Advanced Class

Overloading

- OOP allows the programmers to add some additional functionalities to the operators and method which have basic properties.
- Such a kind of redefining of the entities of the programming structure is called as polymorphism.
- Overloading is a type of polymorphism that adds additional functionalities for the existing properties of objects or the object itself.
- Overloading exhibits code reusability and code readability.
- Overloading is of two types:

Method overloading (or function overloading)

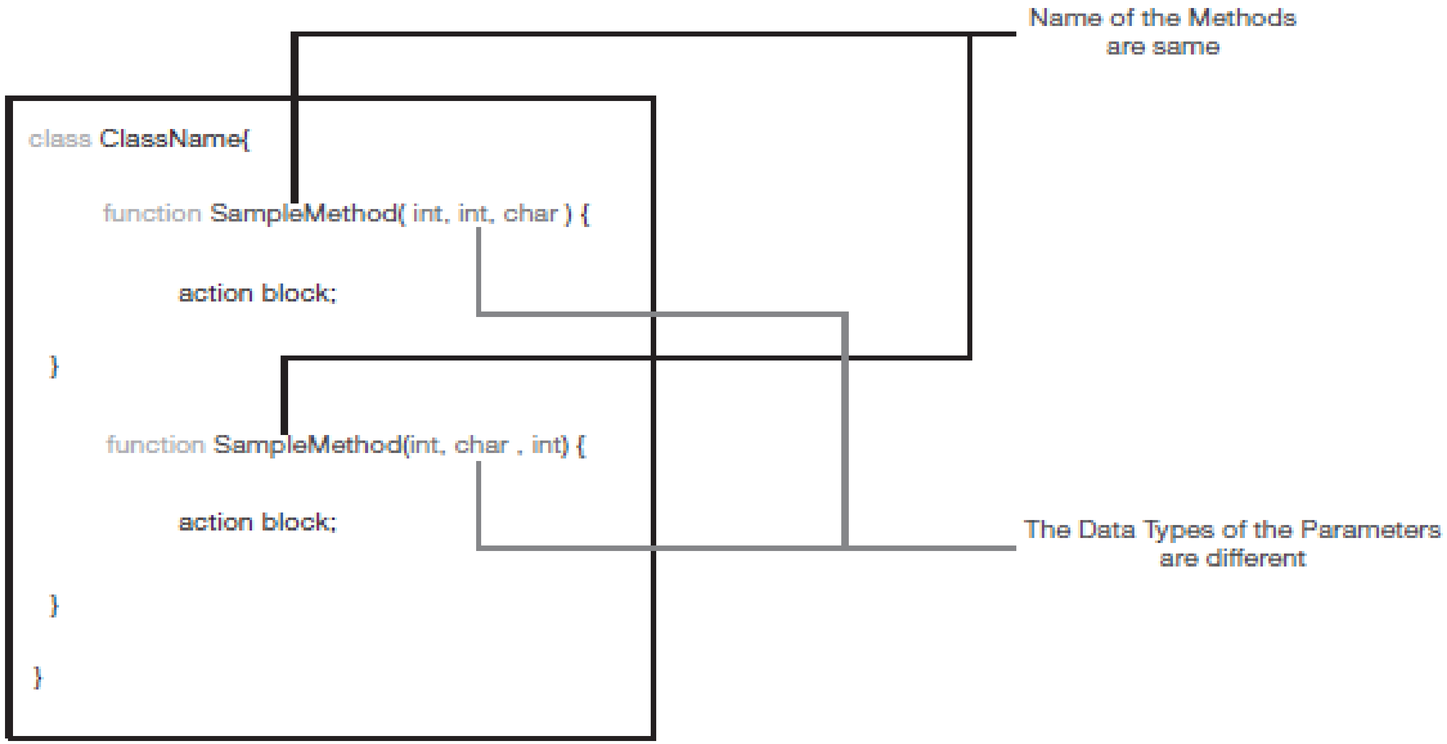
Operator Overloading

Method Overloading

- Method Overloading is adding additional functionalities to a method of a class by altering the parameters of the methods.
- Method Overloading is implemented by defining multiple methods of the same data with

1. Varying data types of the parameters
2. A varying number of parameters.

Method Overloading with parameters of Varying Types.

