Python Programming

by Narendra Allam Copyright 2019

Chapter 6

Modules

Topics Covering

- Python Code files
  - import
  - from import
  - import *
- Python Packages
  - Directory vs Package
  - **init**.py
  - **all**
  - namespace
- Preventing unwanted code execution
  - **_name**
- Recursive imports
  - Hiding symbols from import *

A module in python is a set of re-usable classes, functions and variables. There are two types of modules in python

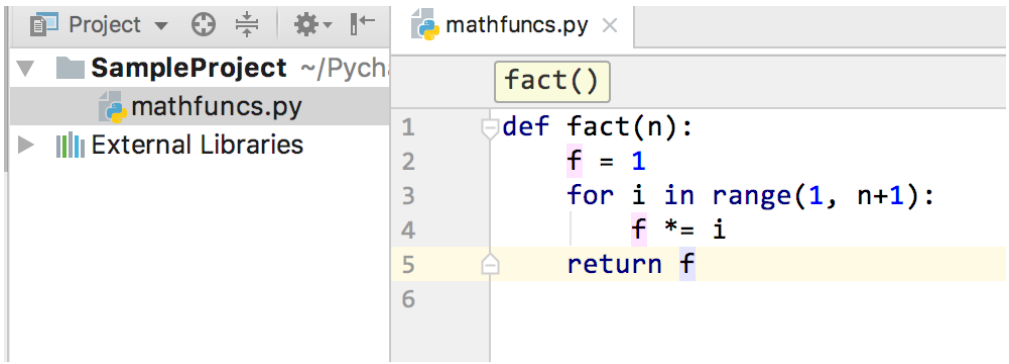
```
1. Python Code Files
2. packages
```

1. Python Code files:

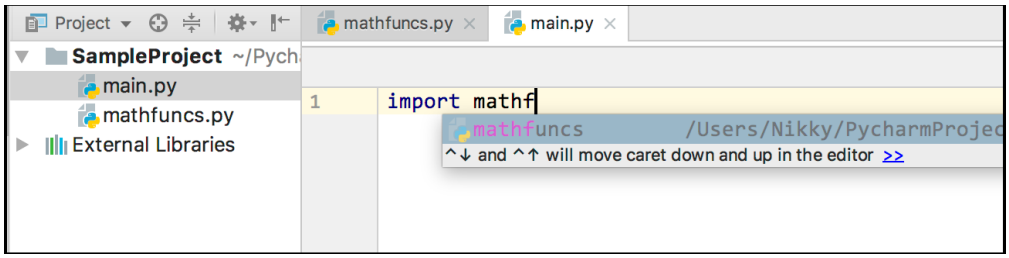
“Every python code file (‘.py’ file) is a module.”

“A module in python is a set of re-usable classes, functions and variables.”

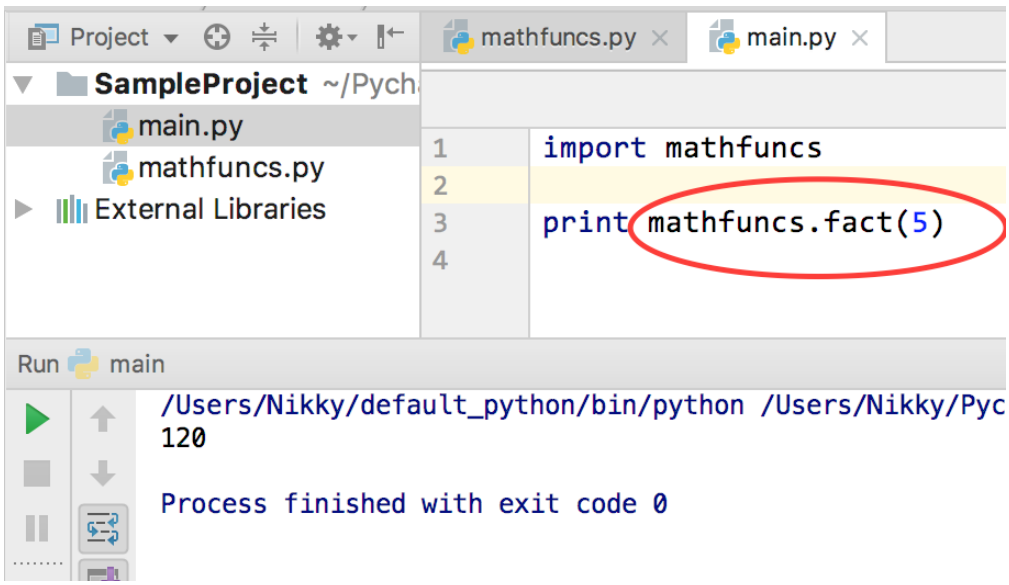
Let’s create a project, ‘SampleProject’ in pycharm. There is a python file ‘mathfuncs.py’ and we defined a function ‘fact’ in it.



