………………………………………………………………………………………………

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter-17\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*METHODS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………………………

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Types of Methods:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

…………………………………………………………………………………………….

Inside Python class 3 types of methods are allowed

1. Instance Methods

2. Class Methods

3. Static Methods

……………………………………………………………………………………………..\*\*\*\*\*\*1. Instance Methods: \*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………………………. Inside method implementation if we are using instance variables then such type of methods are called instance methods.

Inside instance method declaration,we have to pass self variable.

……………………………………………………….

def m1(self):

By using self variable inside method we can able to access instance variables.

Within the class we can call instance method by using self variable and from outside of the class we can call by using object reference.

……………………………………………………………….

1) class Student:

2) def \_\_init\_\_(self,name,marks):

3) self.name=name

4) self.marks=marks

5) def display(self):

6) print('Hi',self.name)

7) print('Your Marks are:',self.marks)

8) def grade(self):

9) if self.marks>=60:

10) print('You got First Grade')

11) elif self.marks>=50:

12) print('Yout got Second Grade')

13) elif self.marks>=35:

14) print('You got Third Grade')

15) else:

16) print('You are Failed')

17) n=int(input('Enter number of students:')) 1

8) for i in range(n):

19) name=input('Enter Name:')

20) marks=int(input('Enter Marks:'))

21) s= Student(name,marks)

22) s.display()

23) s.grade()

24) print()

………………………………………………………………………………….

ouput:

D:\ prasanna \_classes>py test.py

Enter number of students:2

Enter Name:Prasanna

Enter Marks:90

Hi Prasanna

Your Marks are: 90

You got First Grade

…………………………………………………………………….

Enter Name:Ravi

Enter Marks:12

Hi Ravi

Your Marks are: 12

You are Failed

………………………………………………………………………………

\*\*\*\*\*\*\*\*\*\*\*\*Setter and Getter Methods:\*\*\*\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………………… We can set and get the values of instance variables by using getter and setter methods.

………………………………………………………..

\*\*\*\*\*\*\*\*\*\*Setter Method: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

……………………………………………………………………………….

setter methods can be used to set values to the instance variables. setter methods also known as mutator methods.

…………………………………………………………….

syntax:

def setVariable(self,variable):

self.variable=variable

…………………………………………………………………………

Example:

def setName(self,name):

self.name=name

……………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Getter Method: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

…………………………………………………………………………………………….

Getter methods can be used to get values of the instance variables. Getter methods also known as accessor methods.

………………………………………………..

syntax:

def getVariable(self):

return self.variable

………………………………………………

Example:

def getName(self):

return self.name

……………………………………………………………………………………….

Demo Program:

1) class Student:

2) def setName(self,name):

3) self.name=name

4)

5) def getName(self):

6) return self.name

7)

8) def setMarks(self,marks):

9) self.marks=marks

10)

11) def getMarks(self):

12) return self.marks

13)

14) n=int(input('Enter number of students:'))

15) for i in range(n):

16) s=Student()

17) name=input('Enter Name:')

18) s.setName(name)

19) marks=int(input('Enter Marks:'))

20) s.setMarks(marks)

21)

22) print('Hi',s.getName())

23) print('Your Marks are:',s.getMarks())

24) print()

………………………………………………………….

output:

D:\python\_classes>py test.py

Enter number of students:2

Enter Name:Prasanna

Enter Marks:100

Hi Prasanna

Your Marks are: 100

…………………………………………………………………..

Enter Name:Ravi

Enter Marks:80

Hi Ravi

Your Marks are: 80

………………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*2. Class Methods:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

…………………………………………………………………………………………..

Inside method implementation if we are using only class variables (static variables), then such type of methods we should declare as class method.

We can declare class method explicitly by using @classmethod decorator.

For class method we should provide cls variable at the time of declaration

We can call classmethod by using classname or object reference variable.

……………………………………………………………………………..

Demo Program:

1) class Animal:

2) legs=4

3) @classmethod

4) def walk(cls,name):

5) print('{} walks with {} legs...'.format(name,cls.legs))

6) Animal.walk('Dog')

7) Animal.walk('Cat')

……………………………………………..

Output

D:\python\_classes>py test.py

Dog walks with 4 legs...

Cat walks with 4 legs...

