**K. J. Somaiya College of Engineering, Mumbai-77**

**Department of Computer Engineering**

**Roll No.: 1911027**

**Experiment No. 3**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

**Problem Statement:** Create a package 'myPackage' which contains a class myMath. The class contains the following static methods.
i) power (x, y) – to compute $x^y$
ii) fact (x) – to compute x!
Write a program to find the following series.
cos (x),sin(x), tan(x), (Do not make use of inbuilt functions. Use the functions of user defined class MyMath by importing mypackage.)
Similarly implement the same things using module (myModule)

**AIM:** Use concepts of modules and packages

---

**Expected OUTCOME of Experiment:**

**CO1 : Describe the numbers, Math functions, Strings, List, Tuples and Dictionaries in Python.**

**CO2 : Interpret different Decision Making statements, Functions, Object Oriented programming in Python.**

---

**Books/ Journals/ Websites referred:**


1) https://www.programiz.com/python-programming/package
2) https://www.programiz.com/python-programming/modules
3) https://www.tutorialspoint.com/packages-in-python
4) https://realpython.com/python-modules-packages/
5) https://www.python-course.eu/python3_packages.


---

## Pre Lab/ Prior Concepts:

## Modules :

Modules refer to a file containing Python statements and definitions. A file containing Python code, for example: example.py, is called a module, and its module name would be example. We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code. We can define our most used functions in a module and import it, instead of copying their definitions into different programs. A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. 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.

Let us create a module. Type the following and save it as example.py.

```
def add(a, b):
    """This program adds two
    numbers and return the result"""
    result = a + b
    return result
```

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

### How to import modules in Python?

We can import the definitions inside a module to another module or the interactive interpreter in Python. We use the import keyword to do this. To import our previously defined module example, we type the following in the Python prompt.

```
>>> import example
```

This does not import the names of the functions defined in example directly in the current symbol table. It only imports the module name example there.

Using the module name we can access the function using the dot . operator. For example:

```
>>> example.add(4,5.5)
9.5
```

Python has tons of standard modules. These files are in the Lib directory inside the location where you installed Python. Standard modules can be imported the same way as we import our user-defined modules. There are various ways to import modules.

**Python import statement :**

We can import a module using the import statement and access the definitions inside it using the dot operator as described above. Here is an example.

```
# import statement example
# to import standard module math

import math
print("The value of pi is", math.pi)
```

When you run the program, the output will be :

```
The value of pi is 3.141592653589793
```

### Import with renaming :

We can import a module by renaming it as follows :

```
# import module by renaming it

import math as m
print("The value of pi is", m.pi)
```

We have renamed the math module as m. This can save us typing time in some cases. Note that the name math is not recognized in our scope. Hence, math.pi is invalid, and m.pi is the correct implementation.

**Python Module Search Path :** While importing a module, Python looks at several places. Interpreter first looks for a built-in module. Then(if built-in module not found), Python looks into a list of directories defined in sys.path. The search is in this order.

- The current directory.
- PYTHONPATH (an environment variable with a list of directories).
- The installation-dependent default directory.

### Packages :

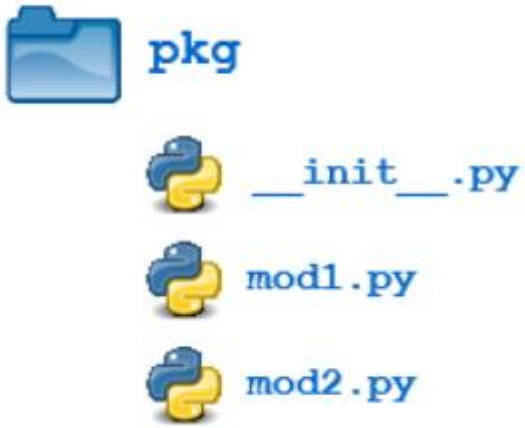
A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on. We don't usually store all of our files on our computer in the same location. We use a well-organized hierarchy of directories for easier access. Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files. As our application program grows larger in

size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear. Similarly, as a directory can contain subdirectories and files, a Python package can have sub-packages and modules. Suppose you have developed a very large application that includes many modules. As the number of modules grows, it becomes difficult to keep track of them all if they are dumped into one location. This is particularly so if they have similar names or functionality. You might wish for a means of grouping and organizing them. Packages allow for a hierarchical structuring of the module namespace using dot notation. In the same way that modules help avoid collisions between global variable names, packages help avoid collisions between module names.

