Dharmsinh Desai University, Nadiad
Faculty of technology,
Department of Computer Engineering
Subject : Software Project
Lab Manual

Lab :4 OOP Packages

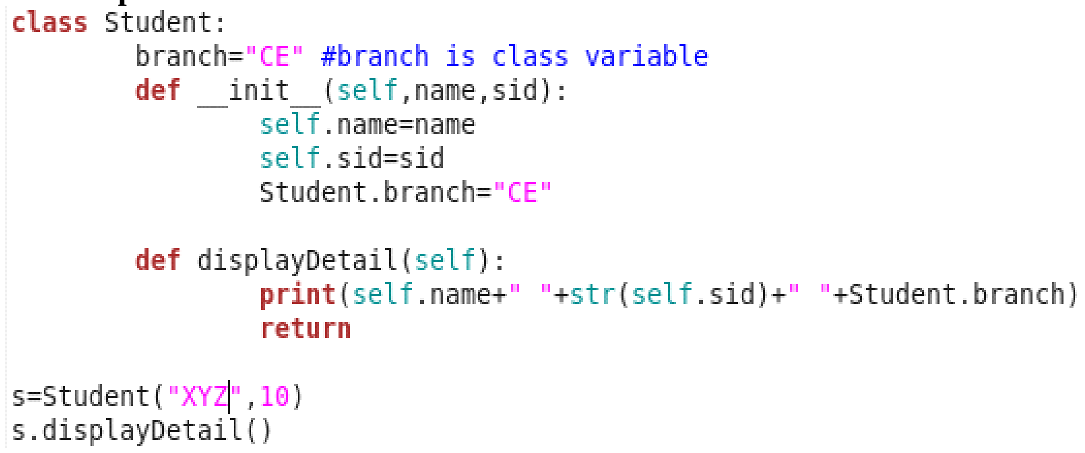
OOP Concepts:

❖ **Class:** To create a class in python following syntax is followed

Syntax:

class ClassName:
'class documentation string which is optional'
class body

Example:



```
class Student:
        branch="CE" #branch is class variable
        def __init__(self,name,sid):
                self.name=name
                self.sid=sid
                Student.branch="CE"

        def displayDetail(self):
                print(self.name+" "+str(self.sid)+" "+Student.branch)
                return

s=Student("XYZ",10)
s.displayDetail()
```

❖ The first method *__init__()* is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.

➢ You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the *self* argument to the list for you; you do not need to include it when you call the methods.

➢ You access the object's attributes using the dot operator with object. Class variable would be accessed using class name.

Instead of using the normal statements to access attributes, you can use the following functions

- The **getattr(obj, name[, default])** − to access the attribute of object.
- The **hasattr(obj,name)** − to check if an attribute exists or not.

- The **setattr(obj,name,value)** − to set an attribute. If attribute does not exist, then it would be created.
- The **delattr(obj, name)** − to delete an attribute.

➢ **Garbage Collection:** Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes.

An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with *del*, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

```
a = 40      # Create object <40>
b = a       # Increase ref. count  of <40>
c = [b]     # Increase ref. count  of <40>

del a       # Decrease ref. count  of <40>
b = 100     # Decrease ref. count  of <40>
c[0] = -1   # Decrease ref. count  of <40>
```

