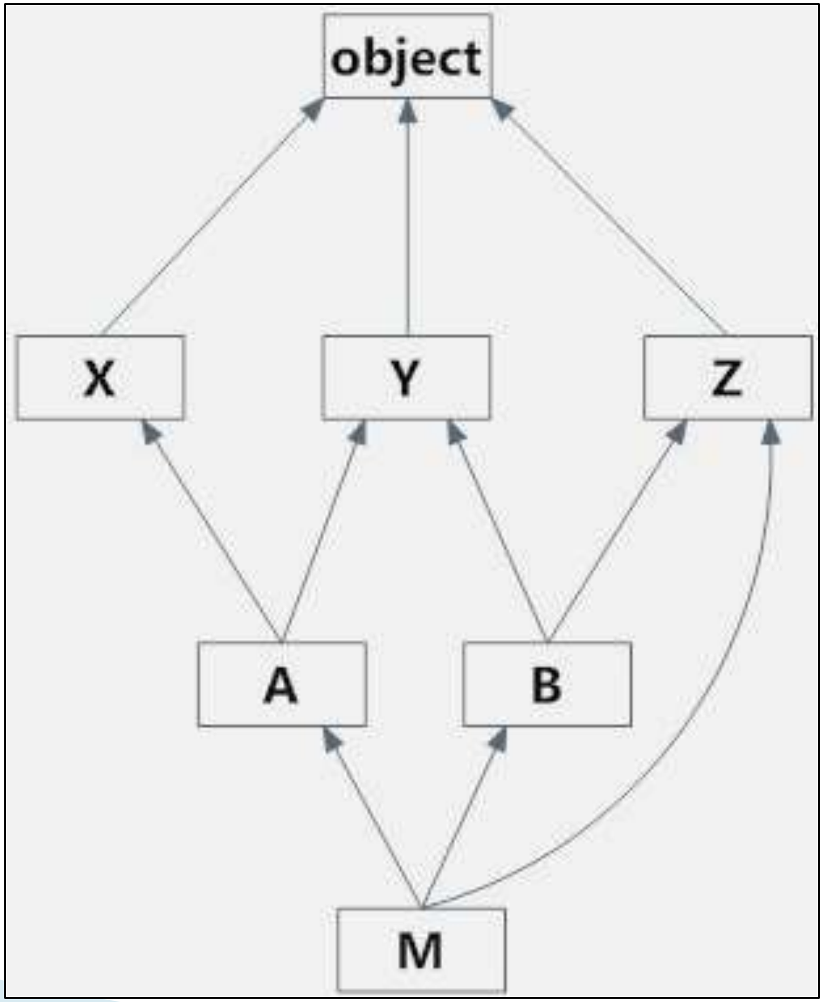
```
>>>MultiDerived.__mro__
  (<class '__main__.MultiDerived'>,
  <class '__main__.Base1'>,
  <class '__main__.Base2'>,
  <class 'object'>)

>>> MultiDerived.mro()
  [<class '__main__.MultiDerived'>,
  <class '__main__.Base1'>,
  <class '__main__.Base2'>,
  <class 'object'>]
```

Here is a little more complex multiple inheritance example and its visualization along with the MRO.

Example:

```
class X:
   pass
class Y:
   pass
class Z:
    pass
class A(X,Y): pass
class B(Y,Z): pass
class M(B,A,Z): pass
print(M.mro())
```

Method Resolution Order in Python for above simple code



Method Overriding

- The ***subclasses*** can override the logic in a ***superclass***, allowing you to change the behavior of your classes without changing the ***superclass*** at all.
- Because changes to program logic can be made via subclasses, the use of classes generally supports code reuse and extension better than traditional functions do.
- Functions have to be rewritten to change how they work whereas classes can just be subclassed to redefine methods.

