Lab Exercises 5 (Based on Modules, Packages & File handling)

1. Write a Python program to read an entire text file.

2. Write a program that counts lines and characters in a file. With your favorite text editor, code a Python module called **mymod.py**, which exports three top-level names:

   a) **A countLines(name) function** that reads an input file and counts the number of lines in it

   b) A countChars(name) function that reads an input file and counts the number of characters in it

   c) A test(name) function that calls both counting functions with a given input filename.

   All three mymod functions should expect a filename string to be passed in.

   Now, test your module interactively, using import and name qualification to fetch your exports.

3. Test your mymod module from Exercise 2 interactively, by using from to load the exports directly, first by name, then using the from* variant to fetch everything.

4. Now, add a line in your mymod module that calls the test function automatically only when the module is run as a script, not when it is imported The line you add will probably test the value of __name__ for the string "__main__", as shown in this unit. Try running your module from the system command line; then, import the module and test its functions interactively.

5. Write a second module**, myclient.py**, which imports mymod and tests its functions; run myclient from the system command line. If myclient uses from to fetch from mymod, will mymod's functions be accessible from the top level of myclient? What if it imports with import instead? Try coding both variations in myclient and test interactively, by importing myclient .

6. *Package imports*. Finally, import your file from a package. Create a subdirectory called mypkg nested in a directory on your module import search path, move the mymod.py module file you created in exercises 2 or 4 into the new directory, and try to import it with a package import of the form: import mypkg.mymod.

7. Experiment with module reloads: perform the tests in the changer.py example, changing the called function's message and/or behavior repeatedly, without stopping the Python interpreter. Depending on your system, you might be able to edit changer in another window.