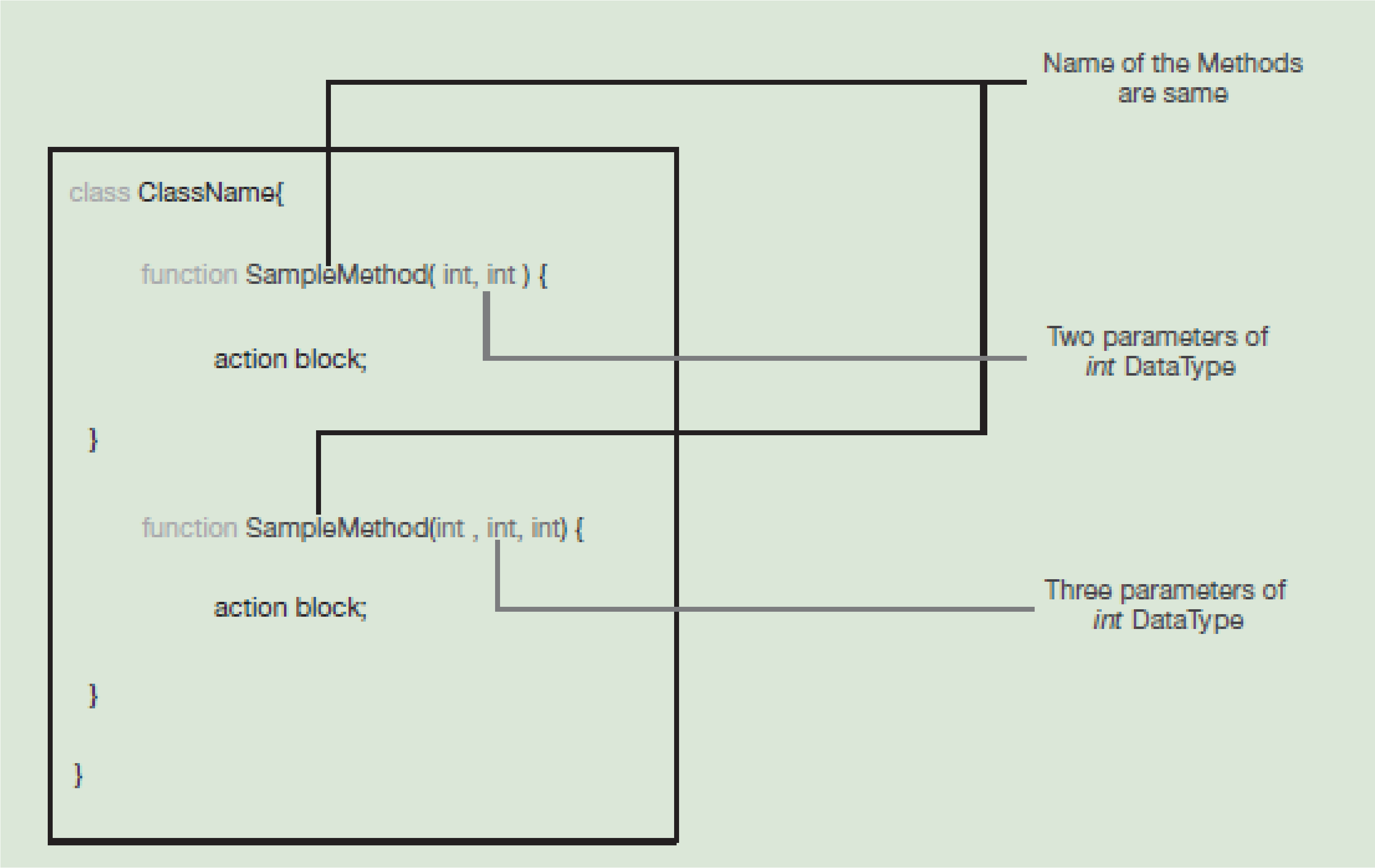


```
class ClassName{

        function SampleMethod( int, int, char ) {

                action block;

  }

        function SampleMethod(int, char , int) {

                action block;

  }

}
```