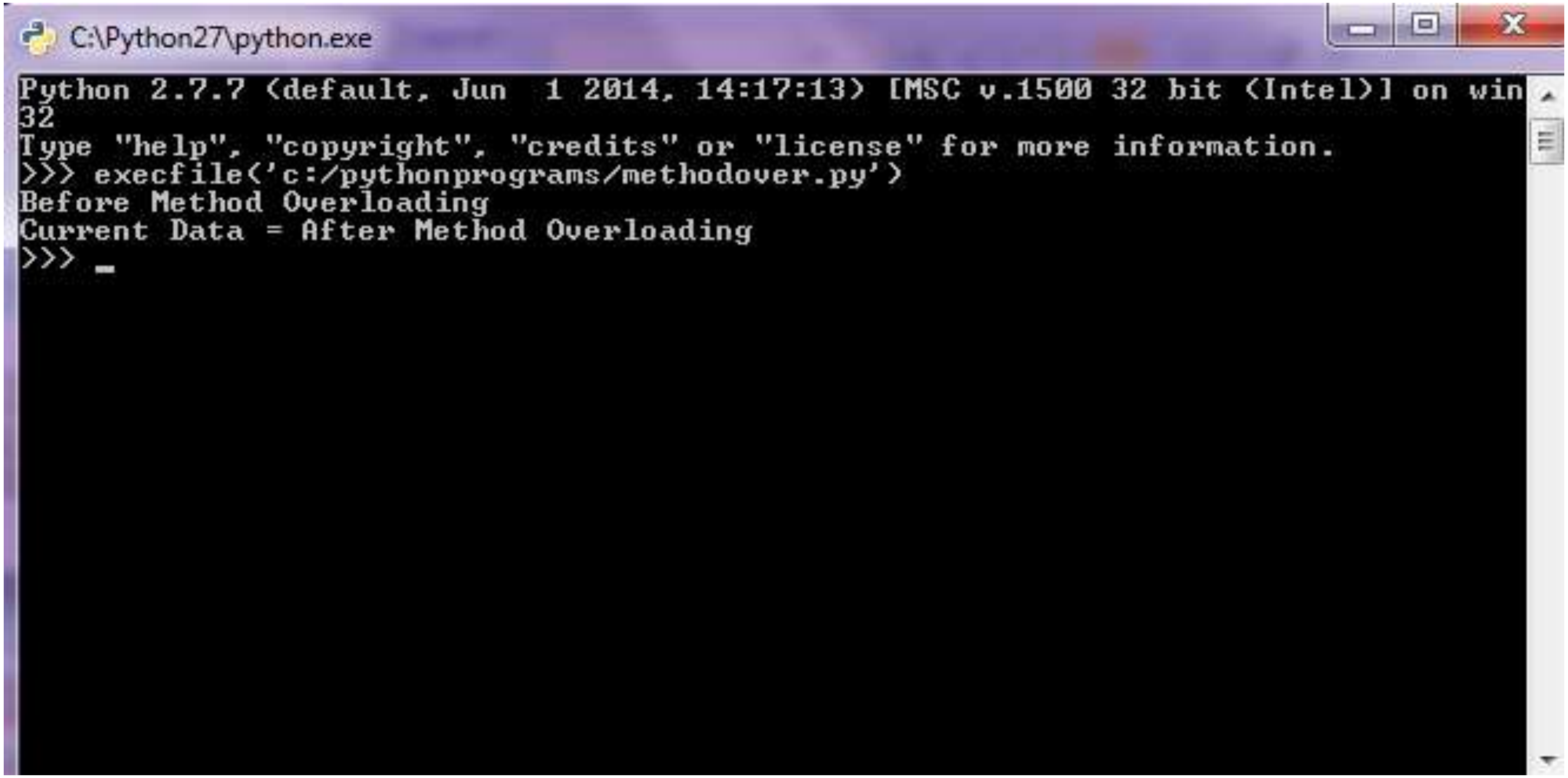
Method Overriding Sample Program

```
class FirstClass:                 #define the super class
    def setdata(self, value):     # define methods
            self.data = value     # ‘self’ refers to an instance
    def display(self):
            print self.data

class SecondClass(FirstClass):                      # inherits from FirstClass
    def display(self):                              # redefines display
            print 'Current Data = %s'  % self.data
x=FirstClass()                    # instance of FirstClass
y=SecondClass()                   # instance of SecondClass
x.setdata('Before Method Overloading')
y.setdata('After Method Overloading')
x.display()
y.display()
```

Explanation for Method Overriding Sample Program

- Both instances (x and y) use the same setdata method from FirstClass; x uses it because it’s an instance of FirstClass while y uses it because SecondClass inherits setdata from FirstClass.
- However, when the display method is called, x uses the definition from FirstClass but y uses the definition from SecondClass, where display is overridden.

Output for the Method Overriding Sample Program



```
Python 2.7.7 (default, Jun  1 2014, 14:17:13) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> execfile('c:/pythonprograms/methodover.py')
Before Method Overloading
Current Data = After Method Overloading
>>> _
```

Method Types

- It Possible to define two kinds of methods with in a class that can be called without an instance

  1) static method

  2) class method
- Normally a class method is passed ‘self’ as its first argument. Self is an instance object
- Some times we need to process data associated with instead of instances
- Let us assume, simple function written outside the class, the code is not well associated with class, can’t be inherited and the name of the method is not localized
- Hence python offers us static and class methods

Static Method

- Simple function with no self argument
- Nested inside class
- Work on class attribute not on instance attributes
- Can be called through both class and instance
- The built in function static method is used to create them

