PYTHON PROGRAMMING (18MCA31C)

UNIT – V
Classes and Objects

FACULTY:
Dr. R. A. Roseline, M.Sc., M.Phil., Ph.D.,
Associate Professor and Head,
Post Graduate and Research Department of Computer Applications,
Government Arts College (Autonomous), Coimbatore – 641 018.

Python Object Oriented Programming

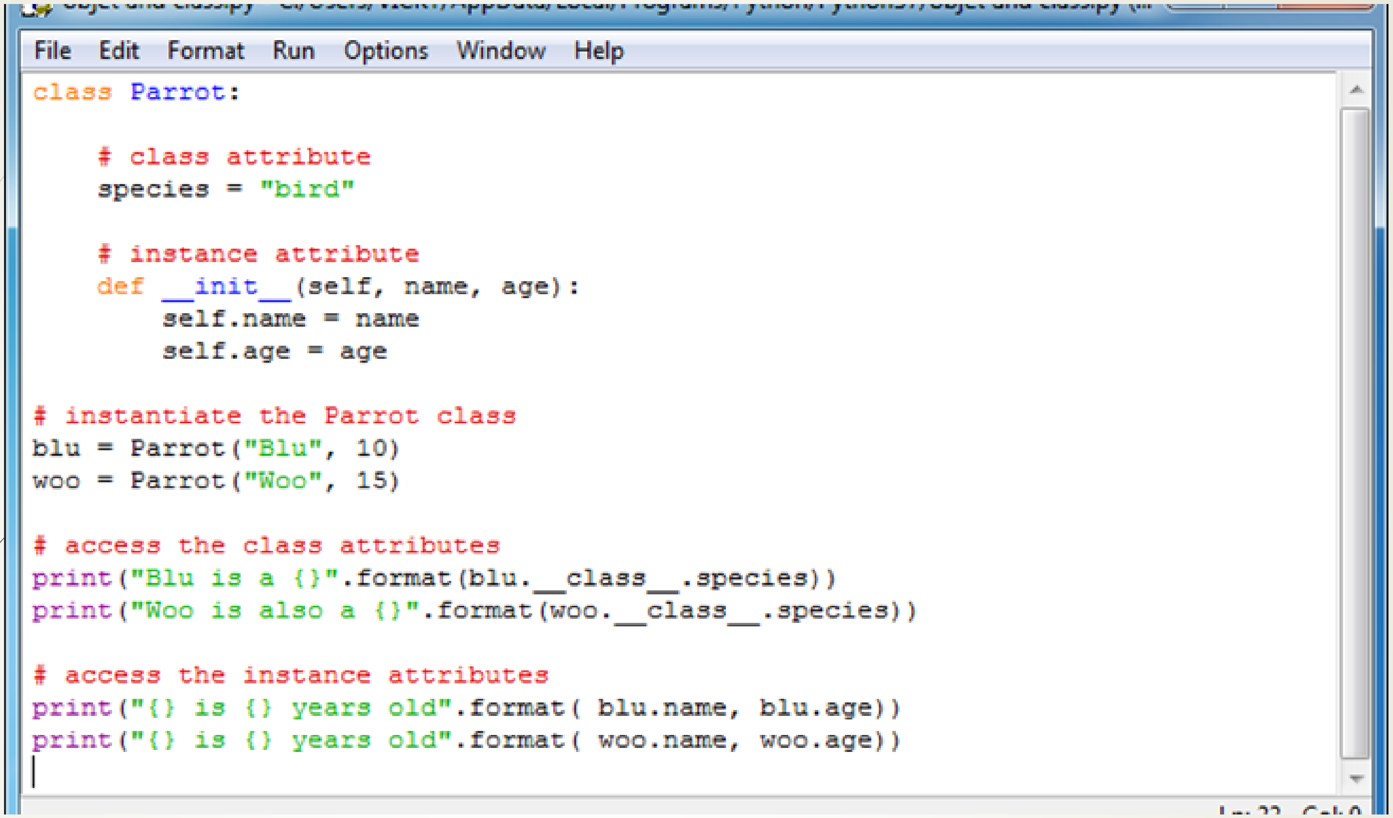
- In this unit, we'll learn about Object-Oriented Programming (OOP) in Python and its fundamental concept with the help of examples.
  - Object Oriented Programming
- Python is a multi-paradigm programming language. It supports different programming approaches. One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).
- An object has two characteristics:
  - Attributes
  - behavior
- Let's take an example:
- A parrot is can be an object, as it has the following properties:
    - name, age, color as attributes
    - singing, dancing as behavior
- The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

Class

- A class is a blueprint for the object.
- We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, a parrot is an object.
- The example for class of parrot can be:

```
class Fruit:

        Pass
```