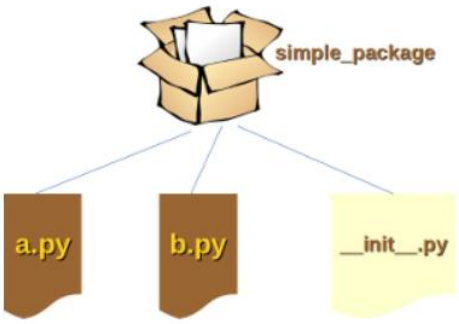
**Package Initialization :**

If a file named __init__.py is present in a package directory, it is invoked when the package or a module in the package is imported. This can be used for execution of package initialization code, such as initialization of package-level data.

Now if we create a package as pkg and add 2 modules in it as mod1 and mod2 and __init__.py then package structure will look like this.



## A simple example :




We will demonstrate with a very simple example how to create a package with some Python modules. First of all, we need a directory. The name of this directory will be the name of the package, which we want to create. We will call our package "simple_package". This directory needs to contain a file with the name __init__.py. This file can be empty, or it can contain valid Python code. This code will be executed when a package is imported, so it can be used to initialize a package, e.g. to make sure that some other modules are imported or some values set. Now we can put all of the Python files which will be the submodules of our module into this directory. We create two simple files a.py and b.py just for the sake of filling the package with modules.

The content of a.py :

```
def bar():
   print("Hello, function 'bar' from module 'a' calling")
```

The content of b.py :

```
def foo():
   print("Hello, function 'foo' from module 'b' calling")
```

There is a way to automatically load these modules. We can use the file __init__.py for this purpose. All we have to do is add the following lines to the so far empty file __init__.py:

```
import simple_package.a

import simple_package.b
```

Now if we write the following lines of code outside the package in a file named say test.py.

import simple_package

simple_package.a.bar()

simple_package.b.foo()

Now if we run that test.py we will get output as :

Hello, function 'bar' from module 'a' calling

Hello, function 'foo' from module 'b' calling

**Program**:

**Package myPackage:**

- **__init__.py**
- **myModule.py**
    - **class myMath**
        - **static method : power**
        - **static method : fact**

**series.py = driver function**

**__init__.py :**

```
import myPackage.myModule
```

## myModule.py :

```
class myMath:
    @staticmethod
    def power(x, y):
        if (y == 0):
            return 1
        elif (x == 1):
            return 1
        else:
            pow = x
            for i in range(0, y - 1):
                pow = pow * x
        return pow
    @staticmethod
    def fact(x):
        if (x == 0 or x == 1):
            return 1
        else:
            fact = 1
            for i in range(1, x + 1):
                fact = fact * i
            return fact
```

## series.py :

```
import myPackage.myModule
def coscal(deg,num):
    cos=float(0)
    for i in range(0, num):
        if(i%2==0):
            cos=cos+(myPackage.myModule.myMath.power(deg, (2 *
i)) / myPackage.myModule.myMath.fact((2 * i)))
        else:
            cos=cos-(myPackage.myModule.myMath.power(deg, (2 *
i)) / myPackage.myModule.myMath.fact(
                (2 * i)))
    return cos
def sincal(deg,num):
    sin=float(0)
    for i in range(0, num):
        if (i%2==0):
```

```
            sin=sin+(myPackage.myModule.myMath.power(deg, (2 *
i) + 1) / myPackage.myModule.myMath.fact((2 * i) + 1))
        else:
            sin=sin-(myPackage.myModule.myMath.power(deg, (2 *
i) + 1) / myPackage.myModule.myMath.fact((2 * i) + 1))
    return sin
sin=float(0)
cos = float(0)
print("-----------------------------------------------------------
-------------------------------------")
deg=float(input("Enter degrees : "))
print("-----------------------------------------------------------
-------------------------------------")
num=int(input("Enter number of terms upto which series is to be
computed : "))
print("-----------------------------------------------------------
-------------------------------------")
pi=22/7
temp=deg
deg=deg*pi/180
sin=sincal(deg,num)
cos=coscal(deg,num)
if(temp==90):
    tan="undefined"
else:
    tan=sin/cos
print("Sin(",temp,") = ",sin)
print("Cos(",temp,") = ",cos)
print("Tan(",temp,") = ",tan)
print("-----------------------------------------------------------
-------------------------------------")
```

