Mahindra École Centrale

CS101 Lab-13
Modules and Packages

Aim

The main aim of this lab is to introduce modular programming in Python. Different ways of importing namespace from modules will be introduced. Some built-in modules in Python will be introduced. Using *dir()* function to find the functions and attributes of a module. Finally introduction to *packages*, which contains multiple modules for efficient modular programming.

Outcome

At the end of this lab students are expected to be familiar with creating and using modules and packages for efficient programming.

Prerequisites

Students are expected to have completed all the tasks related to matrix operations (addition, multiplication etc.) and saved their programme.

Tasks

Task-1 Creating a module

A module in Python is nothing but a file containing variable and function definitions. In that sense, every Python file is also a module. Go ahead and create your first module in Python, *matrix*. For this copy paste all the functions you have written for matrix operations into a new file and save it as *matrix.py*. Do not forget to add the extension *.py* whenever you are creating a module. You can add two or three additional functions into the module such as finding the trace of a matrix, checking whether a matrix is square, checking whether a matrix is identity matrix and finding the transpose of a matrix.

Task-2 Importing a module

Use different methods to import a module and its namespace. For this, open a new Python file and save it as *main.py*. Try to execute the functions in the *matrix* module by importing them in the *main* module.

```
1  import matrix
2  A = [[1,2,3],[4,5,6],[7,8,9]]
3  B = [[3,4,3],[4,7,2],[5,6,10]]
4  print matrix.matrix_mult(A,B)     #matrix_mult is a ↩
       function defined in matrix for multiplication
```

```
1  from matrix import matrix_mult
2  A = [[1,2,3],[4,5,6],[7,8,9]]
3  B = [[3,4,3],[4,7,2],[5,6,10]]
4  print matrix_mult(A,B)     #matrix_mult is a function ↩
       defined in matrix for multiplication
5  print matrix_add(A,B) #matrix_add is a function in ↩
       matrix for addition
```

Note: Your console remembers all the code you executed before. Hence please close the console each time before trying a different import style.

Will the second function call will give any error? Why?

```
from matrix import matrix_mult as my_mat_mult
print my_mat_mult(A,B)     #matrix_mult is a function ↩
    defined in matrix for multiplication
```

```
from matrix import matrix_mult as matrix_mult
print matrix_mult(A,B)     #matrix_mult is a function ↩
    defined in matrix for multiplication
```

Now import the built-in *random* module also into the *main.py*. Generate random matrices with the help of the *random.randint()* function then use the multiplication function in the *matrix* module to multiply them.

Task-3 Exploring modules

Use the *dir()* function to explore the attributes and functions in a module (Use the console).

```
import matrix
dir(matrix)
```

Use the same function to explore the built-in Python modules *os,sys* and *math*. Find the *path* attribute in the *sys* module. It lists all the *paths* where Python searches for a module when it encounters the *import* statement. Copy your *matrix.py* module and paste it inside the *D:* drive and rename as matrix1.py. Now try

```
import matrix1
```

Any error is coming? Why? Add *D:* to the module search path and try to import the module again. **Clue: *sys.path* is a list of strings. How will add a new element to an existing list?**

Task-4 Executing modules as a script

Add this line as the last statement in the *matrix.py* module.

```
print 'This is matrix module, now my name is',__name__
```

Now run the *matrix.py* file (not *main.py*). What output are you seeing? Now run the *main.py* module (matrix should be imported in main). Are you observing any difference in the print? Now add this line to your *main.py* file.

```
print 'This is main module, now my name is',__name__
```

What do you observe? What do you think about the name given to a module by the Python interpreter when it is executed as the top file and when it is imported?
Now there is a requirement. If the *matrix* module is executed as the top file, it has to generate two random matrices and multiply them. If it is imported to another module, this should not happen. How will you do it?

Clue: You can use the __name__ attribute to find whether a module is executed as top file or is imported.

Task-5 Packages

Create a new folder *my_package* inside the folder where you have saved the *main.py* file. Copy paste your *matrix.py* file inside this new folder and rename it as *my_matrix.py*. Now try to import it in the *main* module.

```
import my_package.my_matrix
```

Are you observing any error? Create an empty file *__init__.py* inside the *my_package* folder. Now run the *main.py* again. Here *my_package* is a *package* and *my_matrix.py* is a sub-module. Create one more sub-module called *my_circle.py* which contains functions to calculate the area and perimeter of a circule.

Use the *math* module to get the *pi* value. Use different methods to import functions from a submodule to the *main.py* module.

```
1  import my_package.my_matrix
2  print my_package.my_matrix.matrix_mult(A,B)
```

```
1  from my_package.my_matrix import matrix_mult
2  print matrix_mult(A,B)
```

```
1  from my_package import my_matrix.matrix_mult
2  print matrix.matrix_mult(A,B)
```

```
1  from my_package import my_matrix.matrix_mult as my_matrix_mult
2  print my_matrix_mult(A,B)
```

Try to import *my_circle* module inside the *my_matrix* module and vice-versa. Print the names of the imported modules.