- Here, we use the class keyword to define an empty class fruit From class, we construct instances. An instance is a specific object created from a particular class.



```python
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

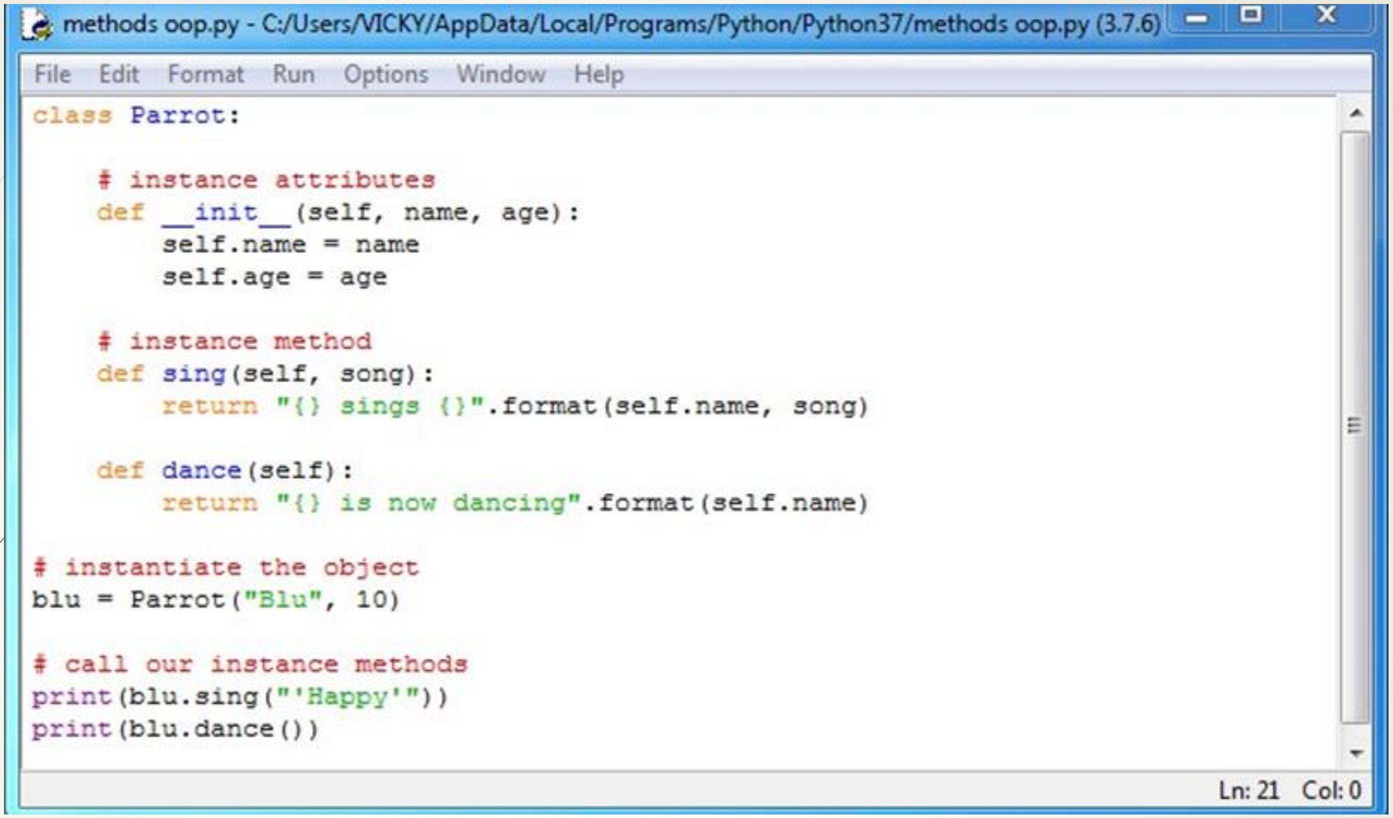
The example for object of parrot class can be:

obj = Parrot()

Here, obj is an object of class Parrot.

Suppose we have details of parrots. Now, we are going to show how to build the class and objects of parrots.

Example 1: Creating Class and Object in Python

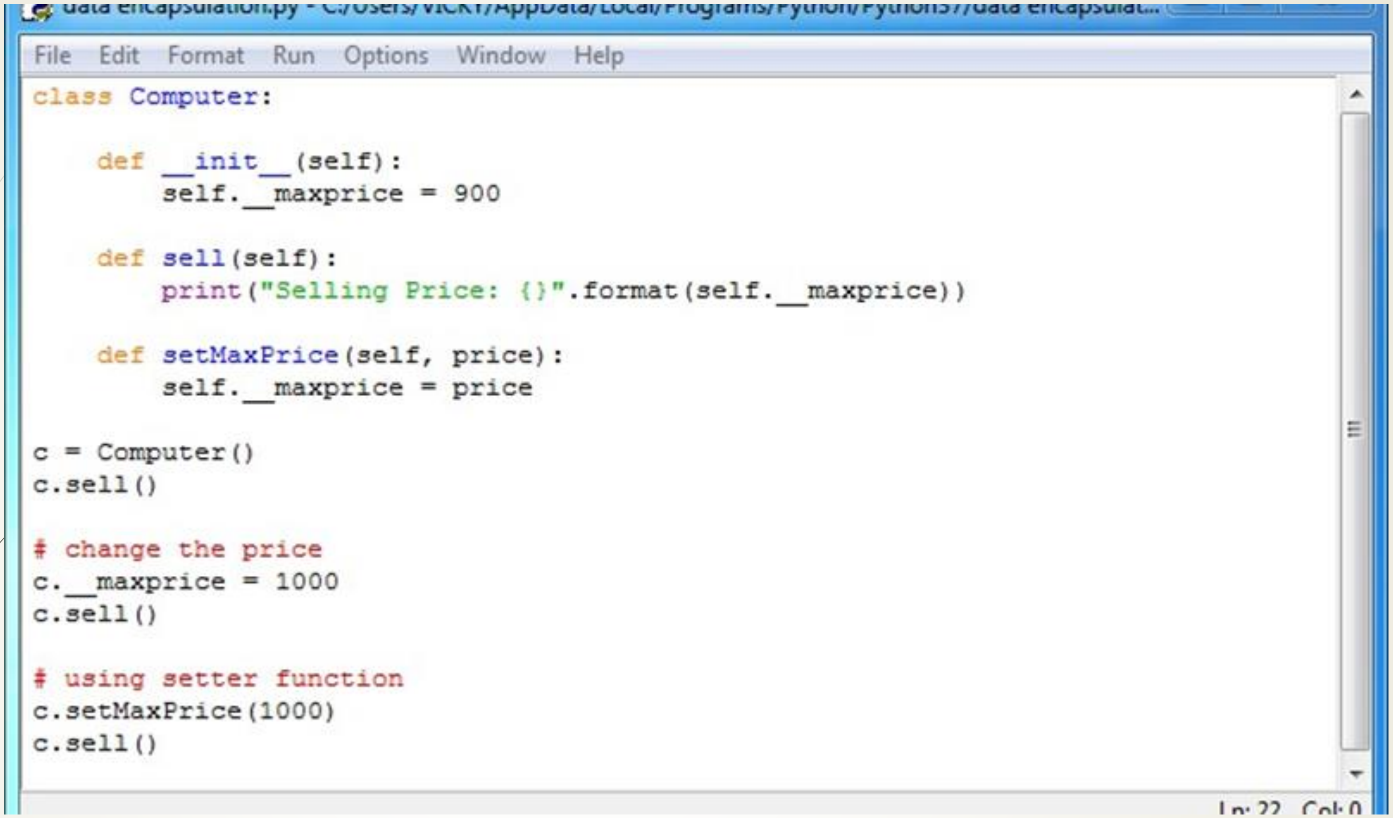

methods oop.py - C:/Users/VICKY/AppData/Local/Programs/Python/Python37/methods oop.py (3.7.6)
File Edit Format Run Options Window Help
Ln: 21 Col: 0

```python
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

In the program, we define two methods i.e sing() and dance(). These are called instance methods because they are called on an instance object i.e blu.