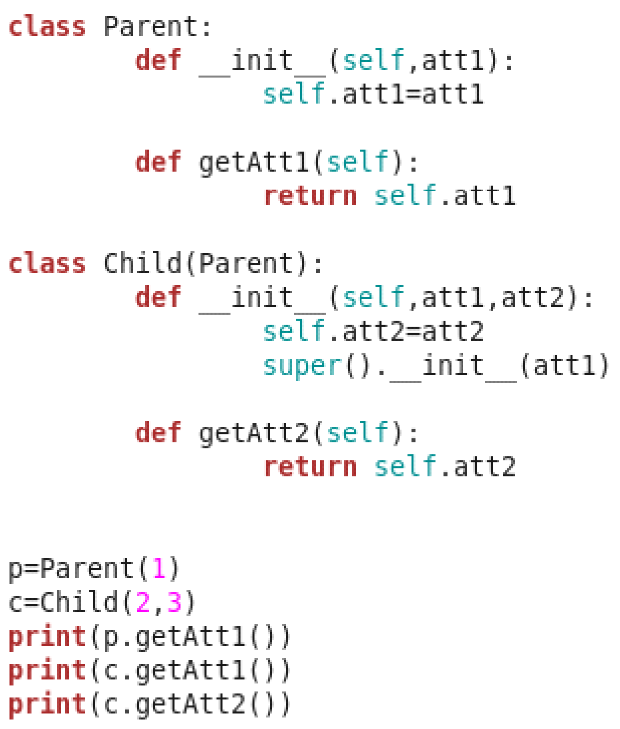
You normally will not notice when the garbage collector destroys an orphaned instance and reclaims its space. But a class can implement the special method *__del__()*, called a destructor, that is invoked when the instance is about to be destroyed. This method might be used to clean up any non memory resources used by an instance.

❖ Inheritance

Syntax:

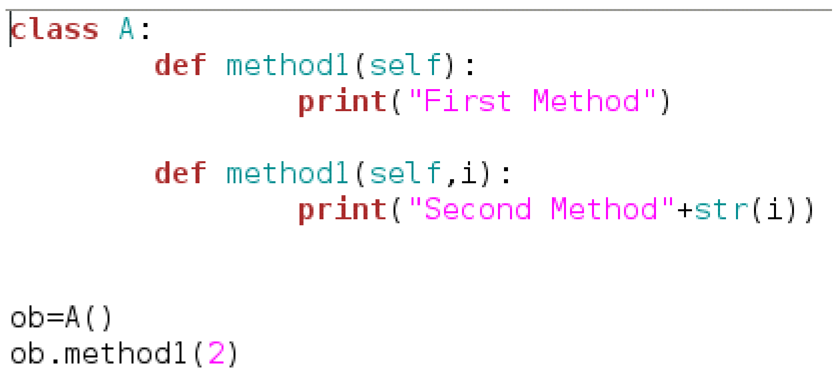
```
class SubClassName (ParentClass1[, ParentClass2, ...]):
        'Optional class documentation string'
        class_body
```

Example:



```
class Parent:
	def __init__(self,att1):
		self.att1=att1

	def getAtt1(self):
		return self.att1

class Child(Parent):
	def __init__(self,att1,att2):
		self.att2=att2
		super().__init__(att1)

	def getAtt2(self):
		return self.att2


p=Parent(1)
c=Child(2,3)
print(p.getAtt1())
print(c.getAtt1())
print(c.getAtt2())
```

In above example, super().__init__() method is used to call __init__() method of parent class. **Python does not support method overloading.** Check the output of following code.



```
class A:
	def method1(self):
		print("First Method")

	def method1(self,i):
		print("Second Method"+str(i))


ob=A()
ob.method1(2)
```

But python supports method overriding as shown in below example.

```
class Parent:
        def __init__(self,att1):
                self.att1=att1

        def getAtt1(self):
                return self.att1

class Child(Parent):
        def __init__(self,att1,att2):
                self.att2=att2
                super().__init__(att1)

        def getAtt2(self):
                return self.att2

        def getAtt1(self):
                return self.att1*2


p=Parent(1)
c=Child(2,3)
print(p.getAtt1())
print(c.getAtt1())
print(c.getAtt2())
```

❖ **Operator Overloading:** In python, we can overload an operator to work as per our definition on objects of specified class .