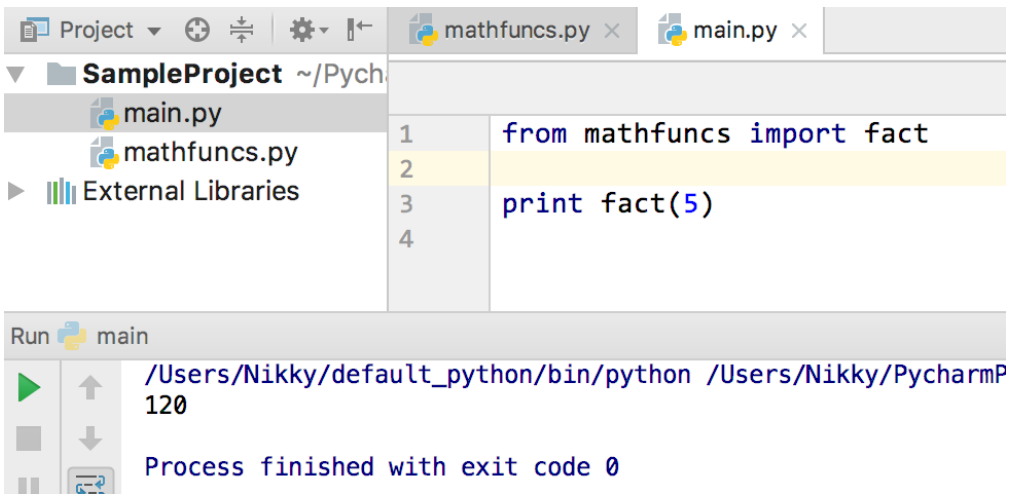
If we want to reuse the function ‘fact’, in any other python file, we have to import the file as module, using import statement.



To access 'fact' function, we have to use '.' (dot) after module name.



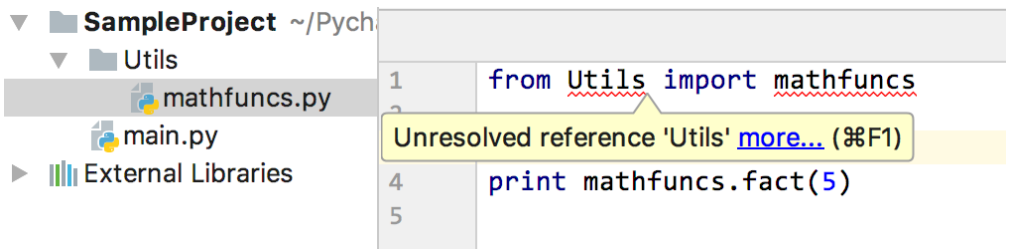
Another way of importing. 'from' keyword is used to import only specific functions in the module 'mathfuncs' without using module name.



