python

Course Id :INT 213

OOPs

In OOPS, everything is an object. In real life, some objects will have similar behavior. For example, all birds have similar behavior like having two wings, two legs, etc. Also, all birds have the ability to fly in the sky. Such objects with similar behavior belong to the same class. So, a class represents common behavior of a group of objects. Since a class represents behavior, it does not exist physically. But objects exist physically. For example, bird is a class; whereas, sparrow, pigeon, crow and peacock are objects of the bird class. Similarly, human being is a class and Arjun, Krishna, Sita are objects of the human being class.

Features of Object Oriented Programming

There are five important features related to Object Oriented Programming System. They are:

- Classes and objects
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Classes and Objects

The entire OOPS methodology has been derived from a single root concept called ‘object’. An object is anything that really exists in the world and can be distinguished from others. This definition specifies that everything in this world is an object. For example, a table, a ball, a car, a dog, a person, etc. will come under objects. Then what is not an object? If something does not really exist, then it is not an object. For example, our thoughts, imagination, plans, ideas etc. are not objects, because they do not physically exist.

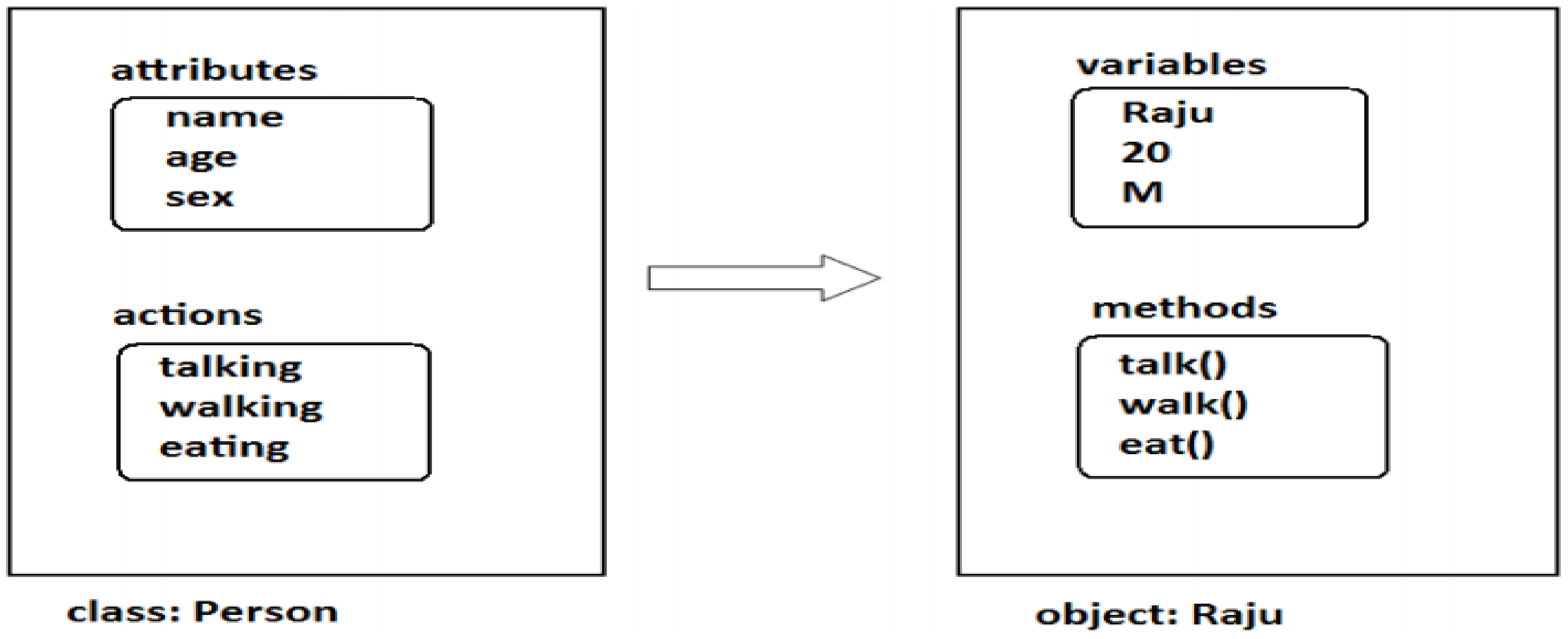
Classes and Objects

Every object has some behavior. The behavior of an object is represented by attributes and actions. For example, let’s take a person whose name is ‘Raju’. Raju is an object because he exists physically. He has attributes like name, age, sex, etc. These attributes can be represented by variables in our programming. For example, ’name’ is a string type variable, ‘age’ is an integer type variable.

Similarly, Raju can perform some actions like talking, walking, eating and sleeping. We may not write code for such actions in programming. But, we can consider calculations and processing of data as actions. These actions are performed by methods. We should understand that a function written inside a class is called a method. So an object contains variables and methods.

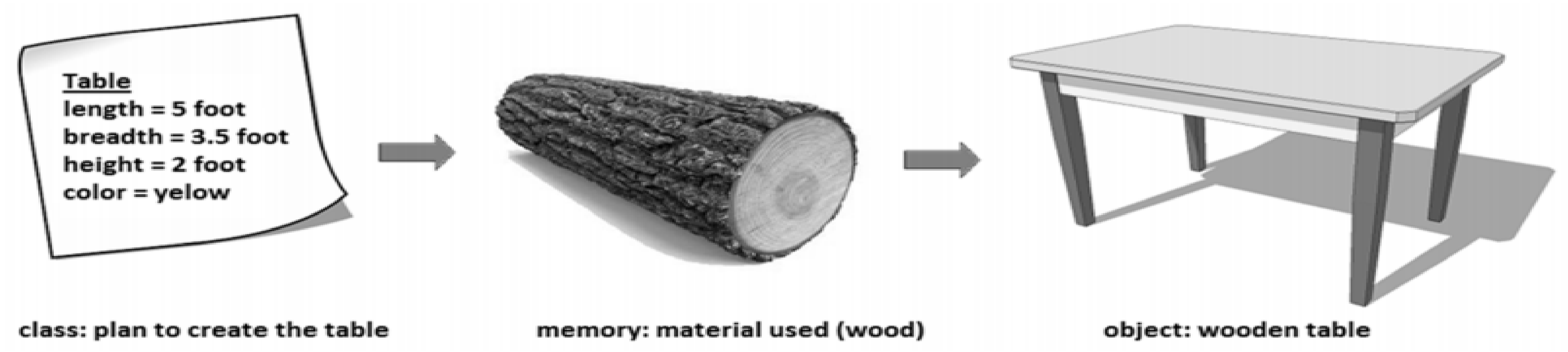
Classes and Objects

It is possible that some objects may have similar behavior. Such objects belong to same category called a 'class'. For example, not only Raju, but all the other persons have various common attributes and actions. So they are all objects of same class, 'Person'. Now observe that the 'Person' will not exist physically but only Raju, Ravi, Sita, etc. exist physically. This means, a class is a group name and does not exist physically, but objects exist physically. See Figure 12.3:

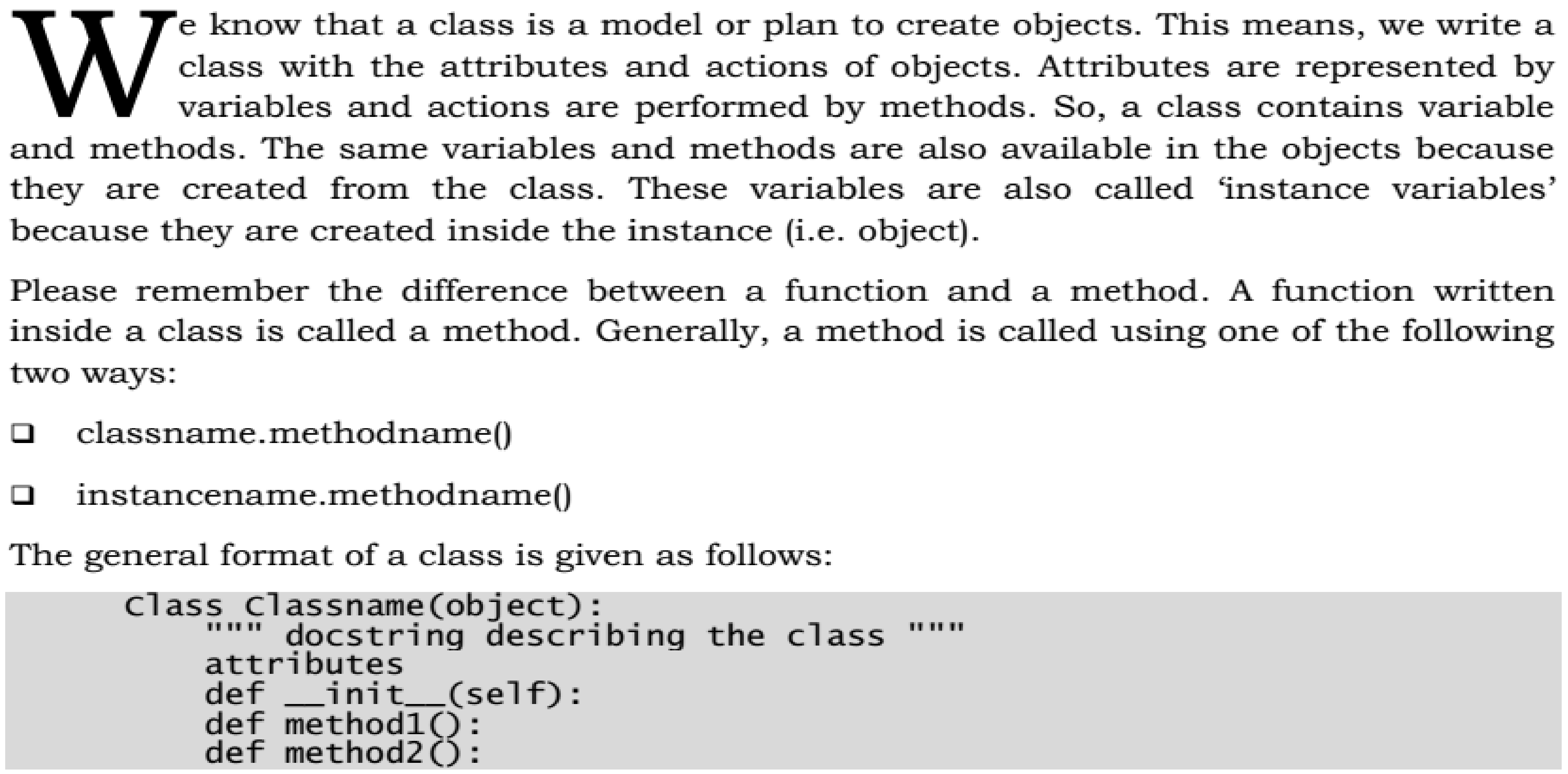


Classes and Objects

Let's take another example. We want a table made by a carpenter. First of all, the carpenter takes a paper and writes the measurements regarding length, breadth and height of the table. He may also draw a picture on the paper that works like a model for creating the original table. This plan or model is called a 'class'. Following this model, he makes the table that can be used by us. This table is called an 'object'. To make the table, we need material, i.e. wood. The wood represents the memory allotted by the PVM for the objects. Remember, objects are created on heap memory by PVM at run time. See Figure 12.4. It is also possible to create several objects (tables) from the same class (plan). An object cannot exist without a class. But a class can exist without any object. We can think that a class is a model and if it physically appears, then it becomes an object. So an object is called 'instance' (physical form) of a class.



Classes and Objects



We know that a class is a model or plan to create objects. This means, we write a class with the attributes and actions of objects. Attributes are represented by variables and actions are performed by methods. So, a class contains variable and methods. The same variables and methods are also available in the objects because they are created from the class. These variables are also called 'instance variables' because they are created inside the instance (i.e. object).

Please remember the difference between a function and a method. A function written inside a class is called a method. Generally, a method is called using one of the following two ways:

- classname.methodname()
- instancename.methodname()

The general format of a class is given as follows:

```
Class Classname(object):
    """ docstring describing the class """
    attributes
    def __init__(self):
    def method1():
    def method2():
```

Creating a class

A class is created with the keyword *class* and then writing the Classname. After the Classname, ‘object’ is written inside the Classname. This ‘object’ represents the base class name from where all classes in Python are derived. Even our own classes are also derived from ‘object’ class. Hence, we should mention ‘object’ in the parentheses. Please note that writing ‘object’ is not compulsory since it is implied.

The docstring is a string which is written using triple double quotes or triple single quotes that gives the complete description about the class and its usage. The docstring is used to create documentation file and hence it is optional. ‘attributes’ are nothing but variables that contains data. __init__(self) is a special method to initialize the variables. method1() and method2(), etc. are methods that are intended to process variables.

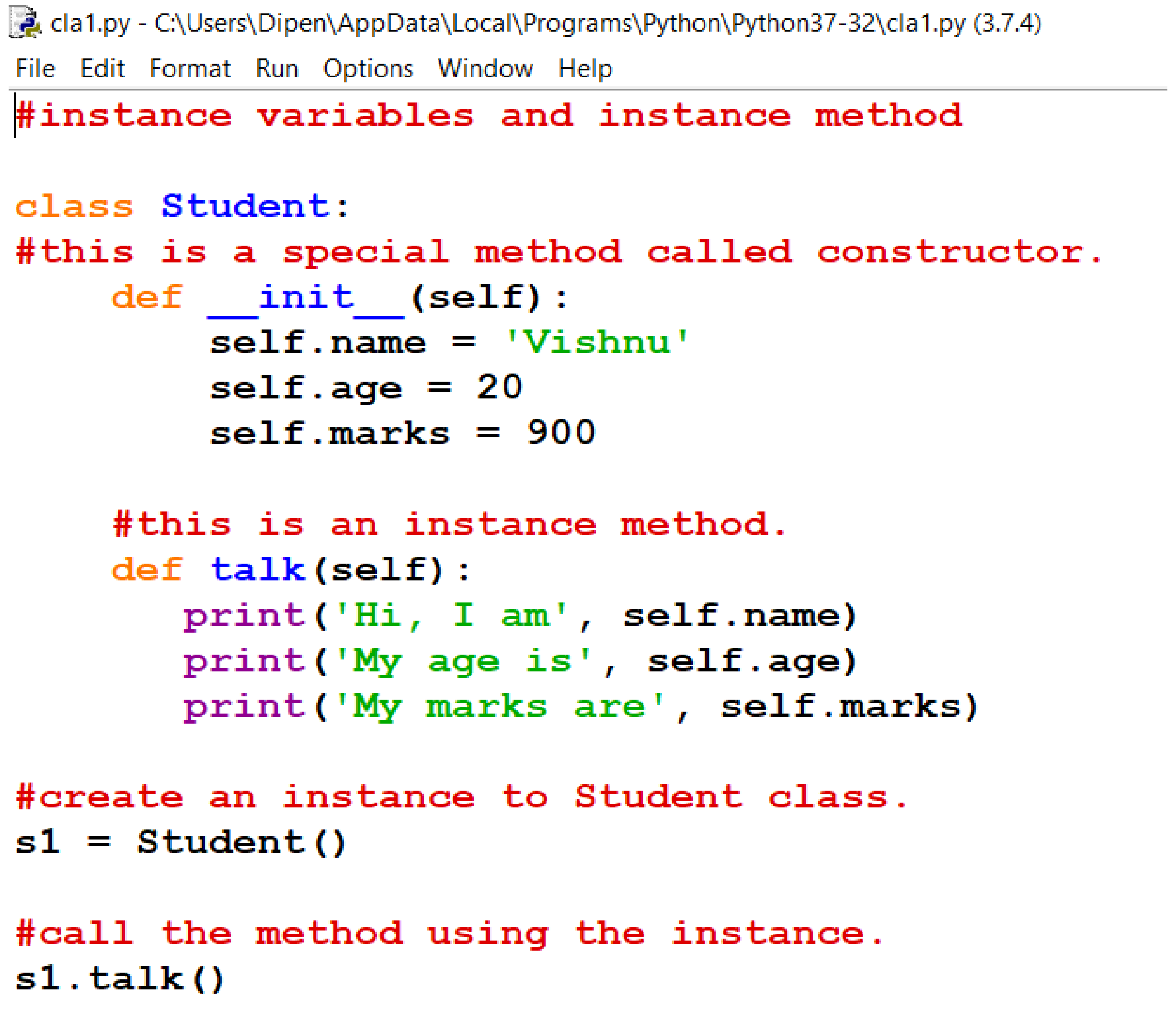
Creating a class

If we take ‘Student’ class, we can write code in the class that specifies the attributes and actions performed by any student. For example, a student has attributes like name, age, marks, etc. These attributes should be written inside the Student class as variables. Similarly, a student can perform actions like talking, writing, reading, etc. These actions should be represented by methods in the Student class. So, the class Student contains these attributes and actions, as shown here:

```
class Student:     # another way is: class Student(object):
    # the below block defines attributes
    def __init__(self):
        self.name = ‘Vishnu’
        self.age = 20
        self.marks = 900

    # the below block defines a method
    def talk(self):
        print(‘Hi, I am ‘, self.name)
        print(‘My age is’, self.age)
        print(‘My marks are’, self.marks)
```

Observe that the keyword *class* is used to declare a class. After this, we should write the class name. So, ‘Student’ is our class name. Generally, a class name should start with a capital letter, hence ‘S’ is capital in ‘Student’. In the class, we write attributes and methods. Since in Python, we cannot declare variables, we have written the variables inside a special method, i.e. __init__(). This method is useful to initialize the variables. Hence, the name ‘init’. The method name has two underscores before and after. This indicates that this method is internally defined and we cannot call this method explicitly. Observe the parameter ‘self’ written after the method name in the parentheses. ‘self’ is a variable that refers to current class instance. When we create an instance for the Student class, a separate memory block is allocated on the heap and that memory location is by default stored in ‘self’. The instance contains the variables ‘name’, ‘age’, ‘marks’ which are called *instance variables*. To refer to instance variables, we can use the dot operator notation along with self as: ‘self.name’, ‘self.age’ and ‘self.marks’.