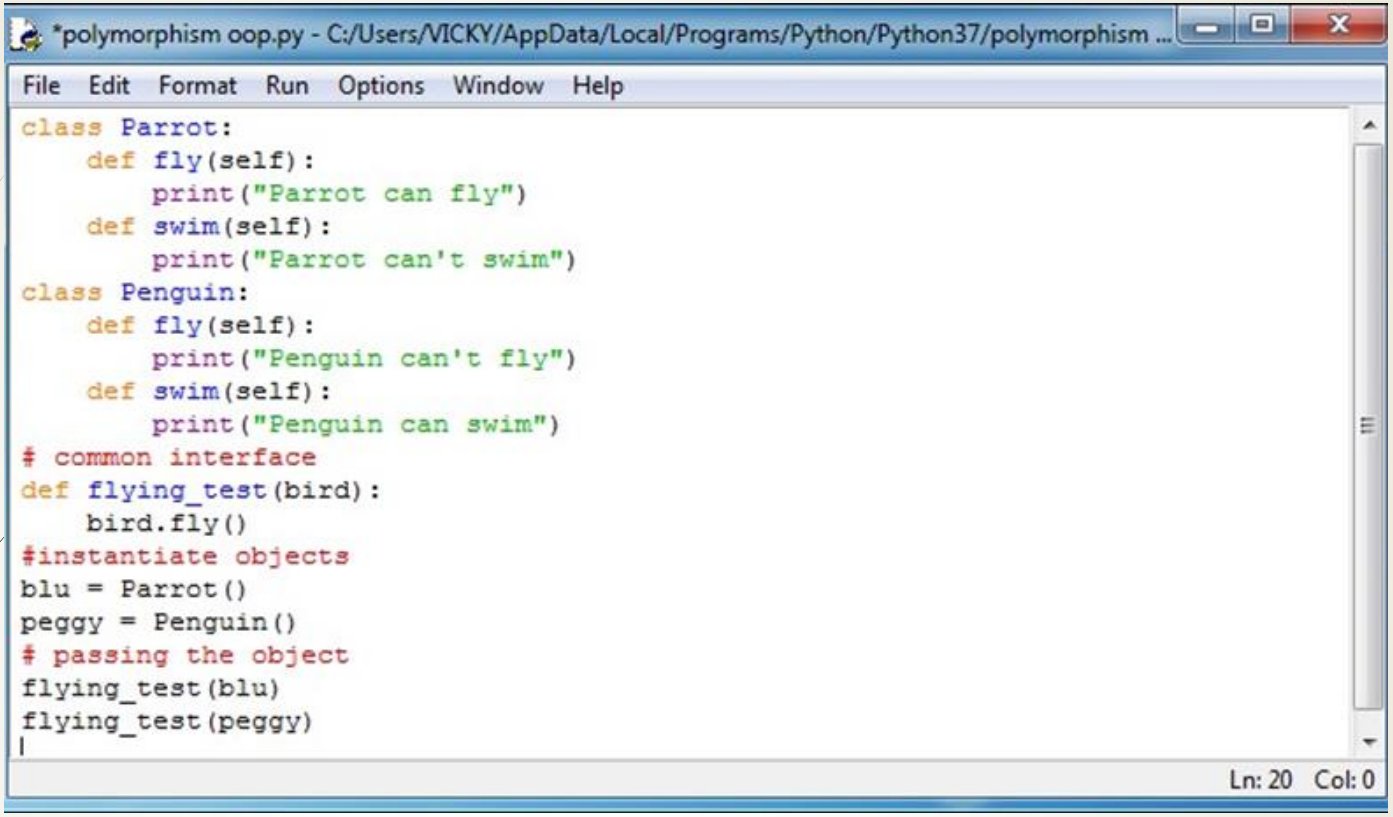

data encapsulation.py - C:/Users/VICKY/AppData/Local/Programs/Python/Python37/data encapsulat...
File Edit Format Run Options Window Help
Ln: 22 Col: 0

```python
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single _ or double .

We used init () method to store the maximum selling price of Computer. We tried to modify the price. However, we can't change it because Python treats the maxprice as private attributes.


*polymorphism oop.py - C:/Users/VICKY/AppData/Local/Programs/Python/Python37/polymorphism ...
File Edit Format Run Options Window Help
Ln: 20 Col: 0

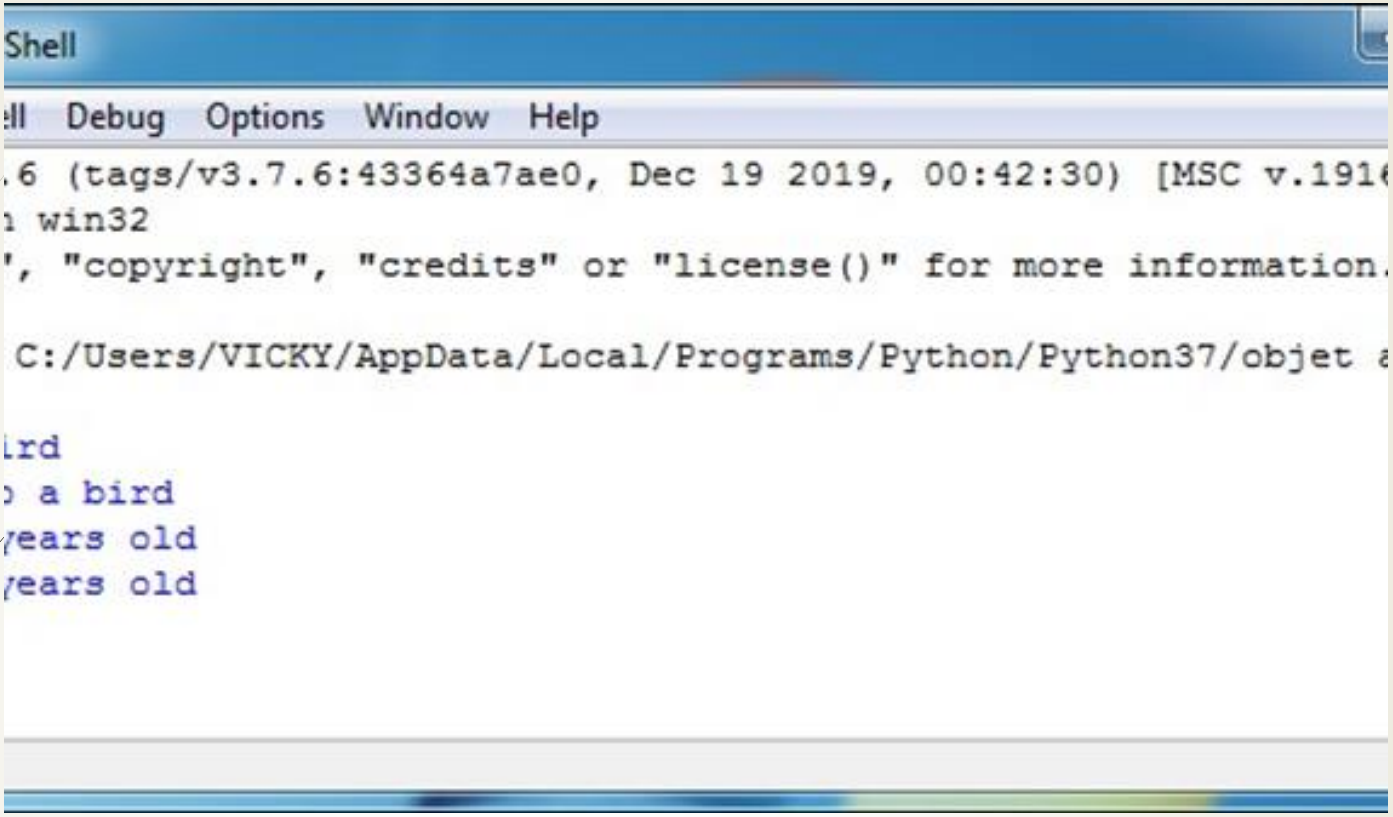
```
class Parrot:
    def fly(self):
        print("Parrot can fly")
    def swim(self):
        print("Parrot can't swim")
class Penguin:
    def fly(self):
        print("Penguin can't fly")
    def swim(self):
        print("Penguin can swim")
# common interface
def flying_test(bird):
    bird.fly()
#instantiate objects
blu = Parrot()
peggy = Penguin()
# passing the object
flying_test(blu)
flying_test(peggy)
```

Polymorphism

Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However we could use the same method to color any shape. This concept is called Polymorphism.

To use polymorphism, we created a common interface i.e flying_test() function that takes any object and calls the object's fly() method. Thus, when we passed the blu and peggy objects in the flying_test() function, it ran effectively.



```
.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.191
n win32
", "copyright", "credits" or "license()" for more information.