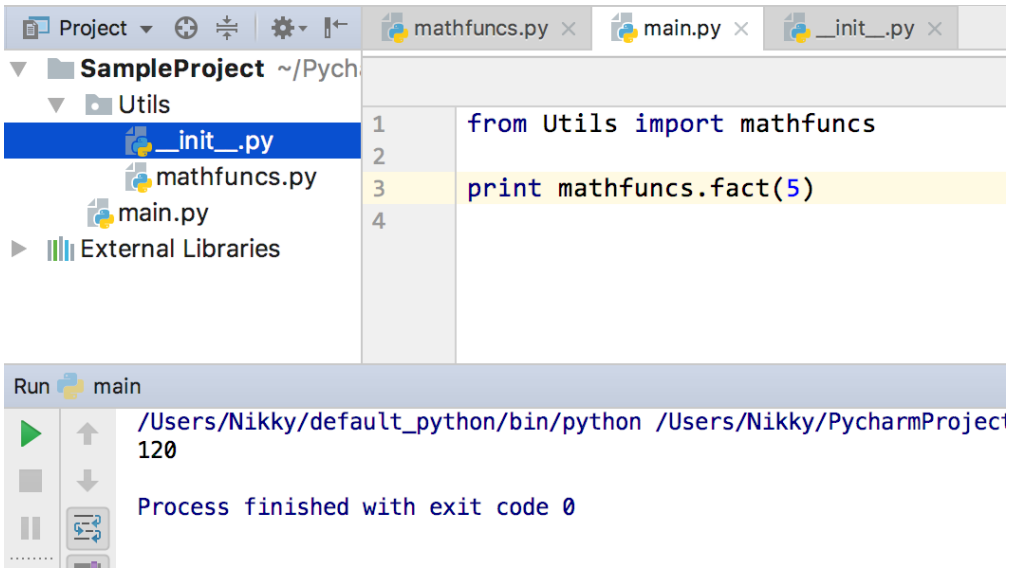
using from import func1, func2, ... we can import multiple functions from a module.

2. Package:

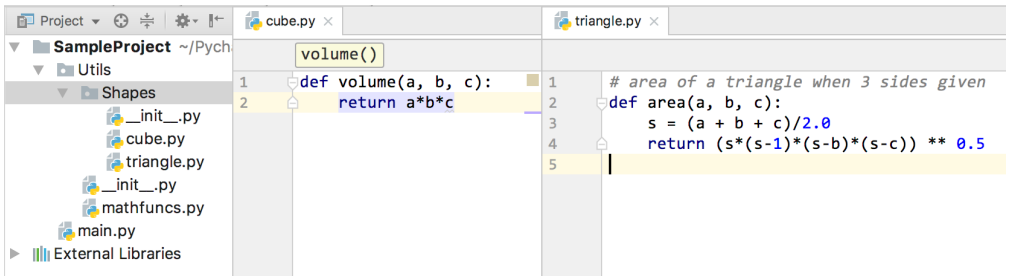
Package is folder in the python project folder structure, which is having **init**.py. This is the main difference between a folder and a package in python. Packages are modules. lets create a folder Utils in the Sample Project.



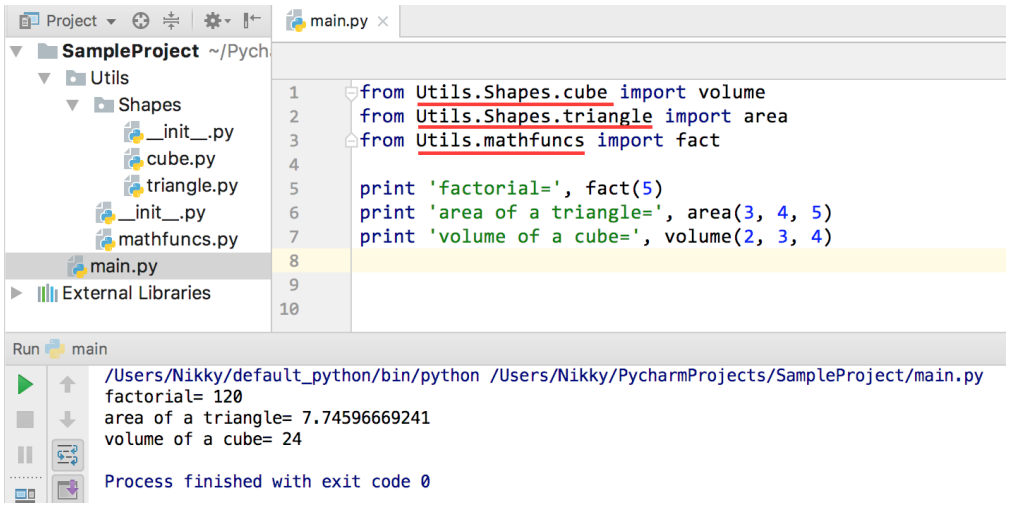
Let's place mathfuncs.py inside Utils folder. Now, if we try to import fact in main.py, we see above error 'No module named Utils'. Because Utils is just a folder, not a module. Only package or a python file is importable. To convert a folder to a package, explicitly we have to create **init**.py file, under Utils folder. E.g., all the functions are available, if we have module, 'fact' is available under 'mathfuncs' module. • All the file names are available under a package, to other files. • All the function names, class names and variable names are available to other files from a python file. Because, both are modules. A module has a namespace." A symbol table is maintained to each module, to group all the names under one roof, which is called namespace." If we can access 'mathfuncs', we can also access 'fact' and If we can access 'Utils', we can also access 'mathfuncs', as 'mathfuncs' and 'Utils' are modules and they have 'fact' and 'mathfuncs' in their namespace.