Name of the Methods
are same

The Data Types of the Parameters
are different

Method Overloading with parameters varying Number of Parameters.



```
class ClassName{

    function SampleMethod( int, int ) {

        action block;

  }

    function SampleMethod(int , int, int) {

        action block;

  }

}
```

Name of the Methods are same

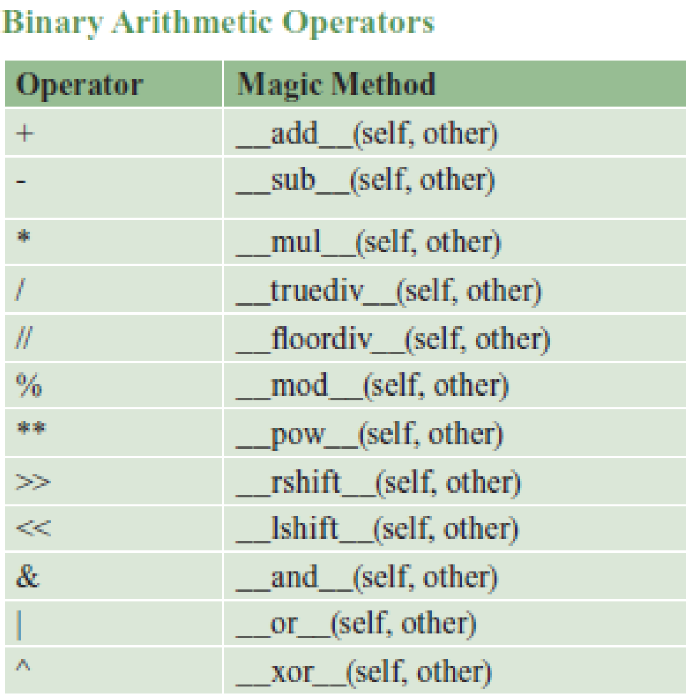
Two parameters of *int* DataType

Three parameters of *int* DataType

Operator Overloading

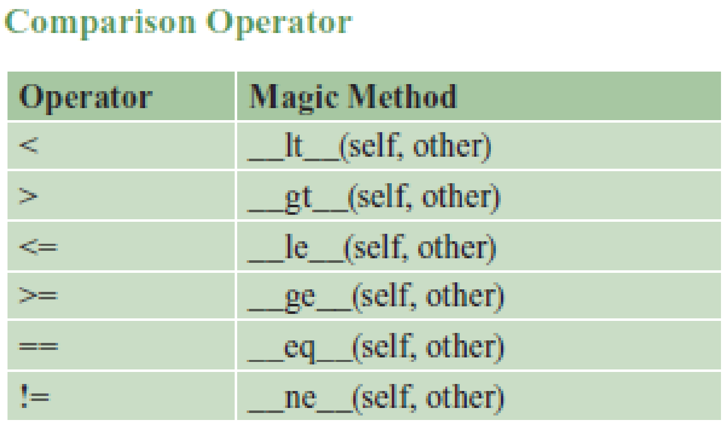
- Operator Overloading is an essential process in OOP, which adds multiple functions to the available operators.
- Operator overloading is the proves of extending the predefined function of an operator to user defined functions.
- For example, the Operator”+” is used to add two integers, join two strings, and merge two lists.
- The operator ‘+’ is used for multiple purposes and is thus called operator overloading.