 C:/Users/VICKY/AppData/Local/Programs/Python/Python37/objet a

ird
o a bird
years old
years old
```

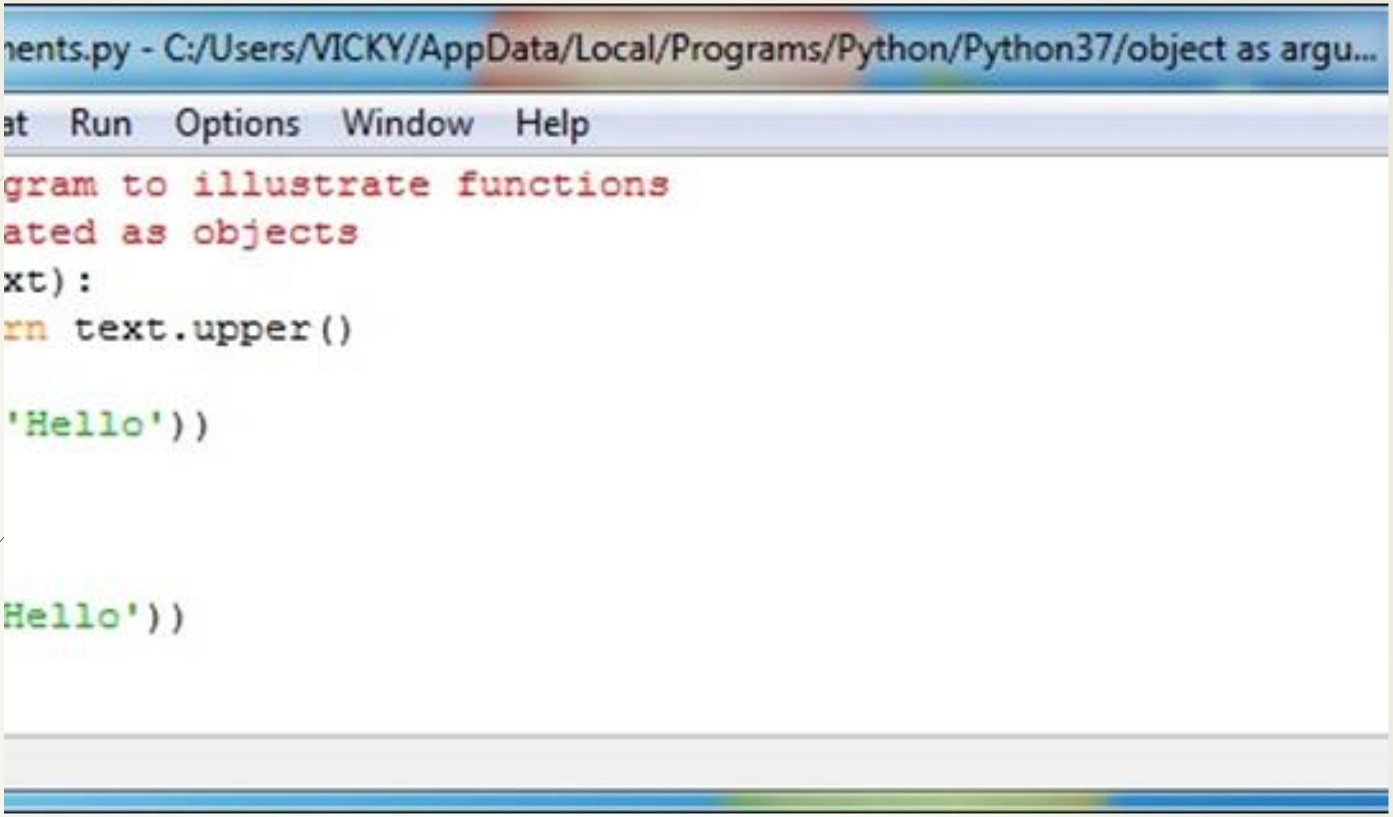
In the above program, we created a class with the name Parrot. Then, we define attributes. The attributes are a characteristic of an object.

These attributes are defined inside the init method of the class. It is the initializer method that is first run as soon as the object is created.

Then, we create instances of the Parrot class. Here, blu and woo are references (value) to our

new objects.

We can access the class attribute using class.species. Class attributes are the same for all instances of a class. Similarly, we access the instance attributes using blu.name and blu.age. However, instance attributes are different for every instance of a class



nents.py - C:/Users/VICKY/AppData/Local/Programs/Python/Python37/object as argu...
at Run Options Window Help

```
gram to illustrate functions
ated as objects
xt):
rn text.upper()

'Hello'))



Hello'))
```

Passing function as an argument in Python

A function can take multiple arguments, these arguments can be objects, variables(of same or differen data type)

and functions. Python functions are first class objects.

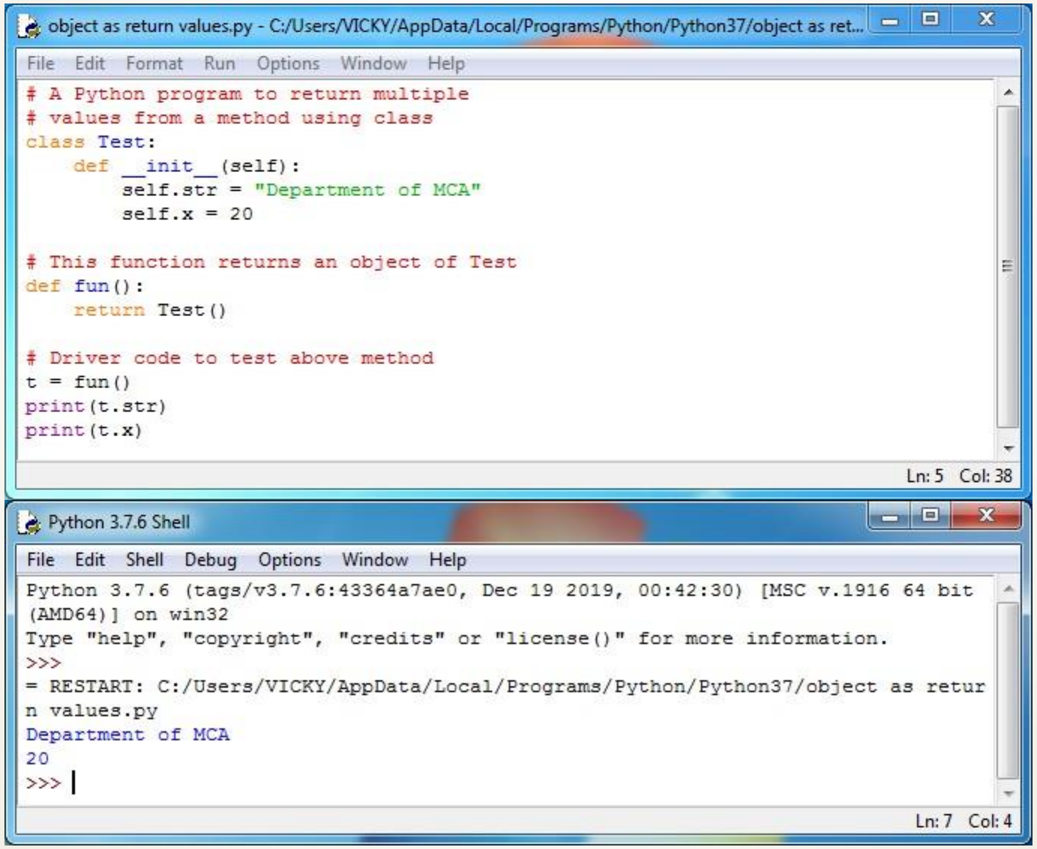
In the example below, a function is assigned to a variable.

This assignment doesn't call the function. It takes the function object referenced by shout and creates second

name pointing to it, yell.

Example2: object as arguments

Returning Multiple Values in Python



object as return values.py - C:/Users/VICKY/AppData/Local/Programs/Python/Python37/object as ret...
File Edit Format Run Options Window Help
Ln: 5 Col: 38
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Ln: 7 Col: 4

```python
# A Python program to return multiple
# values from a method using class
class Test:
    def __init__(self):
        self.str = "Department of MCA"
        self.x = 20

# This function returns an object of Test
def fun():
    return Test()

# Driver code to test above method
t = fun()
print(t.str)
print(t.x)
```

```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/VICKY/AppData/Local/Programs/Python/Python37/object as retur
n values.py
Department of MCA
20
>>> 
```

Build-in Class Attributes

- Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −
  - dict − Dictionary containing the class's namespace.
  - doc − Class documentation string or none, if undefined.
  - name −. This gives us the class name
  - module − Class nameModule name in which the class is defined. This attribute is " main " in interactive mode.
  - bases − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

The ___doc___class attribute

- Program:

```
# class
class Awesome:
'Government Arts College,coimbatore.'

def   init     (self):
print("Hello from  init     method.")