We did not get the error this time as, Utils has been converted to a package. **init**.py is just an empty file, which makes the folder as a package. But there are other uses too. Let's take a little complex project structure,



Shapes is another package with two files, cube.py and triangle.py. volume () and area () are the functions inside those files respectively. Now, how do we access volume () and area () from main.py

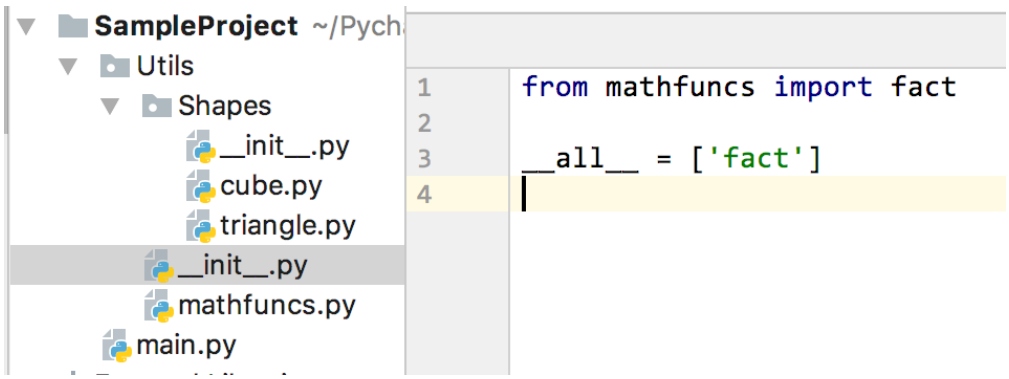


we have to use the long path name. Some developers do not want to expose the intermediate names, like Shapes, cube, triangle etc. What if, we could access all the functions directly from Utils namespace. If we export fact() to Utils name space we can directly access fact from Utils as below.

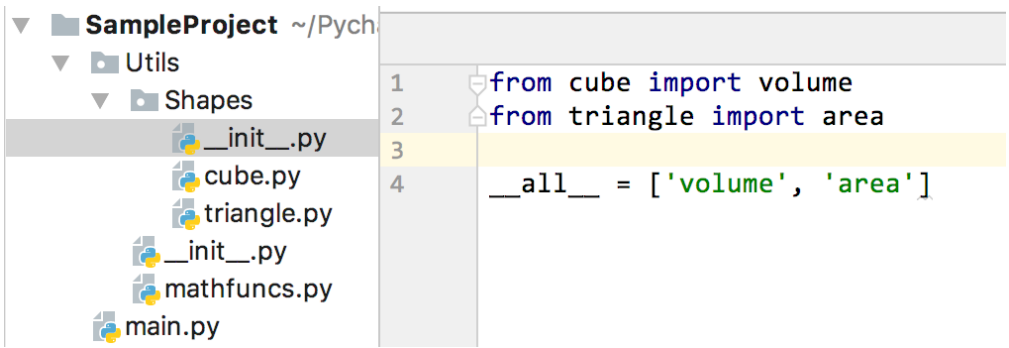
```
from Utils.Shapes.cube import volume
from Utils.Shapes.triangle import area
# from Utils.mathfuncs import fact
from Utils import fact

print 'factorial=', fact(5)
print 'area of a triangle=', area(3, 4, 5)
print 'volume of a cube=', volume(2, 3, 4)
```