Example:

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

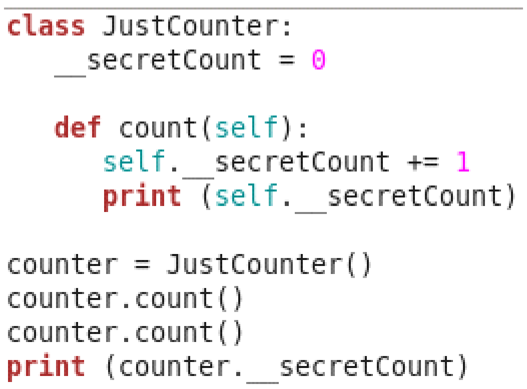
➢ List of operator :

+	__add__(self, other)	Addition
*	__mul__(self, other)	Multiplication

-	__sub__(self, other)	Subtraction
%	__mod__(self, other)	Remainder
/	__truediv__(self, other)	Division
<	__lt__(self, other)	Less than
<=	__le__(self, other)	Less than or equal to
==	__eq__(self, other)	Equal to
!=	__ne__(self, other)	Not equal to
>	__gt__(self, other)	Greater than
>=	__ge__(self, other)	Greater than or equal to
[index]	__getitem__(self, index)	Index operator
in	__contains__(self, value)	Check membership
len	__len__(self)	The number of elements
str	__str__(self)	The string representation

❖ **Data Hidding:** An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then will not be directly visible to outsiders.

Example:



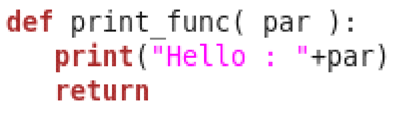
```
class JustCounter:
   __secretCount = 0

   def count(self):
      self.__secretCount += 1
      print (self.__secretCount)

counter = JustCounter()
counter.count()
counter.count()
print (counter.__secretCount)
```

❖ **Module:** A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

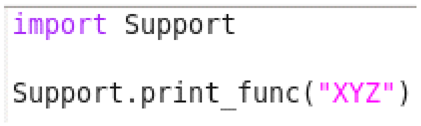
Example:

Support.py



```
def print_func( par ):
   print("Hello : "+par)
   return
```

Test.py



```
import Support

Support.print_func("XYZ")
```

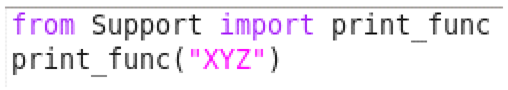
Python's **from** statement lets you import specific attributes from a module into the current namespace. The **from...import** has the following syntax

Syntax:

from modname import name1[, name2[, ... nameN]]

Example:

Test1.py



```
from Support import print_func
print_func("XYZ")
```

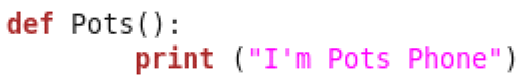
It is also possible to import all the names from a module into the current namespace by using the following import statement –

from modname import *

This provides an easy way to import all the items from a module into the current namespace.

❖ **<u>Package</u>:** A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Consider a file *Pots.py* available in Phone directory. This file has the following line of source code



```
def Pots():
        print ("I'm Pots Phone")
```

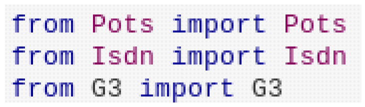
Similar, we have other two files having different functions with the same name as above. They are –

- *Phone/Isdn.py* file having function Isdn()
- *Phone/G3.py* file having function G3()

Now, create one more file __init__.py in the *Phone* directory –

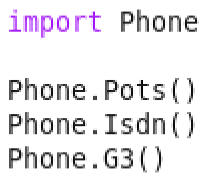
- Phone/__init__.py

To make all of your functions available when you have imported Phone, you need to put explicit import statements in __init__.py as follows



```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

After you add these lines to __init__.py, you have all of these classes available when you import the Phone package.