Magic Method



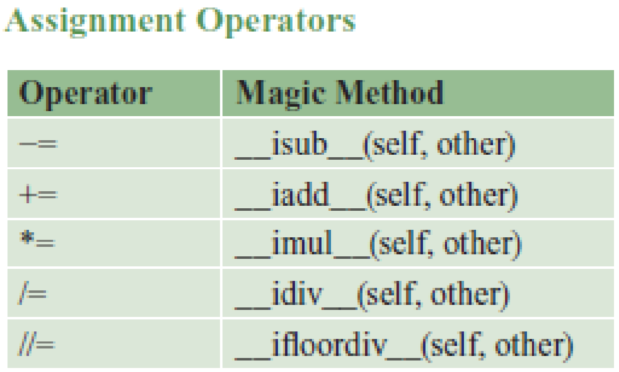
Binary Arithmetic Operators

Operator	Magic Method
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)
>>	__rshift__(self, other)
<<	__lshift__(self, other)
&	__and__(self, other)
|	__or__(self, other)
^	__xor__(self, other)



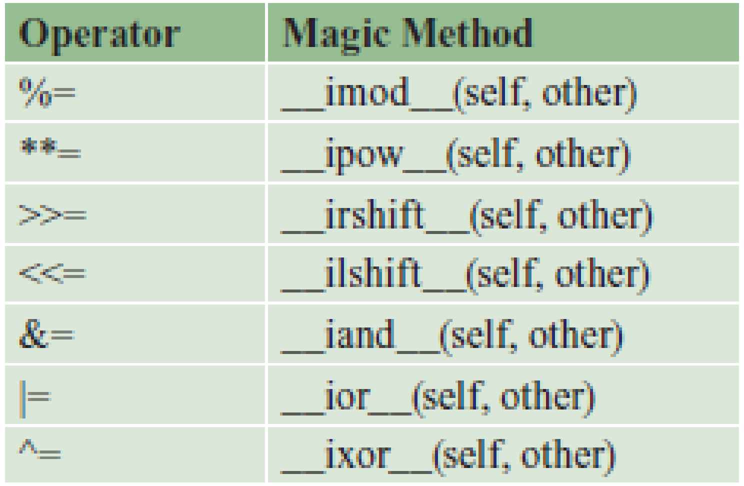
Comparison Operator

Operator	Magic Method
<	__lt__(self, other)
>	__gt__(self, other)
<=	__le__(self, other)
>=	__ge__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)

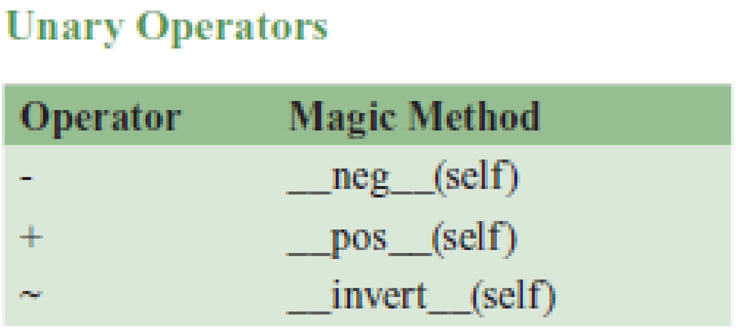


Assignment Operators

Operator	Magic Method
-=	__isub__(self, other)
+=	__iadd__(self, other)
*=	__imul__(self, other)
/=	__idiv__(self, other)
//=	__ifloordiv__(self, other)



Operator	Magic Method
%=	__imod__(self, other)
**=	__ipow__(self, other)
>>=	__irshift__(self, other)
<<=	__ilshift__(self, other)
&=	__iand__(self, other)
|=	__ior__(self, other)
^=	__ixor__(self, other)



Unary Operators

Operator	Magic Method
-	__neg__(self)
+	__pos__(self)
~	__invert__(self)

Method Overriding

- Method Overriding is the process of defining a method that is already defined in their parents' class by modifying the properties of predecessors.
- Method overriding is an ability of any OOP language that allows a child class to provide a specific implementation of a method that is already provided by one of its parent classes or super classes.
- A child class inherited from parent class(es) needs to possess
- different properties for the methods with the same name in the parent class.
- The property of showing different characteristics in the child class and parent class is necessary for implementing complex problems.