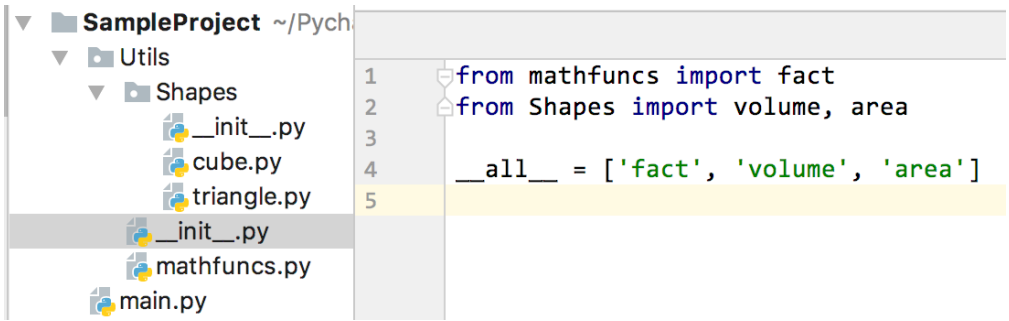
This is where we need **init**.py and **all** built-in variable.



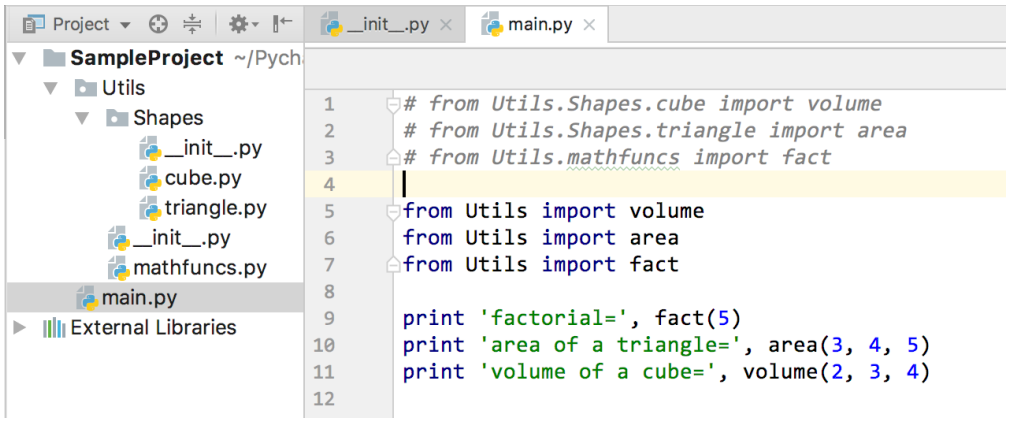
First, we have to import all symbols to **init**.py then add those symbols to **all**. Now all those symbols in **all** are available directly in Utils. To export all functions from Shapes to Utils namespace we have to make changes in both **init**.py files, one is in Shapes and another one is in Utils. **init**.py file acts as a bridge to export symbols to next higher levels, this reduces so much complexity when there are complex project structures. Let's make changes to Shapes/**init**.py.



From now, volume, and area are available directly in Shapes namespace. Let's import them from Shapes and export to Utils namespace.