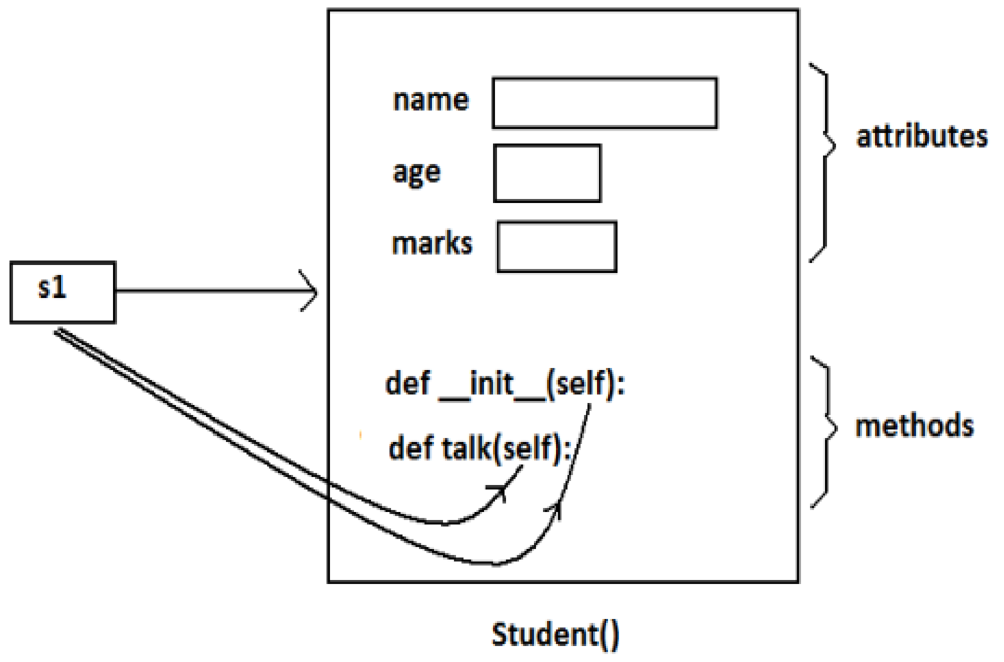
A Python program to define Student class and create an object to it. Also, we will call the method and display the student's details.



cla1.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla1.py (3.7.4)

File Edit Format Run Options Window Help

```
#instance variables and instance method

class Student:
#this is a special method called constructor.
    def __init__(self):
        self.name = 'Vishnu'
        self.age = 20
        self.marks = 900

    #this is an instance method.
    def talk(self):
       print('Hi, I am', self.name)
       print('My age is', self.age)
       print('My marks are', self.marks)

#create an instance to Student class.
s1 = Student()

#call the method using the instance.
s1.talk()
```



The Self variable

'self' is a default variable that contains the memory address of the instance of the current class. So, we can use 'self' to refer to all the instance variables and instance methods.

When an instance to the class is created, the instance name contains the memory location of the instance. This memory location is internally passed to 'self'. For example, we create an instance to Student class as:

```
      s1 = Student()
```

Here, 's1' contains the memory address of the instance. This memory address is internally and by default passed to 'self' variable. Since 'self' knows the memory address of the instance, it can refer to all the members of the instance. We use 'self' in two ways:

- The 'self' variable is used as first parameter in the constructor as:

```
      def __init__(self):
```

  In this case, 'self' can be used to refer to the instance variables inside the constructor.

- 'self' can be used as first parameter in the instance methods as:

```
      def talk(self):
```

  Here, talk() is instance method as it acts on the instance variables. If this method wants to act on the instance variables, it should know the memory location of the instance variables. That memory location is by default available to the talk() method through 'self'.

Constructor

A constructor is a special method that is used to initialize the instance variables of a class. In the constructor, we create the instance variables and initialize them with some starting values. The first parameter of the constructor will be 'self' variable that contains the memory address of the instance. For example,

```
def __init__(self):
    self.name = 'Vishnu'
    self.marks = 900
```

Here, the constructor has only one parameter, i.e. 'self'. Using 'self.name' and 'self.marks', we can access the instance variables of the class. A constructor is called at the time of creating an instance. So, the above constructor will be called when we create an instance as:

```
s1 = Student()
```

Here, 's1' is the name of the instance. Observe the empty parentheses after the class name 'Student'. These empty parentheses represent that we are not passing any values to the constructor. Suppose, we want to pass some values to the constructor, then we have to pass them in the parentheses after the class name. Let's take another example. We can write a constructor with some parameters in addition to 'self' as:

```
def __init__(self, n = '', m=0):
    self.name = n
    self.marks = m
```

Constructor

Here, the formal arguments are 'n' and 'm' whose default values are given as '' (None) and 0 (zero). Hence, if we do not pass any values to constructor at the time of creating an instance, the default values of these formal arguments are stored into name and marks variables. For example,

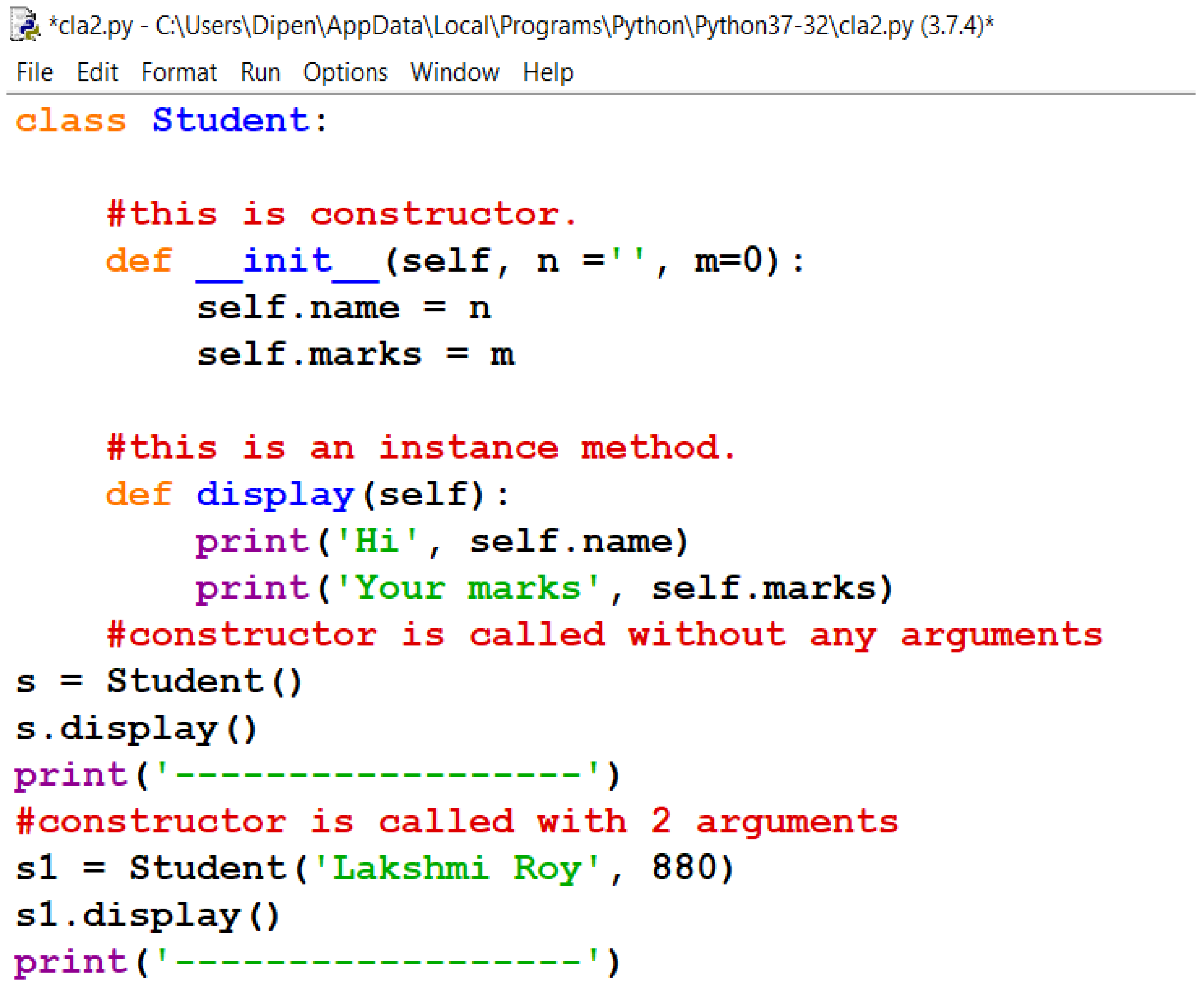
```
        s1 = Student()
```

Since we are not passing any values to the instance, None and zero are stored into name and marks. Suppose, we create an instance as:

```
        s1 = Student('Lakshmi Roy', 880)
```

In this case, we are passing two actual arguments: 'Lakshmi Roy' and 880 to the Student instance. Hence these values are sent to the arguments 'n' and 'm' and from there stored into name and marks variables. We can understand this concept from Program 2.