Method Overriding in Multiple Inheritance

Multiple inheritance is the process of deriving a class from more than one class. This section discusses the method overriding in multiple inheritance. Let us consider the following example.

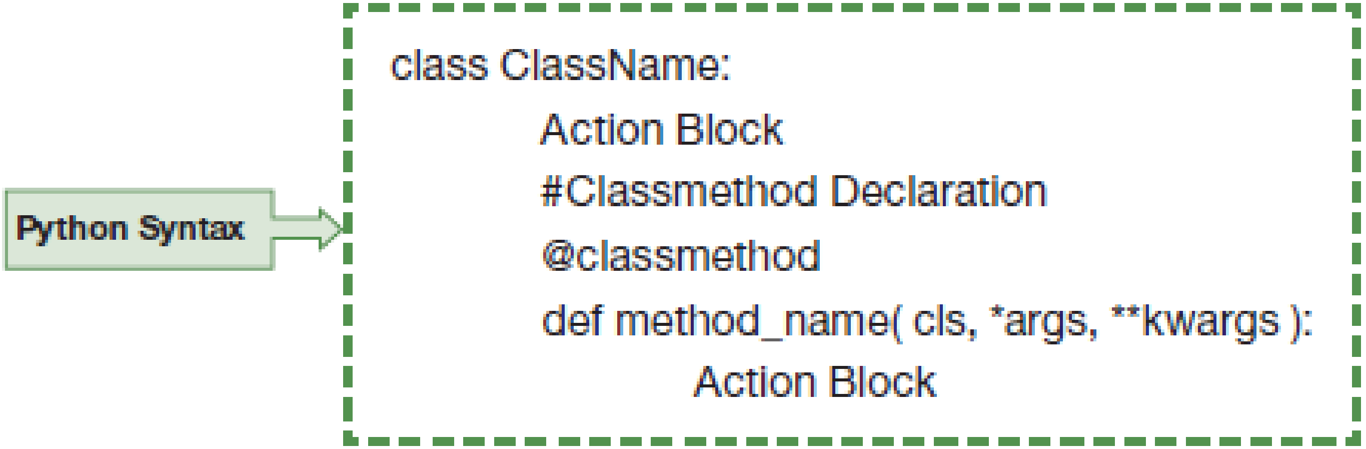
Illustration for method overriding in multiple inheritance.

Class A is defined with the method *printMessage* as follows.

```
>>> #Defining Parent class A
>>> class A:
... def printMessage(self):
... print("Parent Class A")
...
>>>
```

Class Method and Static Method

- The methods is a class are associated with the objects created for it.
- For invoking the methods defined inside the method, the presence of an instance is necessary.
- The classmethod is a method that is also defined inside the class but does not need any object invoke it.

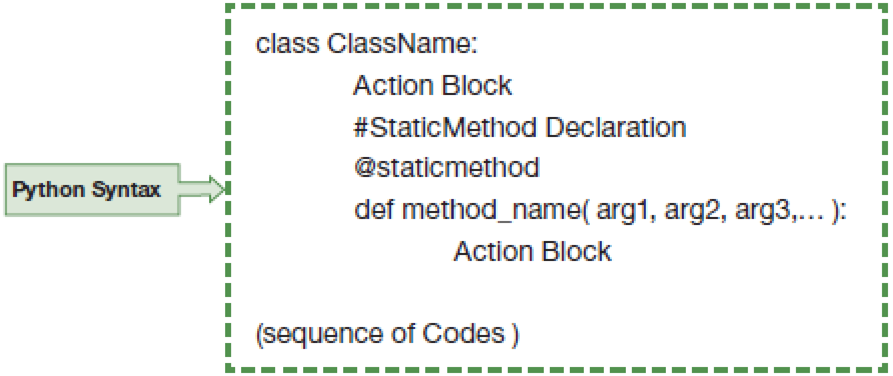


Python Syntax

```
class ClassName:
        Action Block
        #Classmethod Declaration
        @classmethod
        def method_name( cls, *args, **kwargs ):
                Action Block
```