```
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

<u>Some popular and useful packages/librabries:</u>

Request	The most famous http library written by kenneth reitz.
Scrapy	If you are involved in webscraping then this is a must have library for you. After using this library you won't use any other.
wxPython	A gui toolkit for python. Alternative is tkinter.
Pillow	A friendly fork of PIL (Python Imaging Library). It is more user friendly than PIL and is a must have for anyone who works with images.
SQLAlchemy	A database library.
BeautifulSoup	This xml and html parsing library is very useful for beginners.
Twisted	The most important tool for any network application developer. It has a very beautiful api and is used by a lot of famous python developers.
NumPy	It provides many advance math functionalities to python.
SciPy	When we talk about NumPy then we have to talk about scipy. It is a library of algorithms and mathematical tools for python and has caused many scientists to switch from ruby to python.
matplotlib	A numerical plotting library. It is very useful for any data scientist or any data analyzer.
Pygame	This library will help you achieve your goal of 2d game development.
Pyglet	A 3d animation and game creation engine. This is the engine in which the famous python port of minecraft was made.
pyQT	A GUI toolkit for python. It is my second choice after wxpython for developing GUI's for my python scripts.
pyGtk	Another python GUI library. It is the same library in which the famous Bittorrent client is created.
Scapy	A packet sniffer and analyzer for python made in python.
pywin32	A python library which provides some useful methods and classes for interacting with windows.
nltk	Natural Language Toolkit – most people won't be using this one, but it's generic enough. It is a very useful library if you want to manipulate strings. But it's capacity is beyond that.

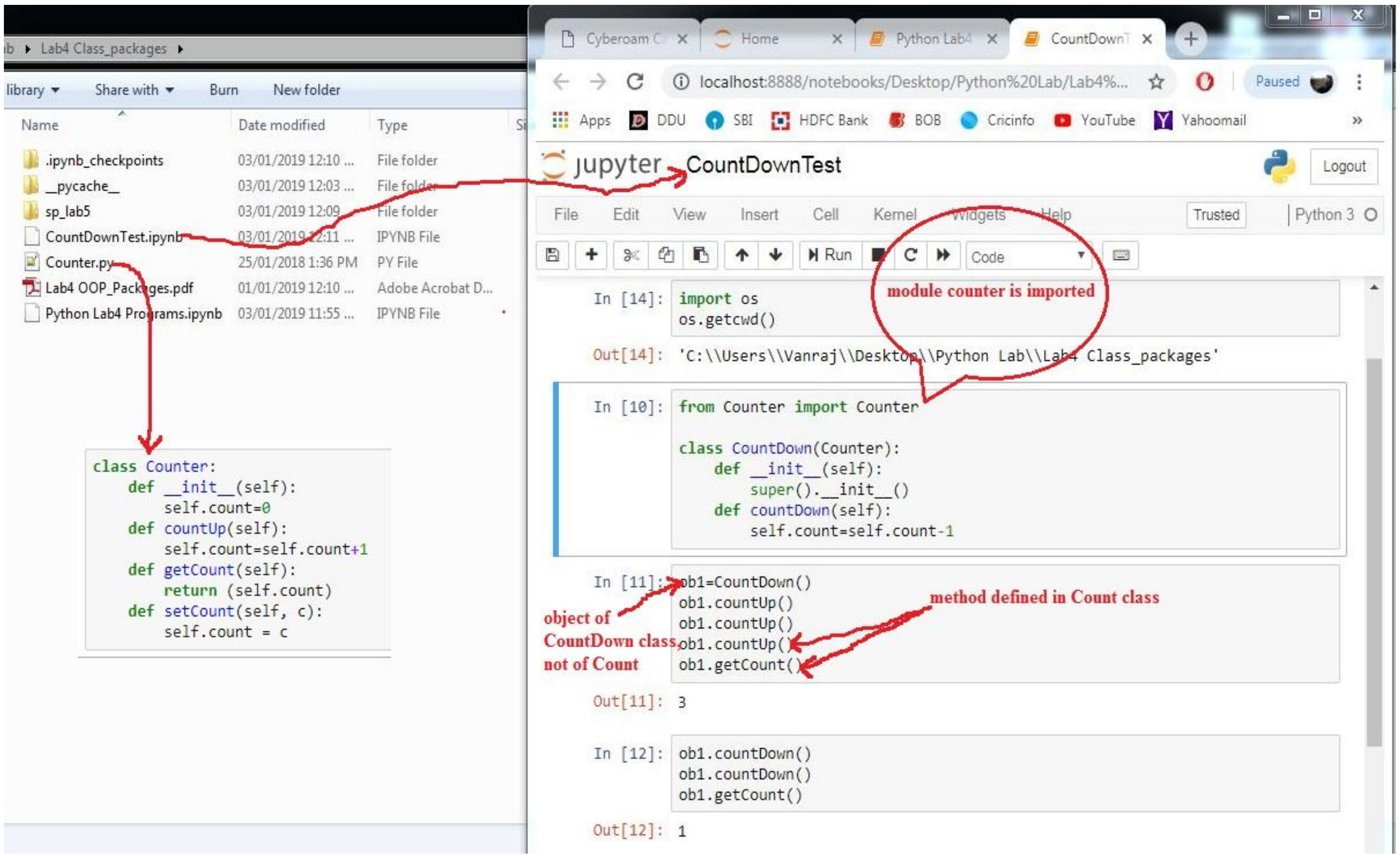
Tensorflow	Dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks
nose	A testing framework for python. It is used by millions of python developers.
Sympy	SymPy can do algebraic evaluation, differentiation, expansion, complex numbers, etc. It is contained in a pure Python distribution.
Ipython	It is a python prompt on steroids. It has completion, history, shell capabilities, and a lot more.