A Python program to create Student class with a constructor having more than one parameter.


cla2.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla2.py (3.7.4)
File Edit Format Run Options Window Help

```
class Student:

    #this is constructor.
    def __init__(self, n ='', m=0):
        self.name = n
        self.marks = m

    #this is an instance method.
    def display(self):
        print('Hi', self.name)
        print('Your marks', self.marks)
    #constructor is called without any arguments
s = Student()
s.display()
print('------------------')
#constructor is called with 2 arguments
s1 = Student('Lakshmi Roy', 880)
s1.display()
print('------------------')
```

We should understand that a constructor does not create an instance. The duty of the constructor is to initialize or store the beginning values into the instance variables. A constructor is called only once at the time of creating an instance. Thus, if 3 instances are created for a class, the constructor will be called once per each instance, thus it is called 3 times.

Types of variables

Instance variable

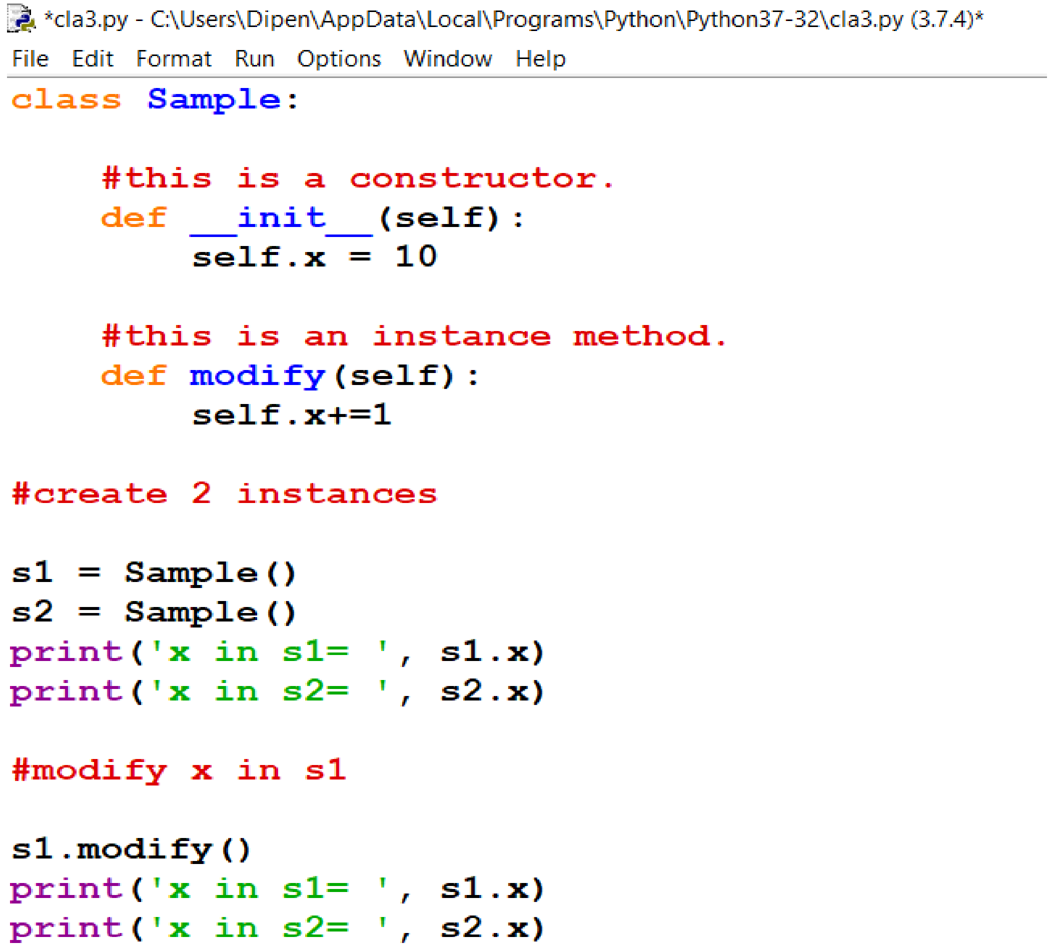
- Instance variables are the variables whose separate copy is created in every instance (or object). For example, if ‘x’ is an instance variable and if we create 3 instances, there will be 3 copies of ‘x’ in these 3 instances. When we modify the copy of ‘x’ in any instance, it will not modify the other two copies.

Class variable/ Static variable

- Unlike instance variables, class variables are the variables whose single copy is available to all the instances of the class. If we modify the copy of class variable in an instance, it will modify all the copies in the other instances. For example, if ‘x’ is a class variable and if we create 3 instances, the same copy of ‘x’ is passed to these 3 instances. When we modify the copy of ‘x’ in any instance using a class method, the modified copy is sent to the other two instances.

Types of variables

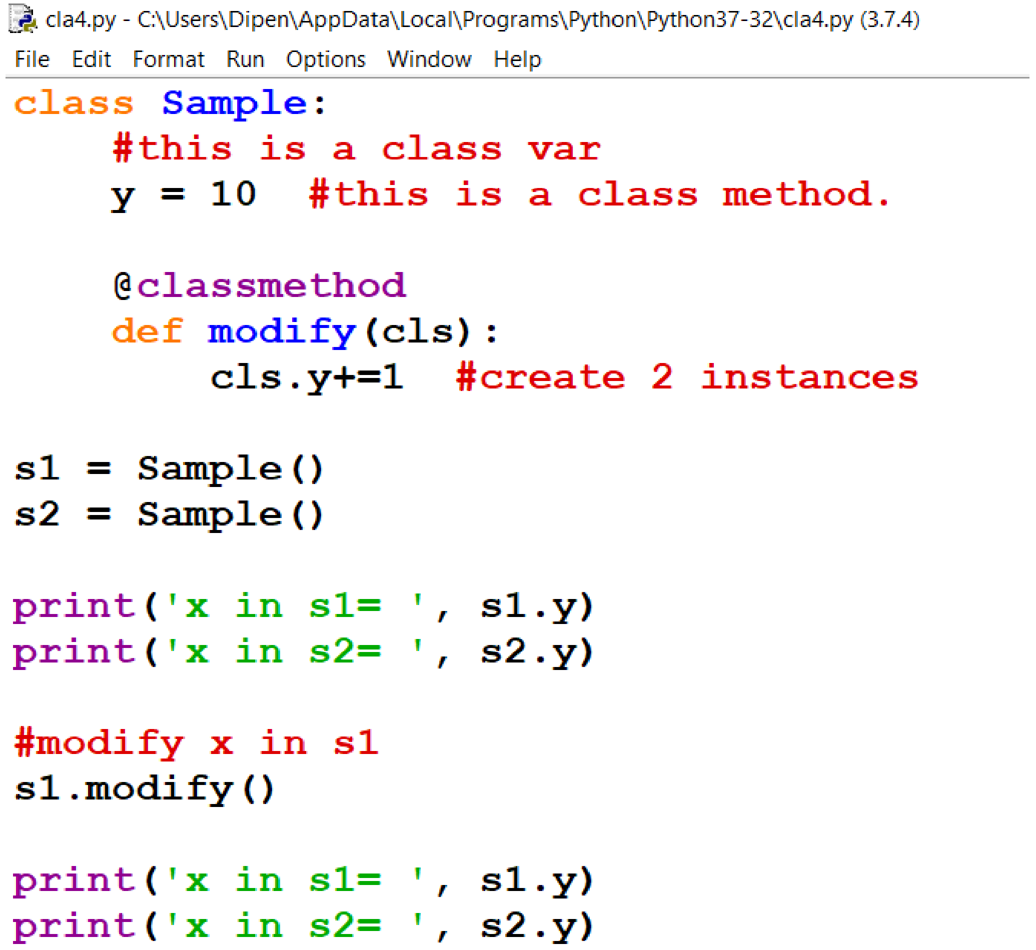
Instance variable



cla3.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla3.py (3.7.4)

```python
class Sample:

    #this is a constructor.
    def __init__(self):
        self.x = 10

    #this is an instance method.
    def modify(self):
        self.x+=1

#create 2 instances

s1 = Sample()
s2 = Sample()
print('x in s1= ', s1.x)
print('x in s2= ', s2.x)

#modify x in s1

s1.modify()
print('x in s1= ', s1.x)
print('x in s2= ', s2.x)
```

Class variable



cla4.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla4.py (3.7.4)

```python
class Sample:
    #this is a class var
    y = 10  #this is a class method.

    @classmethod
    def modify(cls):
        cls.y+=1  #create 2 instances

s1 = Sample()
s2 = Sample()

print('x in s1= ', s1.y)
print('x in s2= ', s2.y)

#modify x in s1
s1.modify()

print('x in s1= ', s1.y)
print('x in s2= ', s2.y)
```