………………………………………………………………………………

Program to track the number of objects created for a class:

1) class Test:

2) count=0

3) def \_\_init\_\_(self):

4) Test.count =Test.count+1

5) @classmethod

6) def noOfObjects(cls):

7) print('The number of objects created for test class:',cls.count)

8)

9) t1=Test()

10) t2=Test()

11) Test.noOfObjects()

12) t3=Test()

13) t4=Test()

14) t5=Test()

15) Test.noOfObjects()

………………………………………………………………......................

1. Static Methods: -

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In general these methods are general utility methods.

Inside these methods we won't use any instance or class variables.

Here we won't provide self or cls arguments at the time of declaration.

We can declare static method explicitly by using @staticmethod decorator

We can access static methods by using classname or object reference

……………………………………………………

1) class PrasannaMath:

2)

3) @staticmethod

4) def add(x,y):

5) print('The Sum:',x+y)

6)

7) @staticmethod

8) def product(x,y):

9) print('The Product:',x\*y)

10)

11) @staticmethod

12) def average(x,y):

13) print('The average:',(x+y)/2)

14)

15) PrasannaMath.add(10,20)

16) PrasannaMath.product(10,20)

17) PrasannaMath.average(10,20)

………………………………………………………………………………

Output

The Sum: 30

The Product: 200

The average: 15.0

……………………………………………………………………..

Note: In general we can use only instance and static methods.Inside static method we can access class level variables by using class name.

……………………………………………………………………………..

class methods are most rarely used methods in python.

……………………………………………………………………………………..

\*\*\*\*\*Passing members of one class to another class:\*\*\*

………………………………………………………………………………………..

We can access members of one class inside another class.

1) class Employee:

2) def \_\_init\_\_(self,eno,ename,esal):

3) self.eno=eno

4) self.ename=ename

5) self.esal=esal

6) def display(self):

7) print('Employee Number:',self.eno)

8) print('Employee Name:',self.ename)

9) print('Employee Salary:',self.esal)

10) class Test:

11) def modify(emp):

12) emp.esal=emp.esal+10000

13) emp.display()

14) e=Employee(100,'Prasanna',10000)

15) Test.modify(e)

………………………………………………………………..

Output

D:\python\_classes>py test.py

Employee Number: 100

Employee Name: Prasanna

Employee Salary: 20000

…………………………………………………………………………………………….

In the above application, Employee class members are available to Test class.

………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Inner classes: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

………………………………………………………………………………………….

Sometimes we can declare a class inside another class,such type of classes are called inner classes.

Without existing one type of object if there is no chance of existing another type of object,then we should go for inner classes.

…………………………………………………………….

Example: Without existing Car object there is no chance of existing Engine object. Hence Engine class should be part of Car class.

class Car:

.....

class Engine:

......

……………………………………………………………………………………..

Example: Without existing university object there is no chance of existing Department object

class University:

.....

class Department:

......

…………………………………………………………………………………………

eg3: Without existing Human there is no chance of existin Head. Hence Head should be part of Human.

class Human:

class Head:

……………………………………………………………………………

Note: Without existing outer class object there is no chance of existing inner class object. Hence inner class object is always associated with outer class object.

……………………………………………………………………………….

Demo Program-1:

1) class Outer:

2) def \_\_init\_\_(self):

3) print("outer class object creation")

4) class Inner:

5) def \_\_init\_\_(self):

6) print("inner class object creation")

7) def m1(self):

8) print("inner class method")

9) o=Outer()

10) i=o.Inner()

11) i.m1()

……………………………………………………………………………..

Output

outer class object creation

inner class object creation

inner class method

/……………………………………………………………….

Note: The following are various possible syntaxes for calling inner class method

……………………………………………………..

1.

o=Outer()

i=o.Inner()

i.m1()

……………………………………………………

2.

i=Outer().Inner()

i.m1()

……………………………………………………………………….

3. Outer().Inner().m1()

…………………………………………………………………………….

Demo Program-2:

1) class Person:

2) def \_\_init\_\_(self):

3) self.name='prasanna'

4) self.db=self.Dob()

5) def display(self):

6) print('Name:',self.name)

7) class Dob:

8) def \_\_init\_\_(self):

9) self.dd=10

10) self.mm=5

11) self.yy=1947

12) def display(self):

13) print('Dob={}/{}/{}'.format(self.dd,self.mm,self.yy)) 14) p=Person()

15) p.display()

16) x=p.db

17) x.display()

…………………………………………………………………………………

Output

Name: prasanna

Dob=10/5/1947

………………………………………………………………………

Demo Program-3:

Inside a class we can declare any number of inner classes.