❖ **Exercise:**

[1]. Create a class Counter having one member variable count. Initialize it with zero with creation of each object and also write member method countUp to increase the value of count by one with every method call.

[2]. Import above module(consider that Counter class is defined in file P1.py) and derive a class CountDown from Counter class with one more method countDown to decrease a value of count by one with every method call.

Understanding for ex:[2]:


Lab4 Class_packages
library
Share with
Burn
New folder
Name
Date modified
Type
.ipynb_checkpoints 03/01/2019 12:10 ... File folder
__pycache__ 03/01/2019 12:03 ... File folder
sp_lab5 03/01/2019 12:09 File folder
CountDownTest.ipynb 03/01/2019 12:11 ... IPYNB File
Counter.py 25/01/2018 1:36 PM PY File
Lab4 OOP_Packages.pdf 01/01/2019 12:10 ... Adobe Acrobat D...
Python Lab4 Programs.ipynb 03/01/2019 11:55 ... IPYNB File
localhost:8888/notebooks/Desktop/Python%20Lab/Lab4%...
jupyter CountDownTest
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
module counter is imported
method defined in Count class
object of CountDown class not of Count

```
class Counter:
    def __init__(self):
        self.count=0
    def countUp(self):
        self.count=self.count+1
    def getCount(self):
        return (self.count)
    def setCount(self, c):
        self.count = c
```

```
In [14]: import os
         os.getcwd()

Out[14]: 'C:\\Users\\Vanraj\\Desktop\\Python Lab\\Lab4 Class_packages'

In [10]: from Counter import Counter

         class CountDown(Counter):
             def __init__(self):
                 super().__init__()
             def countDown(self):
                 self.count=self.count-1

In [11]: ob1=CountDown()
         ob1.countUp()
         ob1.countUp()
         ob1.countUp()
         ob1.getCount()

Out[11]: 3

In [12]: ob1.countDown()
         ob1.countDown()
         ob1.getCount()

Out[12]: 1
```

[3]. In above program override countUp method to increase the value of count by 2 with each method call.

[4]. Create class exam having three member variable to store the marks of three subject. Overload <, > and = operator to compare two object of Exam class based on total marks. Also write executable statements to test Exam class.

[5]. Create a package Calculator having four files add.py, sub.py, mul.py and div.py with functions to perform addition, subtraction, multiplication and division respectively. Import Calculator package in your code and implement simple calculator.