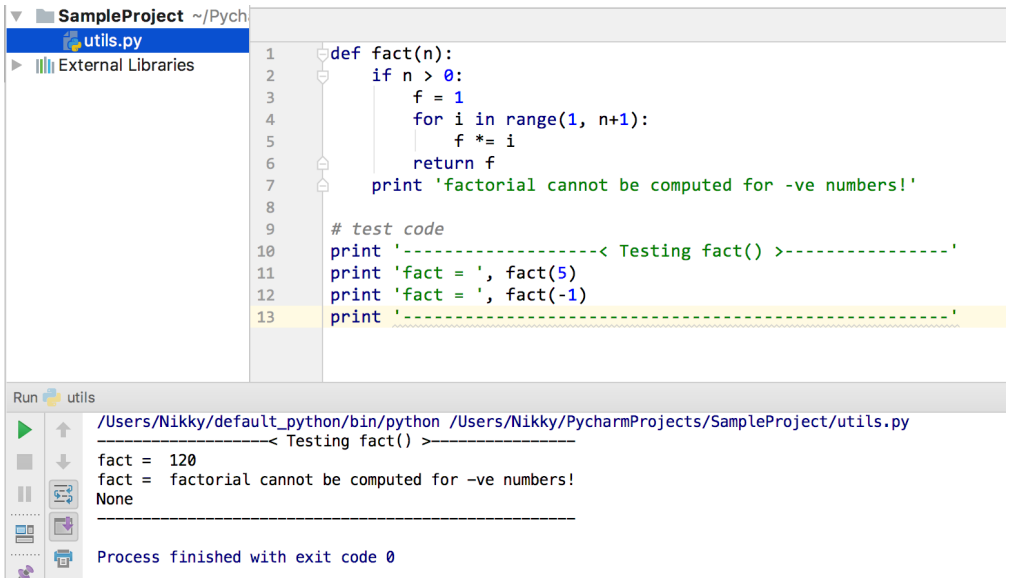


If we observe, we are actually exporting symbols from leaf level to root level in a project structure, by just connecting each level with **init**.py and **all**. Now, we can directly import all the functions from Utils as below.

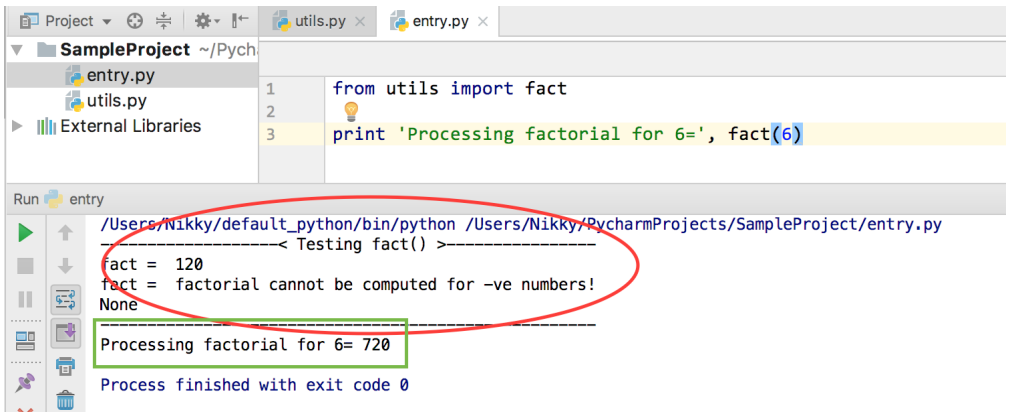


3. Preventing execution of unwanted code

In the below example. I have developed a function fact() and tested it in the same file.