1) class Human:

2)

3) def \_\_init\_\_(self):

4) self.name = 'Sunny'

5) self.head = self.Head()

6) self.brain = self.Brain()

7) def display(self):

8) print("Hello..",self.name)

9)

10) class Head:

11) def talk(self):

12) print('Talking...')

13)

14) class Brain:

15) def think(self):

16) print('Thinking...')

17)

18) h=Human()

19) h.display()

20) h.head.talk()

21) h.brain.think()

………………………………………………………………………………

Output

Hello.. Sunny

Talking...

Thinking...

………………………………………………………………………………………

\*\*\*\*\*\*\*\*\*\*Garbage Collection:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………………..

In old languages like C++, programmer is responsible for both creation and destruction of objects.Usually programmer taking very much care while creating object, but neglecting destruction of useless objects. Because of his neglectance, total memory can be filled with useless objects which creates memory problems and total application will be down with Out of memory error.

But in Python, We have some assistant which is always running in the background to destroy useless objects.Because this assistant the chance of failing Python program with memory problems is very less. This assistant is nothing but Garbage Collector.

Hence the main objective of Garbage Collector is to destroy useless objects.

If an object does not have any reference variable then that object eligible for Garbage Collection.

……………………………………………………………………………………………

How to enable and disable Garbage Collector in our program:

…………………………………………………………………………………………..

By default Gargbage collector is enabled, but we can disable based on our requirement. In this context we can use the following functions of gc module.

1. gc.isenabled():-

……………………………………

Returns True if GC enabled

…………………………………………………………………………………………..

2. gc.disable() To disable GC explicitly

………………………………………………………………………………………….

3. gc.enable() To enable GC explicitly

………………………………………………………………………………..

Example:

1) import gc

2) print(gc.isenabled())

3) gc.disable()

4) print(gc.isenabled())

5) gc.enable()

6) print(gc.isenabled())

………………………………………………………………………….

Output

True

False

True

………………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Destructors:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………………….

Destructor is a special method and the name should be \_\_del\_\_

Just before destroying an object Garbage Collector always calls destructor to perform clean up activities (Resource deallocation activities like close database connection etc).

Once destructor execution completed then Garbage Collector automatically destroys that object.

……………………………………………………………………………………………..

Note: The job of destructor is not to destroy object and it is just to perform clean up activities.

……………………………………………………………………………..

Example:

1) import time

2) class Test:

3) def \_\_init\_\_(self):

4) print("Object Initialization...")

5) def \_\_del\_\_(self):

6) print("Fulfilling Last Wish and performing clean up activities...")

7)

8) t1=Test()

9) t1=None

10) time.sleep(5)

11) print("End of application")

…………………………………………………………………………………..

Output

Object Initialization...

Fulfilling Last Wish and performing clean up activities...

End of application

……………………………………………………………………………………………….Note: If the object does not contain any reference variable then only it is eligible fo GC. ie if the reference count is zero then only object eligible for GC

……………………………………………………………………………………………..

Example:

1) import time

2) class Test:

3) def \_\_init\_\_(self):

4) print("Constructor Execution...")

5) def \_\_del\_\_(self):

6) print("Destructor Execution...")

7)

8) t1=Test()

9) t2=t1

10) t3=t2

11) del t1

12) time.sleep(5)

13) print("object not yet destroyed after deleting t1")

14) del t2

15) time.sleep(5)

16) print("object not yet destroyed even after deleting t2") 17) print("I am trying to delete last reference variable...") 18) del t3

……………………………………………………………………………….

Example:

1) import time

2) class Test:

3) def \_\_init\_\_(self):

4) print("Constructor Execution...")

5) def \_\_del\_\_(self):

6) print("Destructor Execution...")

7)

8) list=[Test(),Test(),Test()]

9) del list

10) time.sleep(5)

11) print("End of application")

………………………………………………………..

Output

Constructor Execution...

Constructor Execution...

Constructor Execution...

Destructor Execution...

Destructor Execution...

Destructor Execution...

End of application

……………………………………………………………………………………………..\*\*\*\*\*\*How to find the number of references of an object:\*\*

………………………………………………………………………………………….. sys module contains getrefcount() function for this purpose.

sys.getrefcount(objectreference)

……………………………………………………………

Example:

1) import sys

2) class Test:

3) pass

4) t1=Test()

5) t2=t1

6) t3=t1

7) t4=t1

8) print(sys.getrefcount(t1))

………………………………….

Output 5

Note: For every object, Python internally maintains one default reference variable self.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

………………………………………………………………………………………………