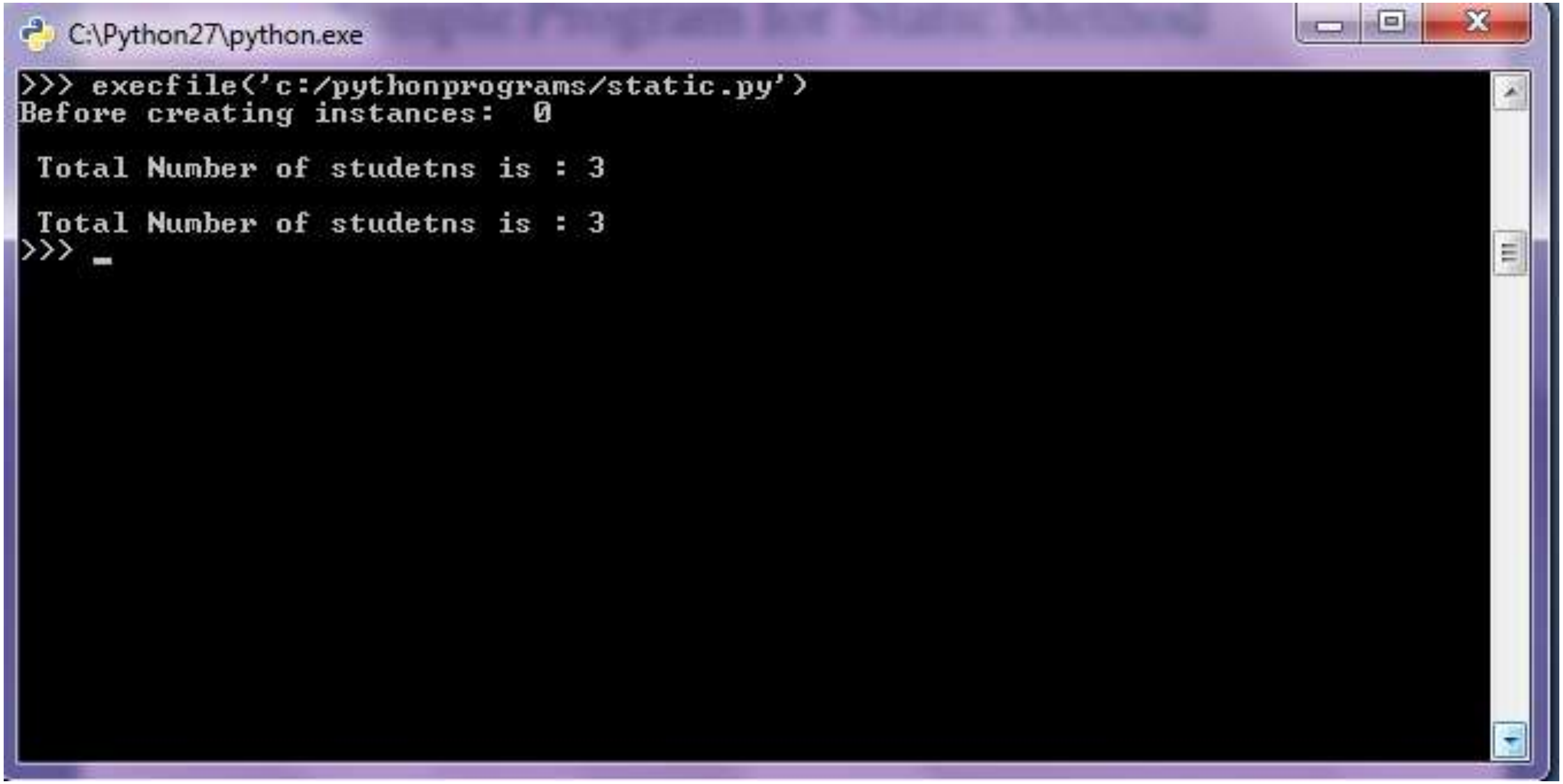
Syntax for Static Method

```
class MyClass:
   def my_static_method():
          --------------------------
          ----rest of the code---
my_static_method=staticmethod (my_static_method)
```

Sample Program for Static Method

```
class Students(object):
    total = 0
    def status():
            print '\n Total Number of studetns is :', Students.total
    status= staticmethod(status)
    def __init__(self, name):
            self.name= name
            Students.total+=1
print ‘Before Creating instance: ‘, Students.total
student1=Students('Guido')
student2=Students('Van')
student3=Students('Rossum')

Students.status() # Accessing the class attribute through direct class name
student1.status() # Accessing the class attribute through an object
```

Output for sample program of Static Method



```
>>> execfile('c:/pythonprograms/static.py')
Before creating instances:  0

 Total Number of studetns is : 3

 Total Number of studetns is : 3
>>> _
```

Class Method

- Functions that have first argument as class name
- Can be called through both class and instance
- These are created with classmethod() inbuilt function
- These always receive the lowest class in an instance’s tree