Types of Methods

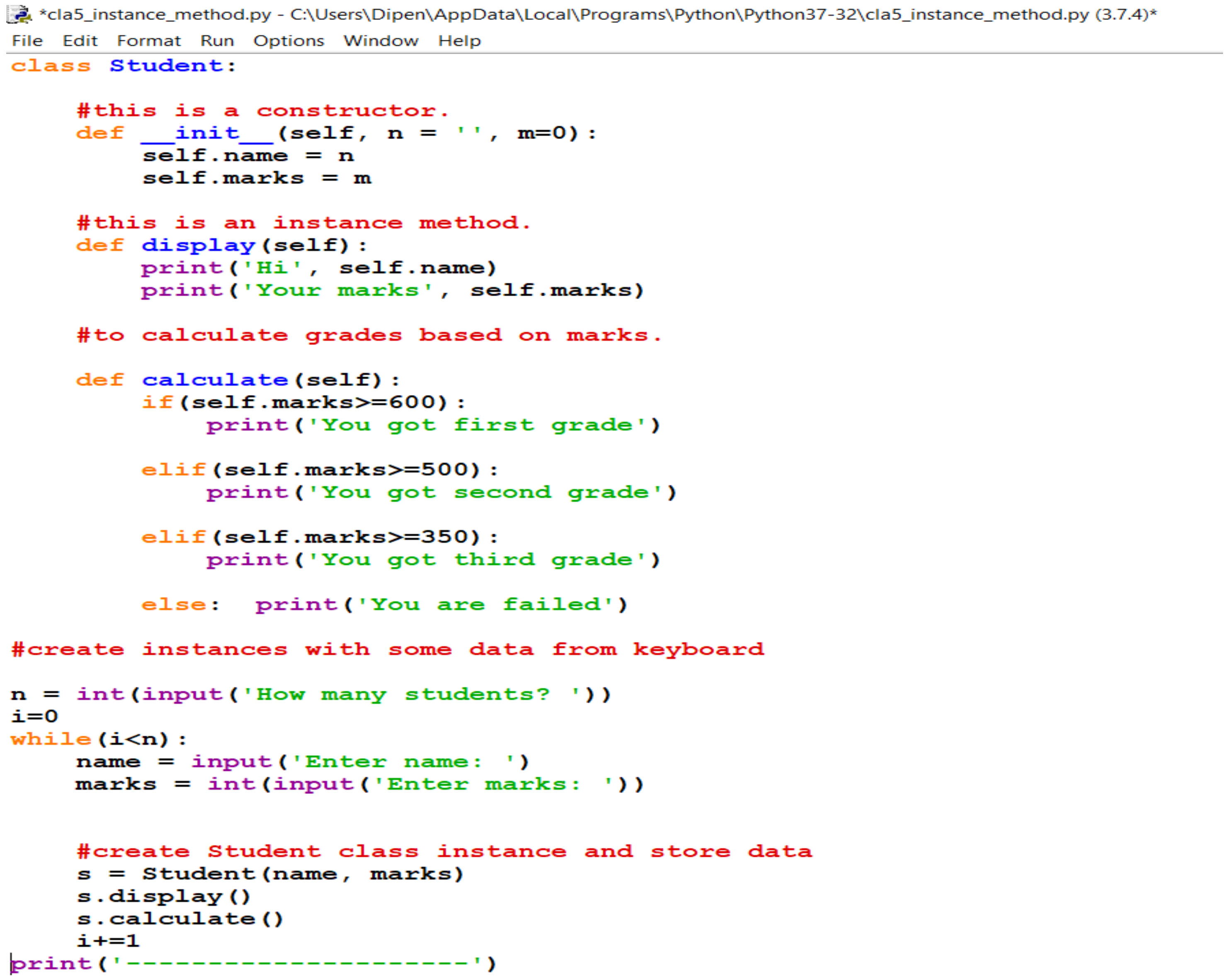
By this time, we got some knowledge about the methods written in a class. The purpose of a method is to process the variables provided in the class or in the method. We already know that the variables declared in the class are called class variables (or static variables) and the variables declared in the constructor are called instance variables. We can classify the methods in the following 3 types:

- Instance methods
  - (a) Accessor methods
  - (b) Mutator methods
- Class methods
- Static methods

Instance Variable

Instance methods are the methods which act upon the instance variables of the class. Instance methods are bound to instances (or objects) and hence called as: instancename.method(). Since instance variables are available in the instance, instance methods need to know the memory address of the instance. This is provided through ‘self’ variable by default as first parameter for the instance method. While calling the instance methods, we need not pass any value to the ‘self’ variable.

Python program using a student class with instance methods to process the data of several students.


cla5_instance_method.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla5_instance_method.py (3.7.4)
File Edit Format Run Options Window Help

```
class Student:

    #this is a constructor.
    def __init__(self, n = '', m=0):
        self.name = n
        self.marks = m

    #this is an instance method.
    def display(self):
        print('Hi', self.name)
        print('Your marks', self.marks)

    #to calculate grades based on marks.

    def calculate(self):
        if(self.marks>=600):
            print('You got first grade')

        elif(self.marks>=500):
            print('You got second grade')

        elif(self.marks>=350):
            print('You got third grade')

        else:  print('You are failed')

#create instances with some data from keyboard

n = int(input('How many students? '))
i=0
while(i<n):
    name = input('Enter name: ')
    marks = int(input('Enter marks: '))


    #create Student class instance and store data
    s = Student(name, marks)
    s.display()
    s.calculate()
    i+=1
print('---------------------')
```

Types of Instance Methods

- **Accessor methods** simply access or read data of the variables. They do not modify the data in the variables.
- Accessor methods are generally written in the form of getXXX() and hence they are also called getter methods. For example, def getName(self): return self.name
- Here, getName() is an accessor method since it is reading and returning the value of 'name' instance variable. It is not modifying the value of the name variable.

- **Mutator methods** are the methods which not only read the data but also modify them.
- They are written in the form of setXXX() and hence they are also called setter methods.

Python program to store data into instances using mutator methods and to retrieve data from the instances using accessor methods.

```
# accessor and mutator methods
class Student:

    # mutator method
    def setName(self, name):
        self.name = name

    # accessor method
    def getName(self):
        return self.name

    # mutator method
    def setMarks(self, marks):
        self.marks = marks

    # accessor method
    def getMarks(self):
        return self.marks

# create instances with some data from keyboard
n = int(input('How many students? '))

i=0
while(i<n):
    # create Student class instance
    s = Student()
    name = input('Enter name: ')
    s.setName(name)
    marks = int(input('Enter marks: '))
    s.setMarks(marks)

    # retrieve data from Student class instance
    print('Hi', s.getName())
    print('Your marks', s.getMarks())

    i+=1
    print('-------------------')
```

Class Methods

- These methods act on class level. Class methods are the methods which act on the class variables or static variables.
- These methods are written using @classmethod decorator above them.
- By default, the first parameter for class methods is ‘cls’ which refers to the class itself. For example, ‘cls.var’ is the format to refer to the class variable.
- These methods are generally called using the classname.method().
- The processing which is commonly needed by all the instances of the class is handled by the class methods. In the next Program , we are going to develop Bird class. All birds in the Nature have only 2 wings. So, we take ‘wings’ as a class variable. Now a copy of this class variable is available to all the instances of Bird class. The class method fly() can be called as Bird.fly().

Python program to use class method to handle the common feature of all the instances of Bird class.

```
# understanding class methods
class Bird:
    # this is a class var
    wings = 2

    # this is a class method
    @classmethod
    def fly(cls, name):
        print('{} flies with {} wings'.format(name, cls.wings))

# display information for 2 birds
Bird.fly('Sparrow')
Bird.fly('Pigeon')
```

Output:

```
C:\>python cl.py
Sparrow flies with 2 wings
Pigeon flies with 2 wings
```

Static Methods

- We need static methods when the processing is at the class level but we need not involve the class or instances.
- Static methods are used when some processing is related to the class but does not need the class or its instances to perform any work. For example, setting environmental variables, counting the number of instances of the class or changing an attribute in another class, etc. are the tasks related to a class. Such tasks are handled by static methods.
- Also, static methods can be used to accept some values, process them and return the result.
- In this case the involvement of neither the class nor the objects is needed. Static methods are written with a decorator @staticmethod above them. Static methods are called in the form of classname.method().