Static Method

- Static method is similar to a class method, which can be accessed without an object.
- A static method is accessible to every object of a class, but methods defined in an instance are only able to be accessed by the object of a class.
- Static methods are nor allowed to access the state of the objects.
- The syntax for the static method is as follows:


Python Syntax

```
class ClassName:
        Action Block
        #StaticMethod Declaration
        @staticmethod
        def method_name( arg1, arg2, arg3,... ):
                Action Block

(sequence of Codes )
```

Differences between class method and static method

Class method

- “cls” is the first argument in the function.
- Data in the class attributed can be accessed and altered.
- Class as a parameter so data can be altered.
- “@classmethod” is used for converting a method to classmethod

Static Method

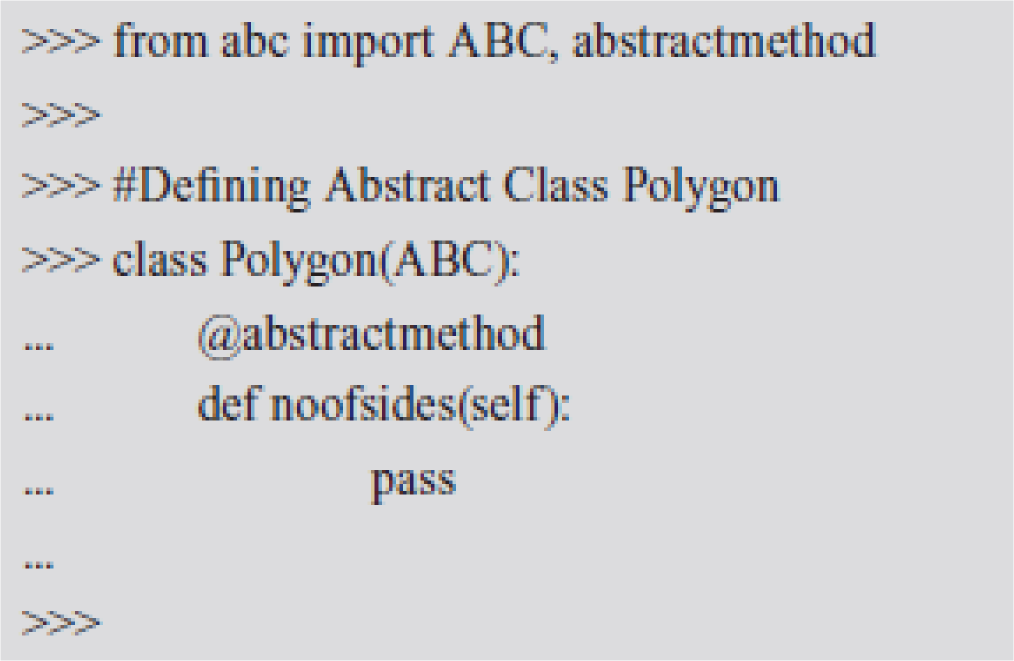
- No specific parameter is needed
- Data in the class attributed can not be acceded by a static method.
- Static methods are utility functions that can take action on parameters alone.
- “@staticmethod” is used for converting a method to a static method.

Abstract Base Class(ABC)

- Abstraction is the process of hiding unwanted information from accessing the data, and abstract class is considered a blueprint for other classes.
- A class that contains one or more abstract methods is called an abstract class.
- An abstract method is a class method that has the declaration in the abstract class but is defined in its derived class.
- The ABC module works by decorating methods of the base class with decorator @abstractmethod and then registering concrete classes as implementations of the abstract base.

Illustration for Abstract Base Class (ABC).

- Let's have an example of declaring and printing the sides of the polynomials with the concept of an abstract class.