**Output:**

```
D:\KJSCE\OSTPL Lab>python series.py
-----------------------------------------------------------------------------------------------
Enter degrees : 0
-----------------------------------------------------------------------------------------------
Enter number of terms upto which series is to be computed : 10
-----------------------------------------------------------------------------------------------
Sin( 0.0 ) =  0.0
Cos( 0.0 ) =  1.0
Tan( 0.0 ) =  0.0
-----------------------------------------------------------------------------------------------
```

```
D:\KJSCE\OSTPL Lab>python series.py
-----------------------------------------------------------------------------------------------
Enter degrees : 30
-----------------------------------------------------------------------------------------------
Enter number of terms upto which series is to be computed : 10
-----------------------------------------------------------------------------------------------
Sin( 30.0 ) =  0.5001825021996698
Cos( 30.0 ) =  0.8659200104474302
Tan( 30.0 ) =  0.5776313010034496
-----------------------------------------------------------------------------------------------
```

```
D:\KJSCE\OSTPL Lab>python series.py
-----------------------------------------------------------------------------------------------
Enter degrees : 45
-----------------------------------------------------------------------------------------------
Enter number of terms upto which series is to be computed : 10
-----------------------------------------------------------------------------------------------
Sin( 45.0 ) =  0.7073302780849811
Cos( 45.0 ) =  0.7068832136245868
Tan( 45.0 ) =  1.0006324445845898
-----------------------------------------------------------------------------------------------
```

```
D:\KJSCE\OSTPL Lab>python series.py
-----------------------------------------------------------------------------------------------
Enter degrees : 60
-----------------------------------------------------------------------------------------------
Enter number of terms upto which series is to be computed : 10
-----------------------------------------------------------------------------------------------
Sin( 60.0 ) =  0.8662360750607193
Cos( 60.0 ) =  0.4996349289865548
Tan( 60.0 ) =  1.7337380251171945
-----------------------------------------------------------------------------------------------
```

**Conclusion: By performing this experiment we understood the concept of packages in python. We also understood modules in python. Successfully imported a package in a program and also used functionalities present in that function. We also touched some concepts of classes and static methods.**

**Date: 24 / 2 / 2021** **Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1.Which of these definitions correctly describes a module?

a) Denoted by triple quotes for providing the specification of certain program elements

b) Design and implementation of specific functionality to be incorporated into a program

c) Defines the specification of how it is to be used
d) Any program that reuses code
**ANS) b) Design and implementation of specific functionality to be incorporated into a program**

2.Which of the following is not an advantage of using modules?
a) Provides a means of reuse of program code
b) Provides a means of dividing up tasks
c) Provides a means of reducing the size of the program
d) Provides a means of testing individual parts of the program
**ANS) c) Provides a means of reducing the size of the program**

3.Program code making use of a given module is called a _____ of the module.
a) Client
b) Docstring
c) Interface
d) Modularity
**ANS) a) Client**

4.In top-down design every module is broken into same number of submodules.
a) True
b) False
**ANS) b) False**

5.What will be the output of the following Python code?

```
#mod1
def change(a):
    b=[x*2 for x in a]
    print(b)
#mod2
def change(a):
    b=[x*x for x in a]
    print(b)
from mod1 import change
from mod2 import change
#main
s=[1,2,3]
change(s)
```

a) [2,4,6]
b) [1,4,9]
c)[2,4,6]
[1,4,9]
d) There is a name clash

**ANS) d) There is a name clash**

6. Which of the following is false about “import modulename” form of import?
a) The namespace of imported module becomes part of importing module
b) This form of import prevents name clash
c) The namespace of imported module becomes available to importing module
d) The identifiers in module are accessed as: modulename.identifier

**ANS) a) The namespace of imported module becomes part of importing module**

7. What will be the output of the following Python code?

```
from math import factorial
print(math.factorial(5))
```

a) 120
b) Nothing is printed
c) Error, method factorial doesn’t exist in math module
d) Error, the statement should be: print(factorial(5))

**ANS) d) Error, the statement should be: print(factorial(5))**