Python program to create a static method that counts the number of instances created for a class.

```
# understanding static methods
class Myclass:
    # this is class var or static var
    n=0

    # constructor that increments n when an instance is created
    def __init__(self):
        Myclass.n = Myclass.n+1

    # this is a static method to display the no. of instances
    @staticmethod
    def noObjects():
        print(‘No. of instances created: ‘, Myclass.n)

# create 3 instances
obj1 = Myclass()
obj2 = Myclass()
obj3 = Myclass()
Myclass.noObjects()
```

Passing Members of One Class to Another Class

It is possible to pass the members (i.e. attributes and methods) of a class to another class. Let's take an Emp class with a constructor that defines attributes 'id', 'name', and 'salary'. This class has an instance method display() to display these values. If we create an object (or instance) of Emp class, it contains a copy of all the attributes and methods. To pass all these members of Emp class to another class, we should pass Emp class instance to the other class. For example, let's create an instance of Emp class as:

```
        e = Emp()
```

Then pass this instance 'e' to a method of other class, as:

```
        Myclass.mymethod(e)
```

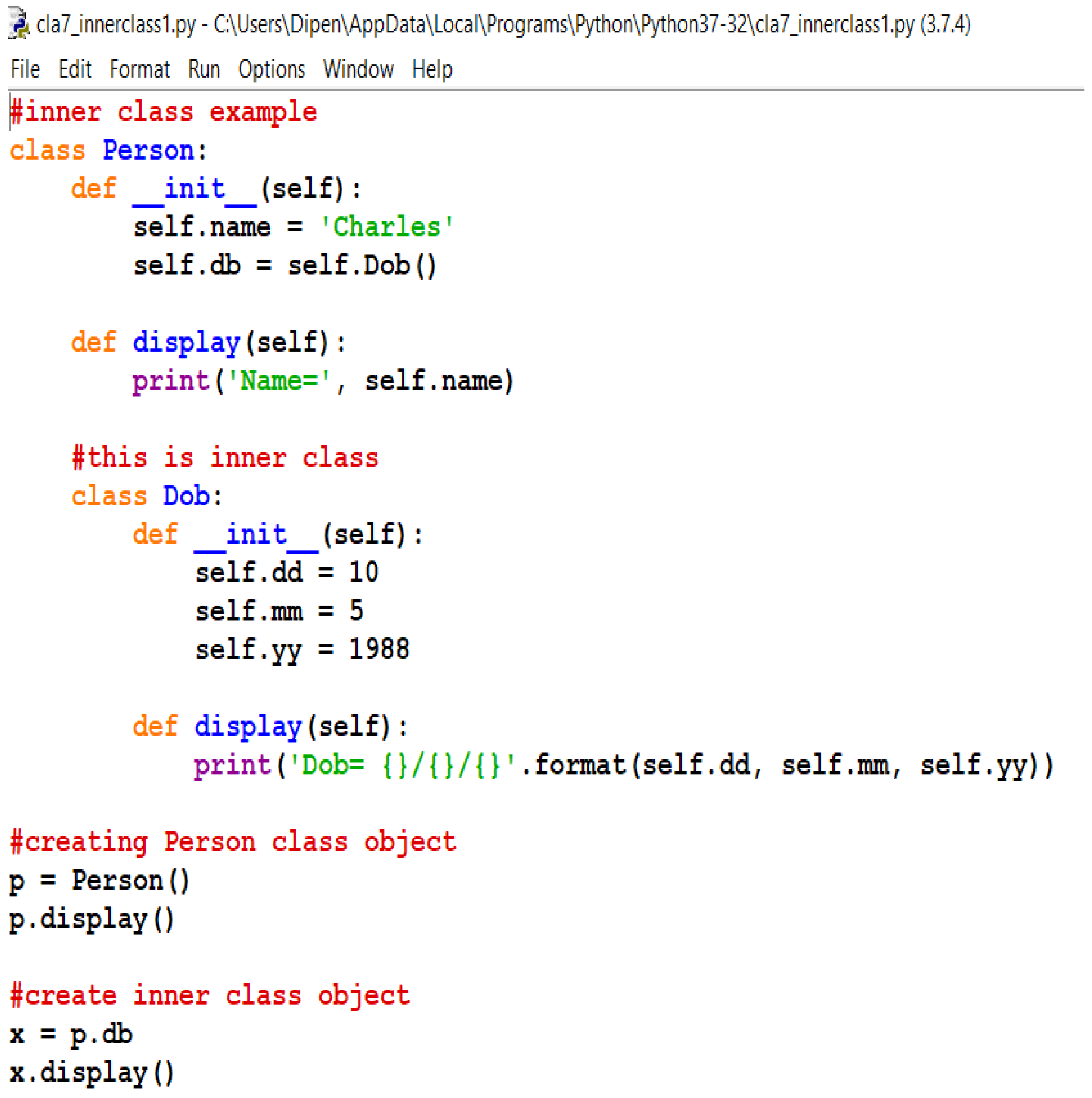
Passing Members of One Class to Another Class

- Here, Myclass is the other class and mymethod() is a static method that belongs to Myclass. In Myclass, the method mymethod() will be declared as a static method as it acts neither on the class variables nor instance variables of Myclass. The purpose of mymethod() is to change the attribute of Emp class
- So, the point is this: by passing the instance of a class, we are passing all the attributes and methods to another class. In the other class, it is possible to utilize them as needed. In our example, Myclass method, i.e. mymethod() is utilizing the salary attribute and display() methods of Emp class.
-

A Python program to create Emp class and make all the members of the Emp class available to another class, i.e. Myclass.

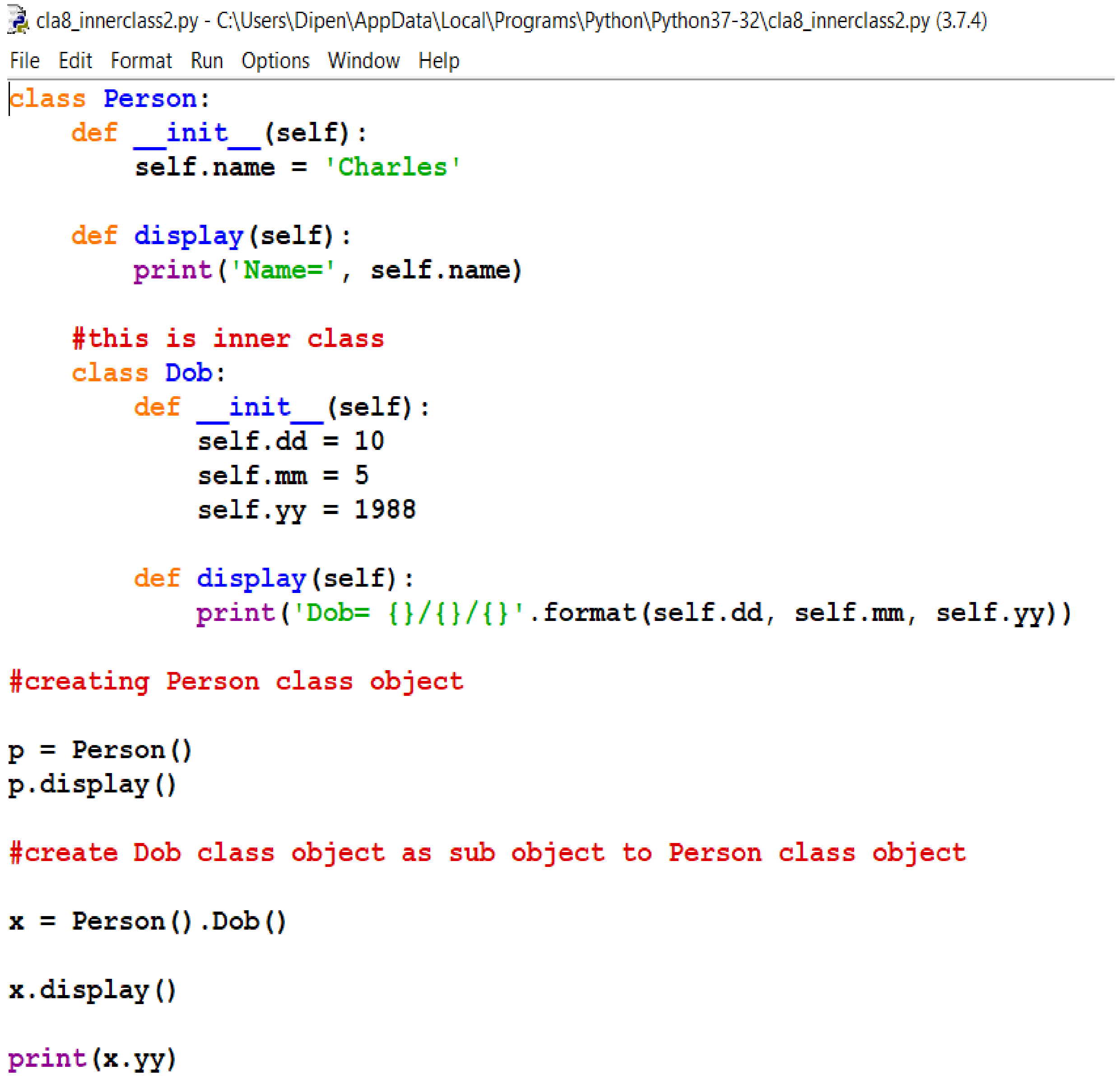
```python
# this class contains employee details
class Emp:
    # this is a constructor.
    def __init__(self, id, name, salary):
        self.id = id
        self.name = name
        self.salary = salary

    # this is an instance method.
    def display(self):
        print('Id=', self.id)
        print('Name=', self.name)
        print('Salary= ', self.salary)

# this class displays employee details
class Myclass:
    # method to receive Emp class instance
    # and display employee details
    @staticmethod
    def mymethod(e):
        # increment salary of e by 1000
        e.salary+=1000;
        e.display()

# create Emp class instance e
e = Emp(10, 'Raj kumar', 15000.75)
# call static method of Myclass and pass e
Myclass.mymethod(e)
```

Inner Class



cla7_innerclass1.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla7_innerclass1.py (3.7.4)