# class built-in attribute  print(Awesome.         doc   )
```

The above code will give us the following output.

```
Government Arts College,coimbatore.
```

The name   class attribute

- In the following example we are printing the name of the class.

- # class
- class Example:
- 'This is a sample class called Awesome.'
- def__init__(self):
  - print("Hello from__init__method.")
- # class built-in attribute print(Example.__name__)
- **Output:**
- Example

The module   class attribute

- In the following example we are printing the module of the class.

```
# class

class Example:  def         init    (self):

print("Hello from      init    method.")


# class built-in attribute  print(Example. module     )


Output:


      main
```

Python Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.
  - **Python Inheritance Syntax**

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

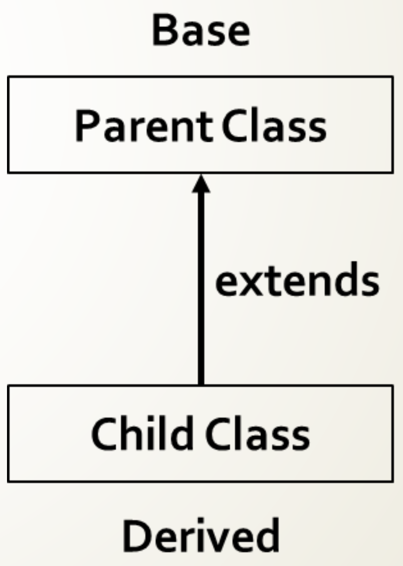
Types Of Inheritance

Types of inheritance depends upon the number of child and parent classes involved.

- There are five types of inheritance in python
  - Single inheritance
  - Multiple inheritance
  - Multilevel inheritance
  - Hierarchical inheritance
  - Hybrid inheritance

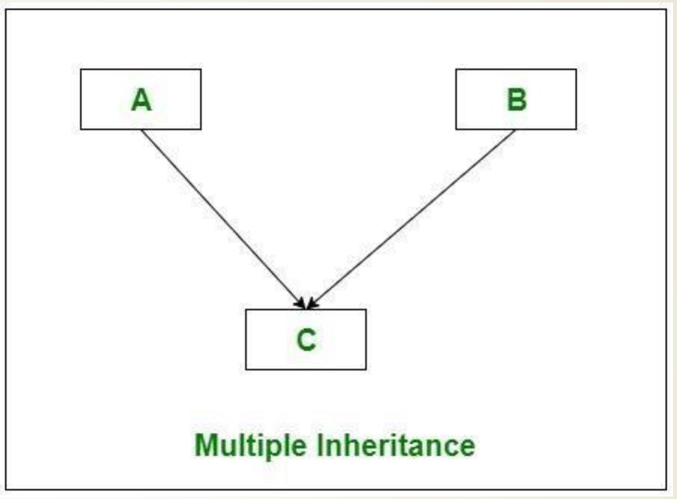
Single inheritance:

One base class and one derived class calls single inheritance.



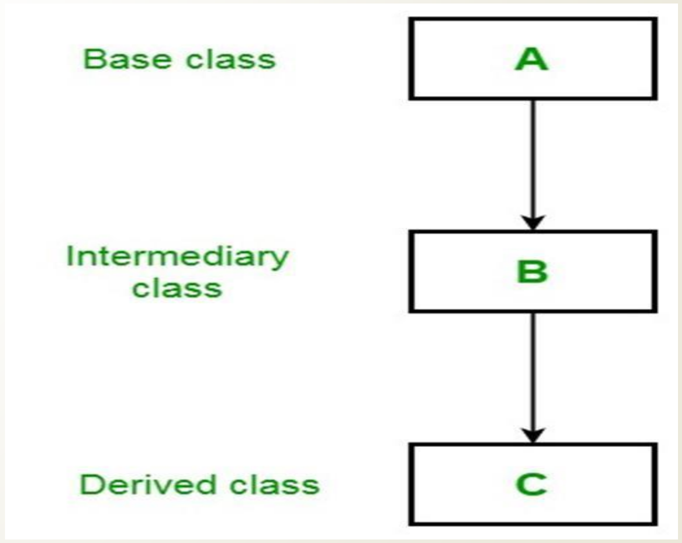
2.Multiple inheritance:

- ❖ One derived class and two or more base classes



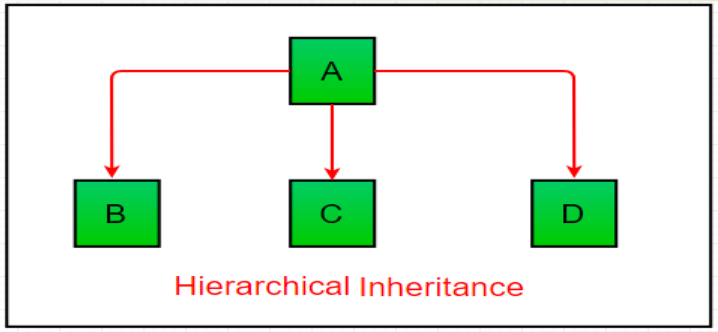
3.Multilevel inheritance:

- ❖ One base class(A),one derived class(B) which in turn serves as a base class for one or more derived(C)class.



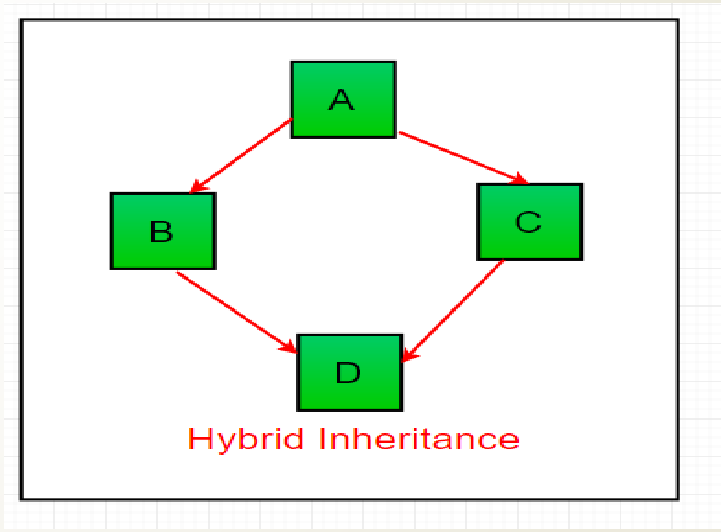
4.Hierarchical inheritance:

❖ One base class and one or more derived classes.



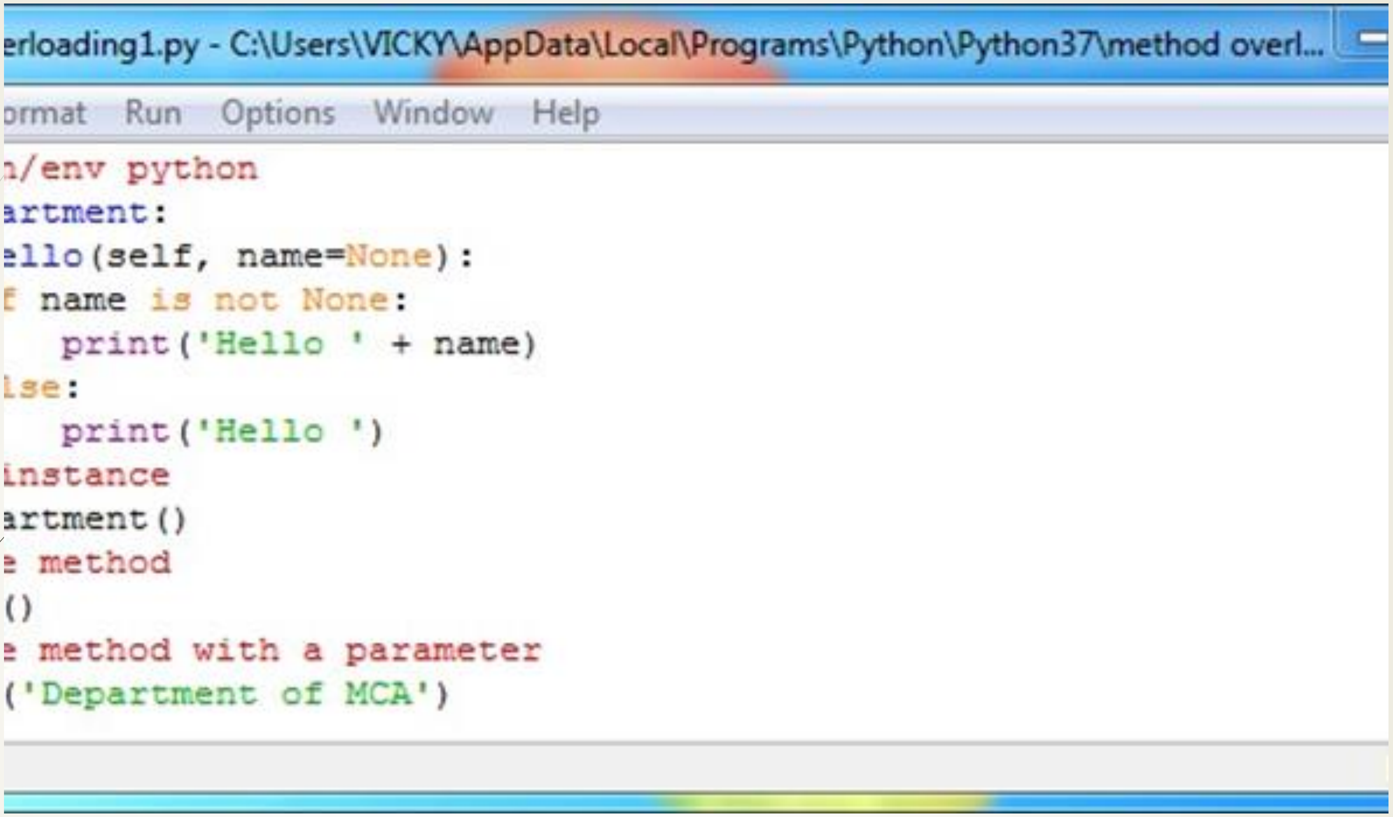
5.Hybrid inheritance:

❖ Combination of two or more inheritance.



Example : PYTHON INHERITANCE

Output:


erloading1.py - C:\Users\VICKY\AppData\Local\Programs\Python\Python37\method overl...
ormat Run Options Window Help

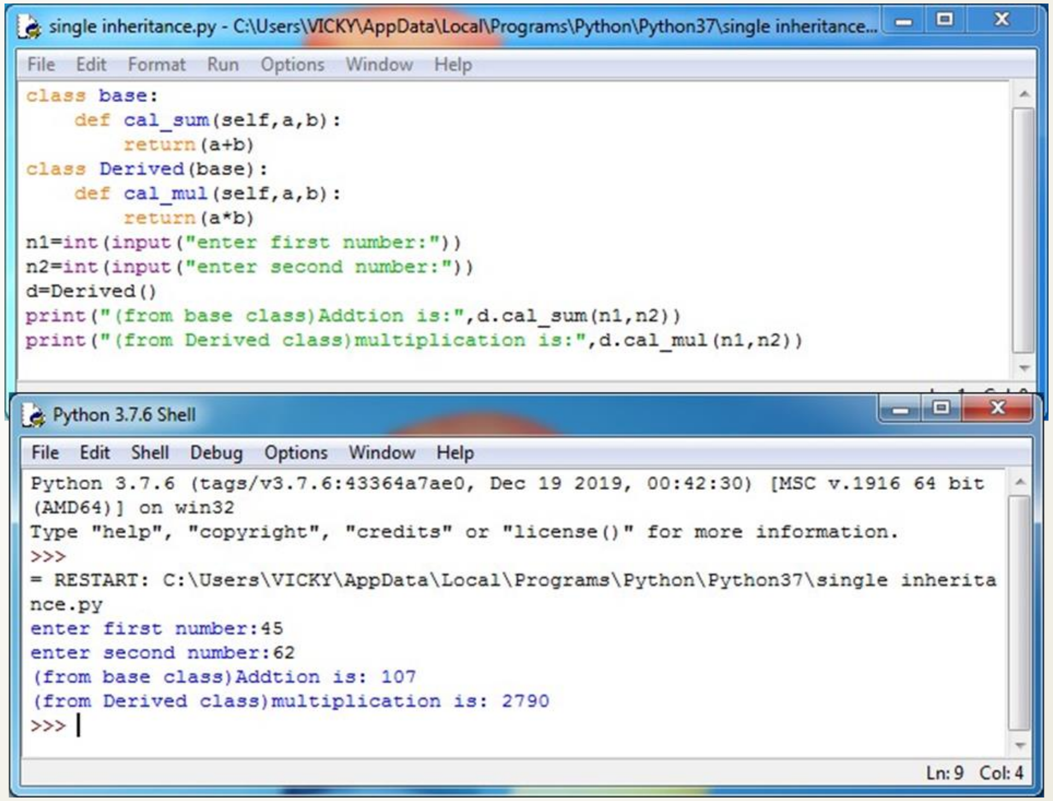
```
n/env python
artment:
ello(self, name=None):
f name is not None:
   print('Hello ' + name)
lse:
   print('Hello ')
instance
artment()
e method
()
e method with a parameter
('Department of MCA')
```

METHOD OVERLOADING

- Method Overloading is the class having methods that are the same name with different arguments.
- Arguments different will be based on a number of arguments and types of arguments.
- It is used in a single class. It is also used to write the code clarity as well as reduce complexity

EXAMPLES

EXAMPLE1: SINGLE INHERITANCE

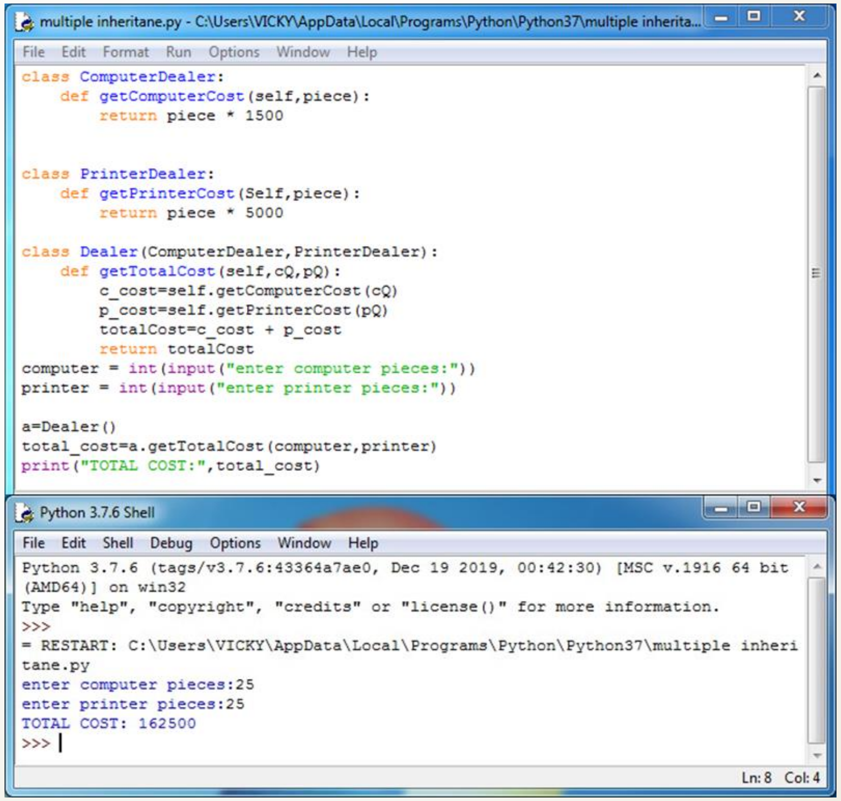

single inheritance.py - C:\Users\VICKY\AppData\Local\Programs\Python\Python37\single inheritance...
File Edit Format Run Options Window Help