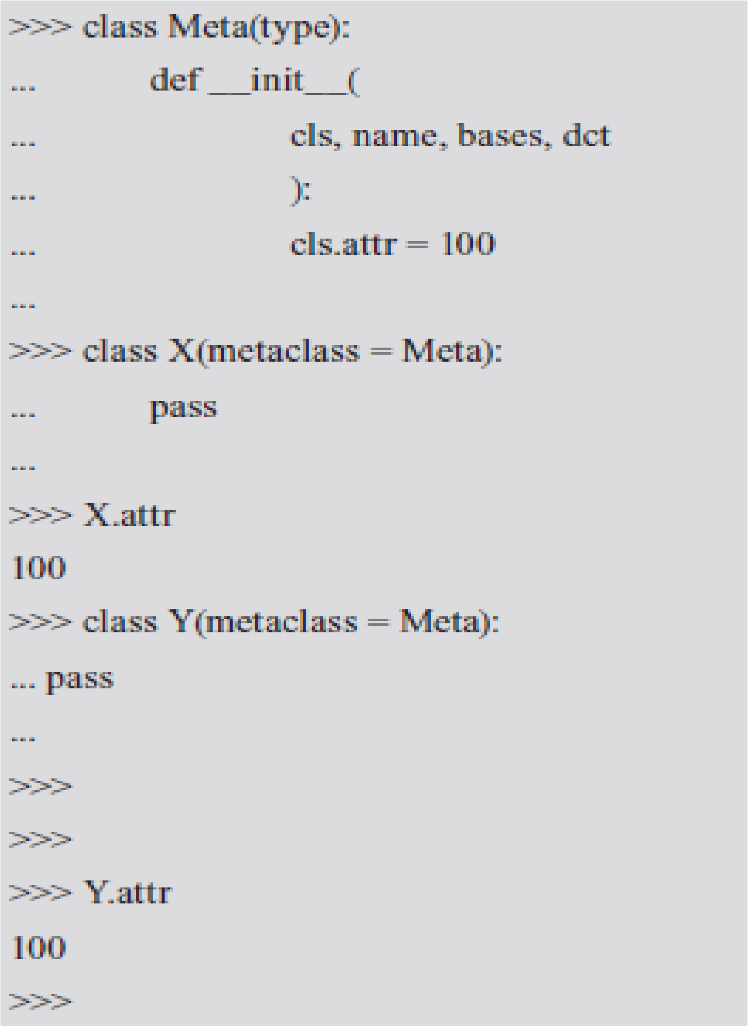


```
>>> from abc import ABC, abstractmethod
>>>
>>> #Defining Abstract Class Polygon
>>> class Polygon(ABC):
...         @abstractmethod
...         def noofsides(self):
...                     pass
...
>>>
```

Meta Class

- Meta programming is another programming paradigm where a computer program can treat another program as its data where reading, generating, analysing, and transforming other programs are possible.
- Meta programming also involves modifying the program itself. The primary idea of metaprogramming is to address the new situation without going for the recompilation of data.
- In OOPs, instances of meta classes are classes.

Illustration for Meta class



```
>>> class Meta(type):
...         def __init__(
...                     cls, name, bases, dct
...                     ):
...                     cls.attr = 100
...
>>> class X(metaclass = Meta):
...         pass
...
>>> X.attr
100
>>> class Y(metaclass = Meta):
... pass
...
>>>
>>>
>>> Y.attr
100
>>>
```

- In the above program, a class name "Meta" (could be other names also) with the argument of "type" is defined using the code "class Meta (type)". The metaclass consists of a constructor that holds an attribute of attr with the value of 100. Then, class X and class Y are derived from the metaclass "Meta" with the "metaclass = Meta" argument. While executing "X.attr", it displays the value of 100. The following things can be observed from the above program.
- The "Meta" class acts as a Template for the Classes X and Y.
- Being the template, the attributes are also reflected in classes X and Y, which is why the attributes are called without declaring any object for the class.