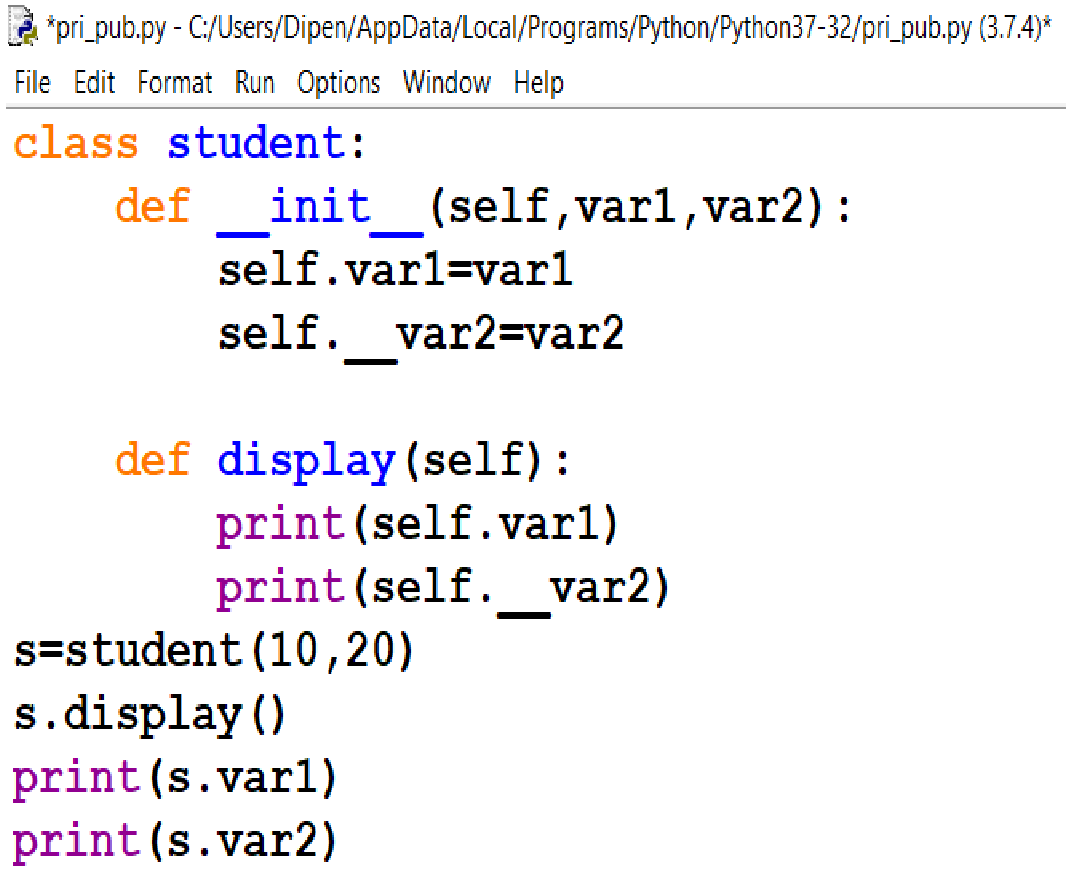
```
#inner class example
class Person:
    def __init__(self):
        self.name = 'Charles'
        self.db = self.Dob()

    def display(self):
        print('Name=', self.name)

    #this is inner class
    class Dob:
        def __init__(self):
            self.dd = 10
            self.mm = 5
            self.yy = 1988

        def display(self):
            print('Dob= {}/{}/{}'.format(self.dd, self.mm, self.yy))

#creating Person class object
p = Person()
p.display()

#create inner class object
x = p.db
x.display()
```



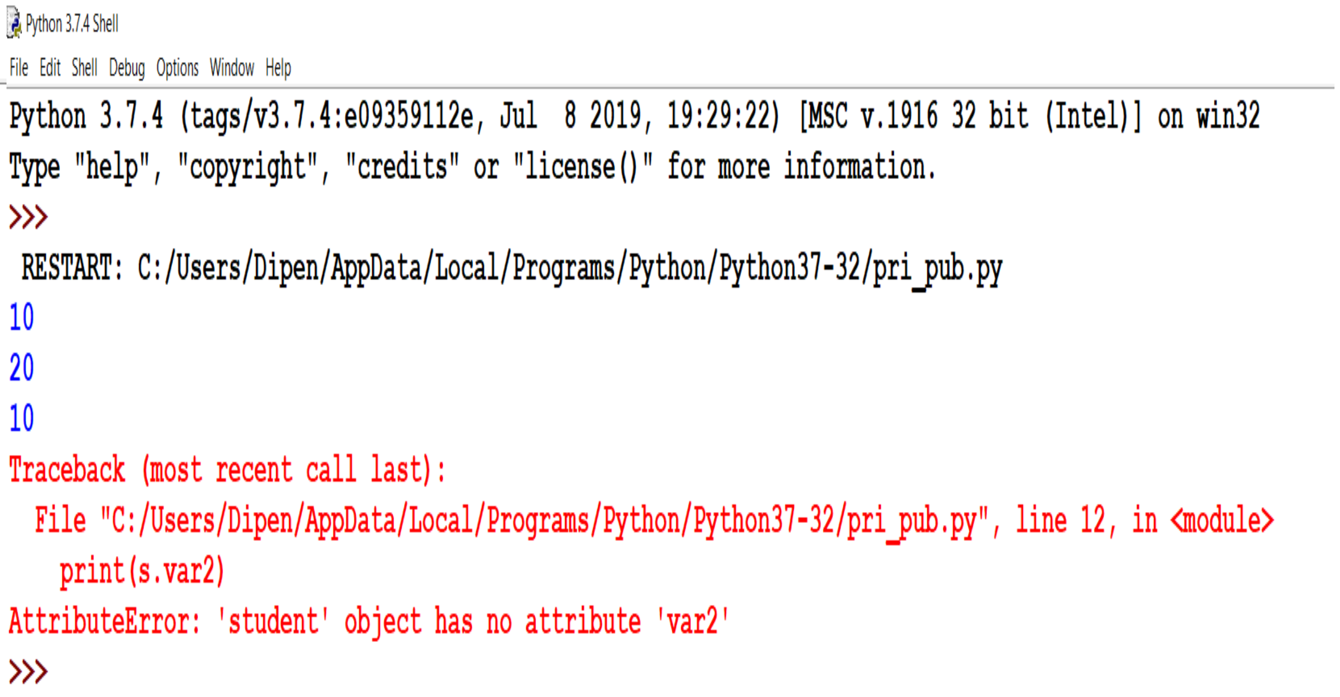
cla8_innerclass2.py - C:\Users\Dipen\AppData\Local\Programs\Python\Python37-32\cla8_innerclass2.py (3.7.4)

```
class Person:
    def __init__(self):
        self.name = 'Charles'

    def display(self):
        print('Name=', self.name)

    #this is inner class
    class Dob:
        def __init__(self):
            self.dd = 10
            self.mm = 5
            self.yy = 1988

        def display(self):
            print('Dob= {}/{}/{}'.format(self.dd, self.mm, self.yy))

#creating Person class object

p = Person()
p.display()

#create Dob class object as sub object to Person class object

x = Person().Dob()

x.display()

print(x.yy)
```

Public and Private Data Members

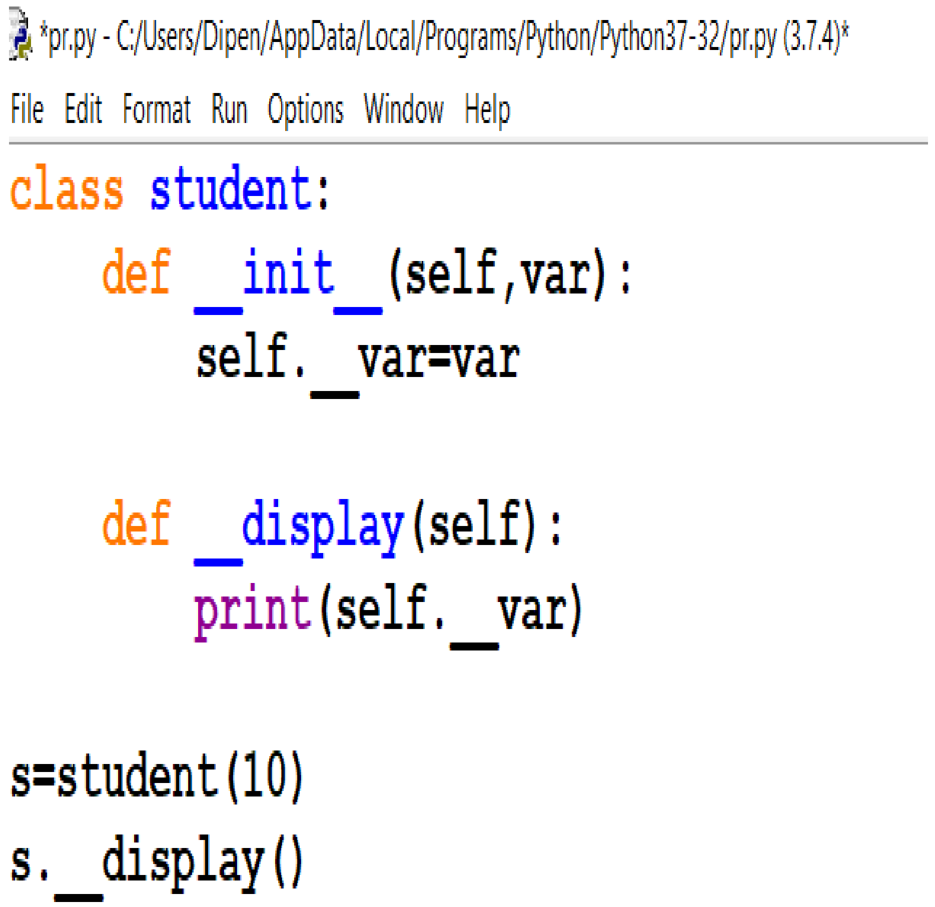

pri_pub.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pri_pub.py (3.7.4)
File Edit Format Run Options Window Help

```
class student:
    def __init__(self,var1,var2):
        self.var1=var1
        self.__var2=var2

    def display(self):
        print(self.var1)
        print(self.__var2)
s=student(10,20)
s.display()
print(s.var1)
print(s.var2)
```