```python
class base:
    def cal_sum(self,a,b):
        return(a+b)
class Derived(base):
    def cal_mul(self,a,b):
        return(a*b)
n1=int(input("enter first number:"))
n2=int(input("enter second number:"))
d=Derived()
print("(from base class)Addtion is:",d.cal_sum(n1,n2))
print("(from Derived class)multiplication is:",d.cal_mul(n1,n2))
```

Python 3.7.6 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\VICKY\AppData\Local\Programs\Python\Python37\single inherita
nce.py
enter first number:45
enter second number:62
(from base class)Addtion is: 107
(from Derived class)multiplication is: 2790
>>> |
```

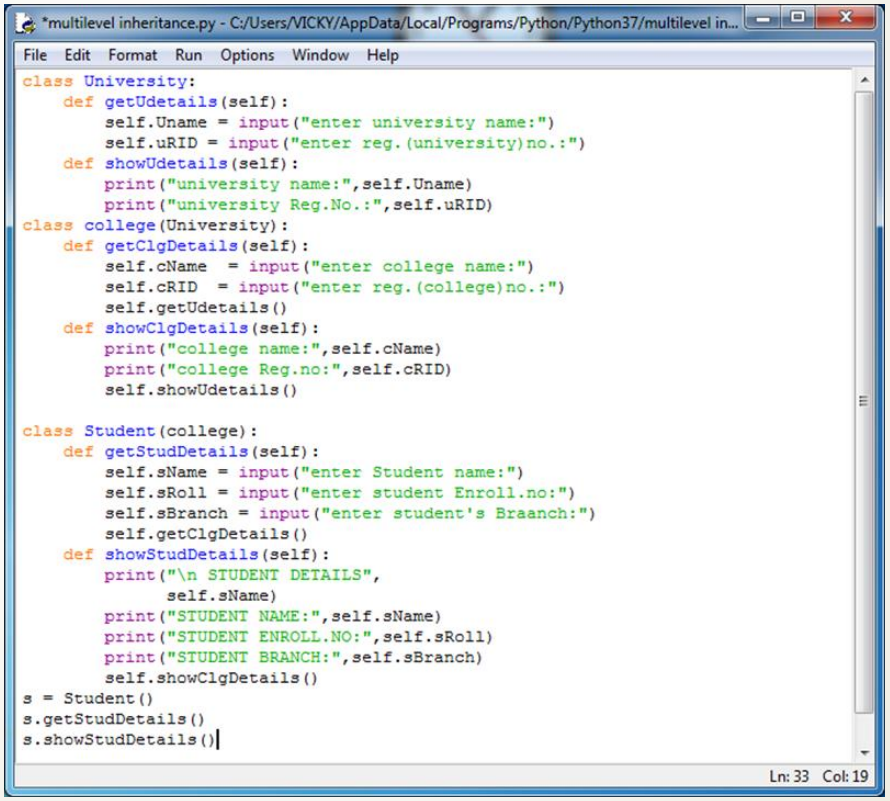
EXAMPLE2: MULTIPLE INHERITANCE


multiple inheritane.py - C:\Users\VICKY\AppData\Local\Programs\Python\Python37\multiple inherita...
File Edit Format Run Options Window Help