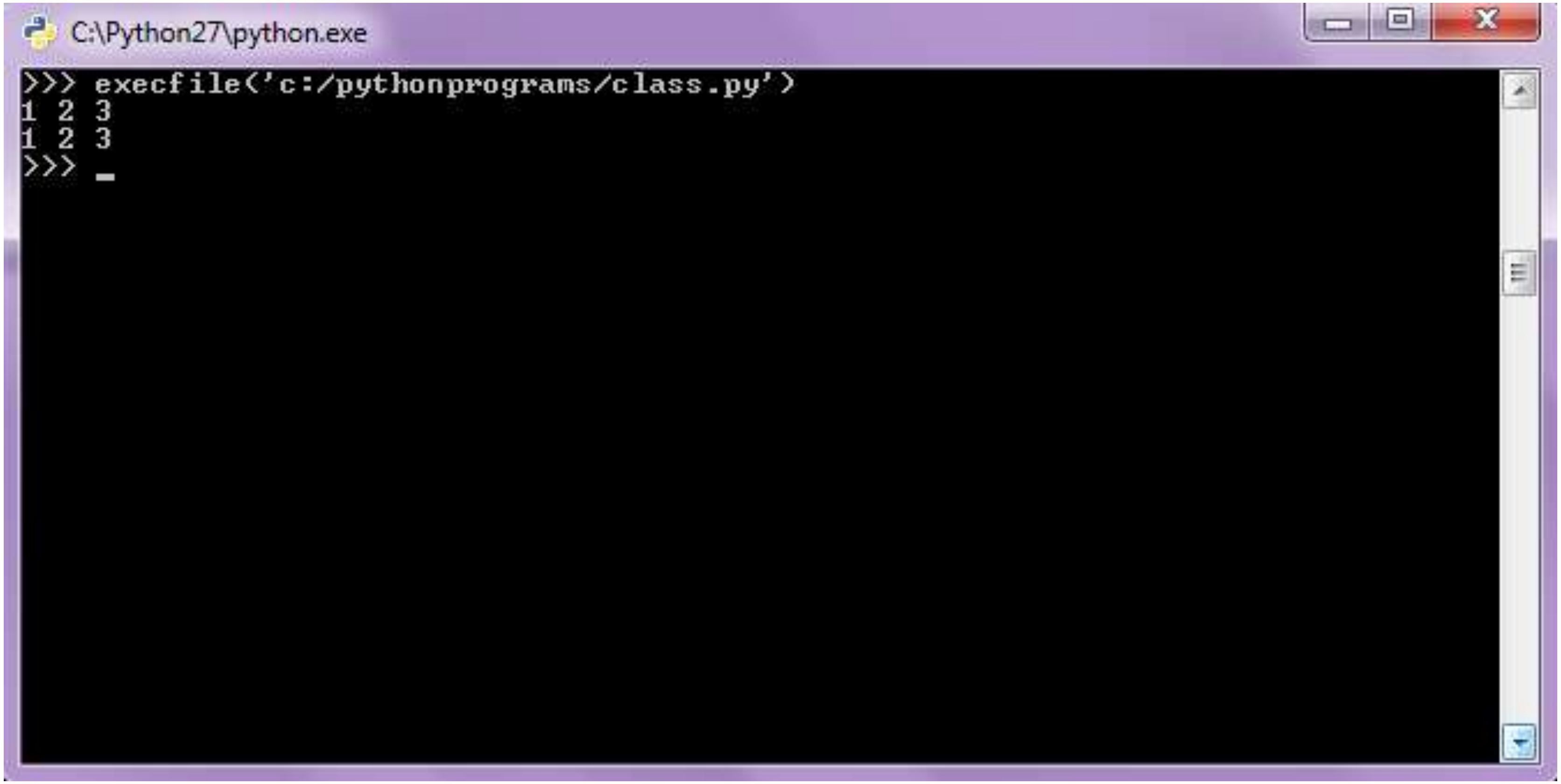
Syntax for Class Method

```
class MyClass:
   def my_class_method(class _ var):
         --------------------------
         ----rest of the code---
my_class_method=classmethod (my_class_method)
```

Sample code for Class Method

```
class Spam:
    numinstances = 0
    def count(cls):
            cls.numinstances +=1
    def __init__(self):
            self.count()
    count=classmethod(count)    # Converts the count function to class method
class Sub(Spam):
    numinstances = 0
class Other(Spam):
    numinstances = 0
S= Spam()
y1,y2=Sub(),Sub()
z1,z2,z3=Other(),Other(),Other()
print S.numinstances, y1.numinstances,z1.numinstances
print Spam.numinstances, Sub.numinstances,Other.numinstances
```

Output for Sample Program of Class Method



```
>>> execfile('c:/pythonprograms/class.py')
1 2 3
1 2 3
>>> _
```