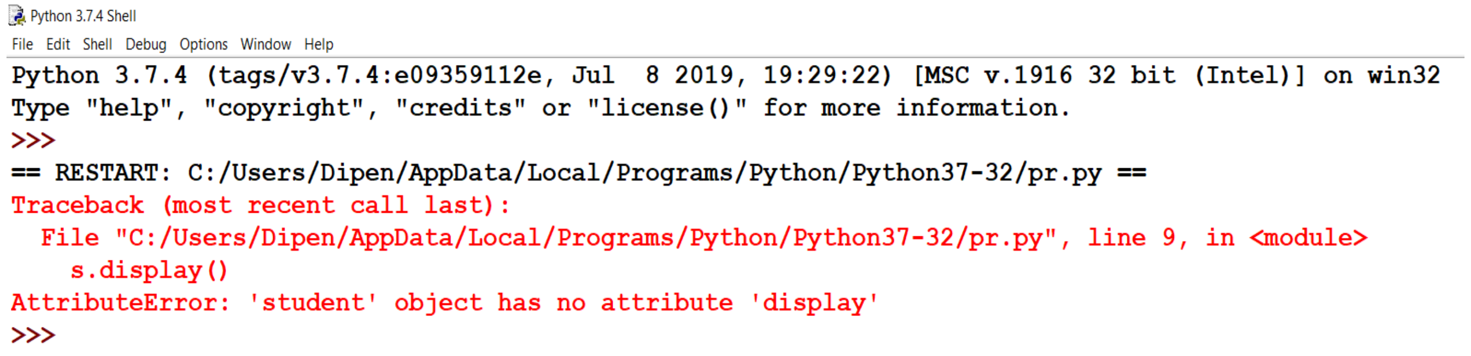


```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pri_pub.py
10
20
10
Traceback (most recent call last):
  File "C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pri_pub.py", line 12, in <module>
    print(s.var2)
AttributeError: 'student' object has no attribute 'var2'
>>>
```

Private Methods

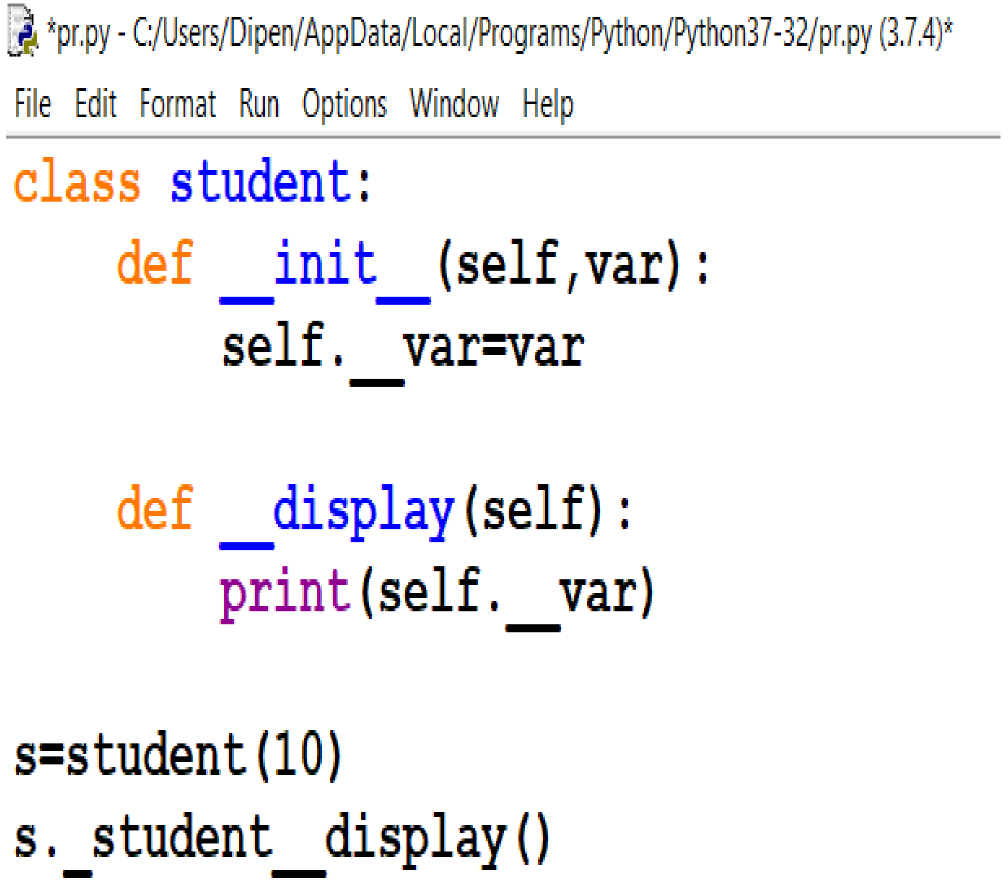

pr.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pr.py (3.7.4)
File Edit Format Run Options Window Help

```
class student:
    def __init__(self,var):
        self.__var=var

    def __display(self):
        print(self.__var)

s=student(10)
s.__display()
```

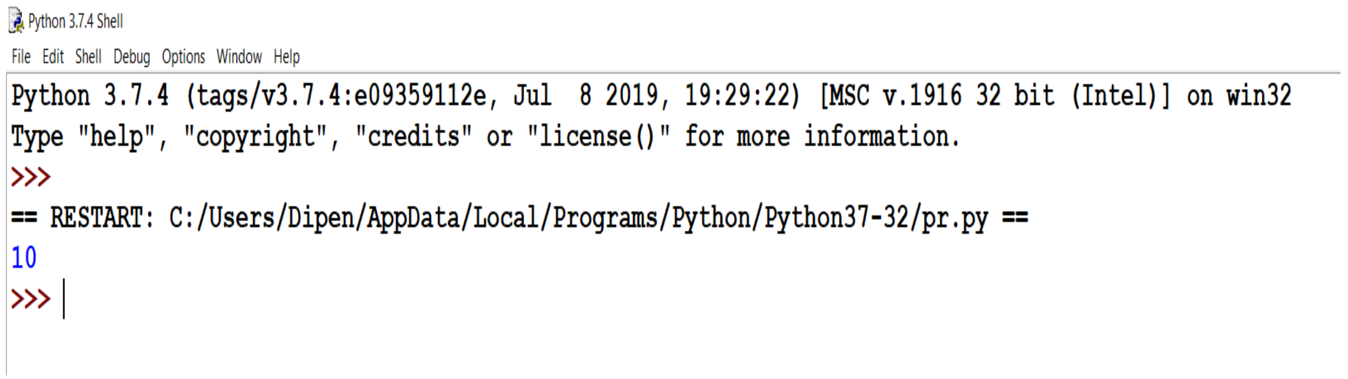


```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pr.py ==
Traceback (most recent call last):
  File "C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pr.py", line 9, in <module>
    s.display()
AttributeError: 'student' object has no attribute 'display'
>>>
```

Private Methods


pr.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pr.py (3.7.4)
File Edit Format Run Options Window Help

```
class student:
    def __init__(self,var):
        self.__var=var

    def __display(self):
        print(self.__var)

s=student(10)
s._student__display()
```



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/pr.py ==
10
>>> 
```

What is the relation between object and class?

a. A class is an instance of an object

b. An object is an instance of an object

c. An object is an attribute of a class

d. None of the above

What is used to create an object?

a. Constructor

b. Class

c. Method

d. None of the above

_________ represents an entity in the real world which can be distinctly identified?

a. Object

b. Class

c. Method

d. None of the above

Which of the following creates a new type?

(a) Class

(b) Object

(c) Attribute

(d) metthod

Which variables are used to keep a count of number of objects created from a class?

(a) Class

(b) Object

(c) Ordinary

(d) temporary

Method overriding is ____________________.

a. A method with different name,

b. A method in a subclass which has the same name and same header as that of the super class

c. Both a and b

d. None of the above