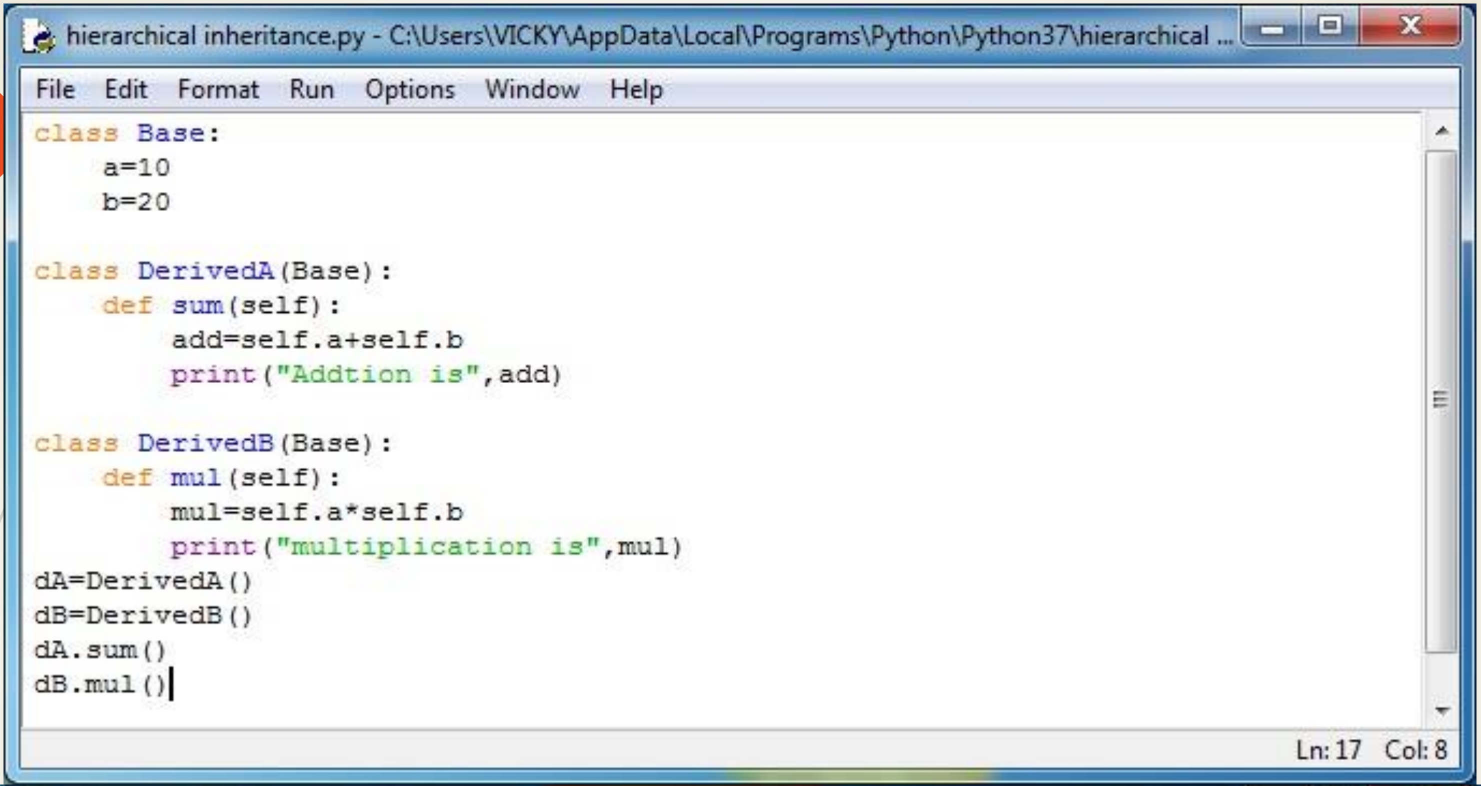
```python
class ComputerDealer:
    def getComputerCost(self,piece):
        return piece * 1500


class PrinterDealer:
    def getPrinterCost(Self,piece):
        return piece * 5000

class Dealer(ComputerDealer,PrinterDealer):
    def getTotalCost(self,cQ,pQ):
        c_cost=self.getComputerCost(cQ)
        p_cost=self.getPrinterCost(pQ)
        totalCost=c_cost + p_cost
        return totalCost
computer = int(input("enter computer pieces:"))
printer = int(input("enter printer pieces:"))

a=Dealer()
total_cost=a.getTotalCost(computer,printer)
print("TOTAL COST:",total_cost)
```

```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\VICKY\AppData\Local\Programs\Python\Python37\multiple inheri
tane.py
enter computer pieces:25
enter printer pieces:25
TOTAL COST: 162500
>>> 
```

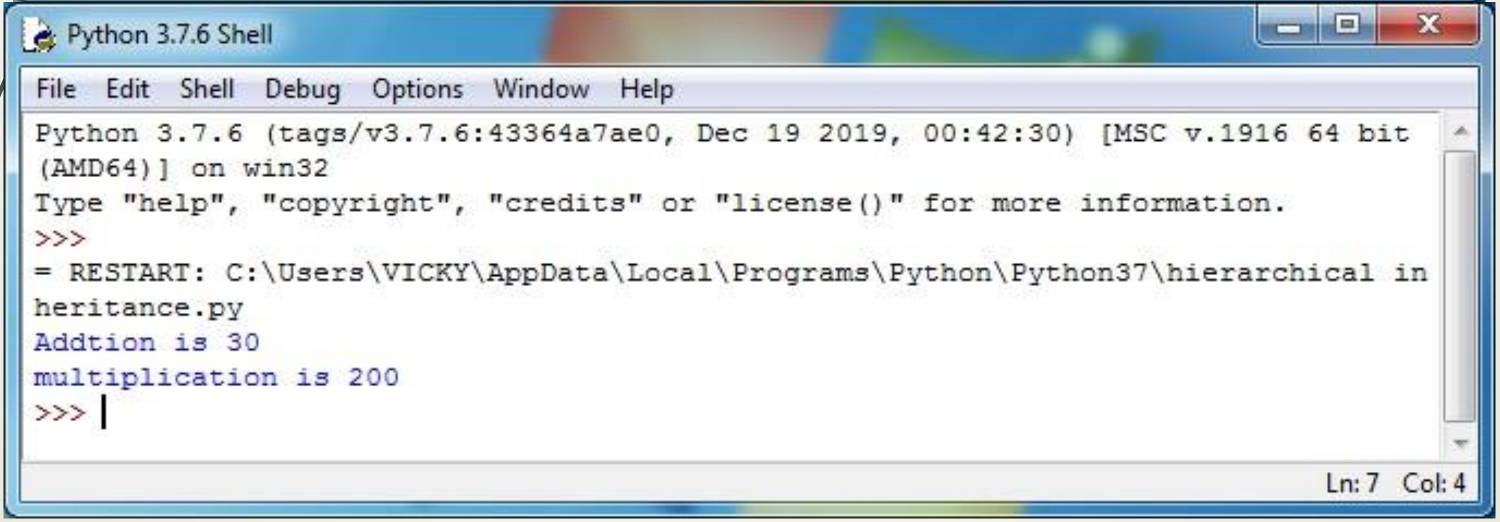
EXAMPLE2: MULTILEVEL INHERITANCE


*multilevel inheritance.py - C:/Users/VICKY/AppData/Local/Programs/Python/Python37/multilevel in...
File Edit Format Run Options Window Help
Ln: 33 Col: 19

```python
class University:
    def getUdetails(self):
        self.Uname = input("enter university name:")
        self.uRID = input("enter reg.(university)no.:")
    def showUdetails(self):
        print("university name:",self.Uname)
        print("university Reg.No.:",self.uRID)
class college(University):
    def getClgDetails(self):
        self.cName  = input("enter college name:")
        self.cRID  = input("enter reg.(college)no.:")
        self.getUdetails()
    def showClgDetails(self):
        print("college name:",self.cName)
        print("college Reg.no:",self.cRID)
        self.showUdetails()

class Student(college):
    def getStudDetails(self):
        self.sName = input("enter Student name:")
        self.sRoll = input("enter student Enroll.no:")
        self.sBranch = input("enter student's Braanch:")
        self.getClgDetails()
    def showStudDetails(self):
        print("\n STUDENT DETAILS",
              self.sName)
        print("STUDENT NAME:",self.sName)
        print("STUDENT ENROLL.NO:",self.sRoll)
        print("STUDENT BRANCH:",self.sBranch)
        self.showClgDetails()
s = Student()
s.getStudDetails()
s.showStudDetails()
```


hierarchical inheritance.py - C:\Users\VICKY\AppData\Local\Programs\Python\Python37\hierarchical ...
File Edit Format Run Options Window Help
Ln: 17 Col: 8

```
class Base:
    a=10
    b=20

class DerivedA(Base):
    def sum(self):
        add=self.a+self.b
        print("Addtion is",add)

class DerivedB(Base):
    def mul(self):
        mul=self.a*self.b
        print("multiplication is",mul)
dA=DerivedA()
dB=DerivedB()
dA.sum()
dB.mul()
```



```
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\VICKY\AppData\Local\Programs\Python\Python37\hierarchical in
heritance.py
Addtion is 30
multiplication is 200
>>> 
```

Thank you

The Content in this Material are from the Textbooks and Reference books given in the Syllabus