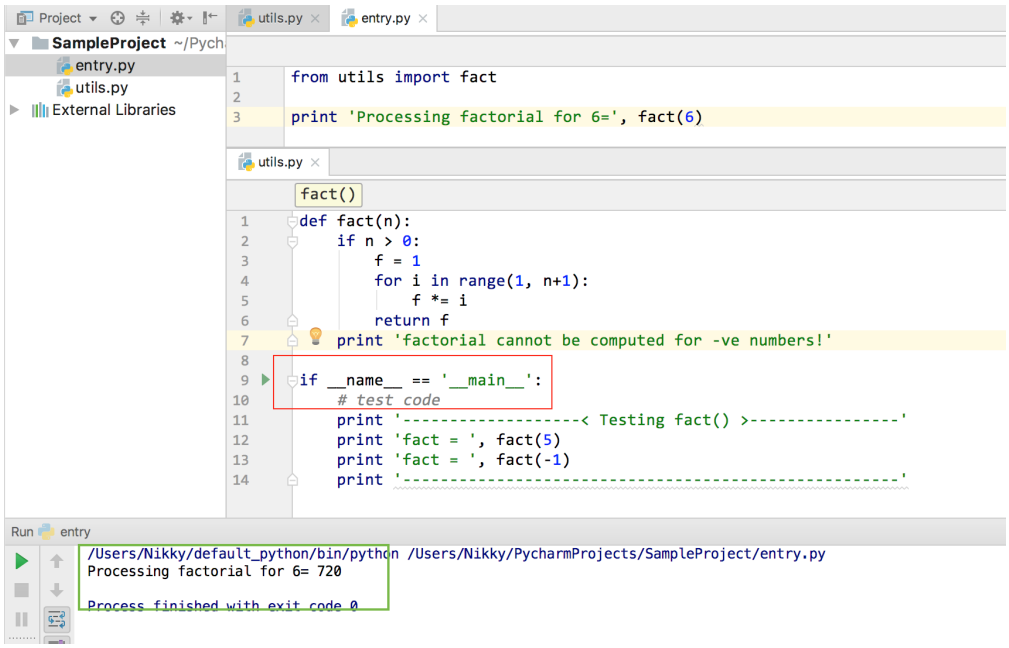
Now I want to reuse the same function fact() in another module called entry.py. I imported fact into entry.py and executed some code.



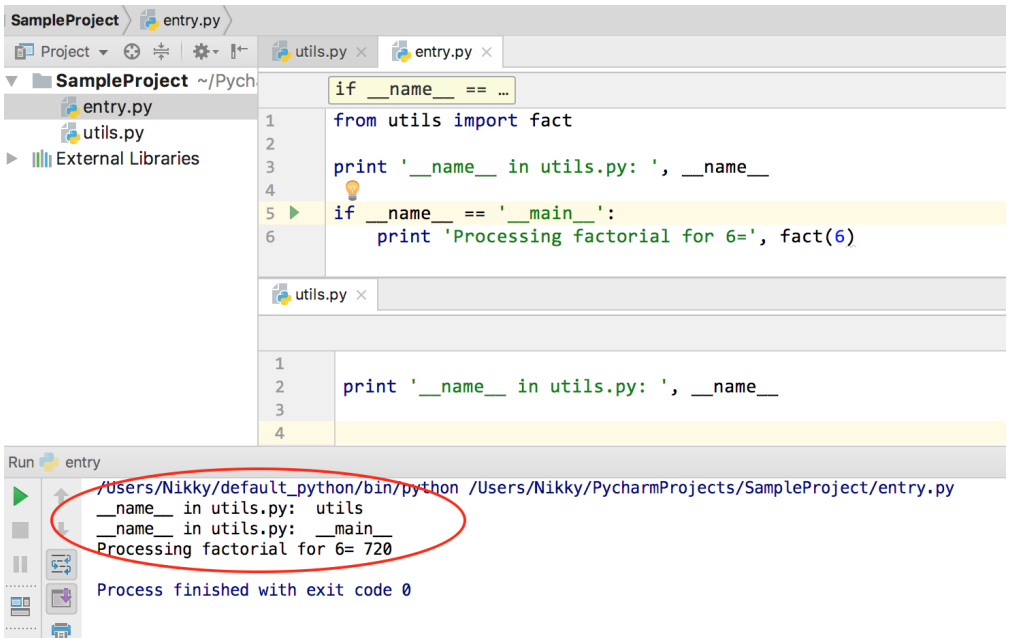
Why we are seeing unwanted output if we want to execute entry.py? Because, all statements in a module are executed when module is loading first time. When we are importing fact from Utils, all the statements (test code) are executed once.

How to prevent this? We should use **name**.

name __: Within a module, the module's name (as a string) is available as the value of the global variable. Every module has a separate __name global variable. All the global statements should be conditionally executed using **name**, unless it is really required.

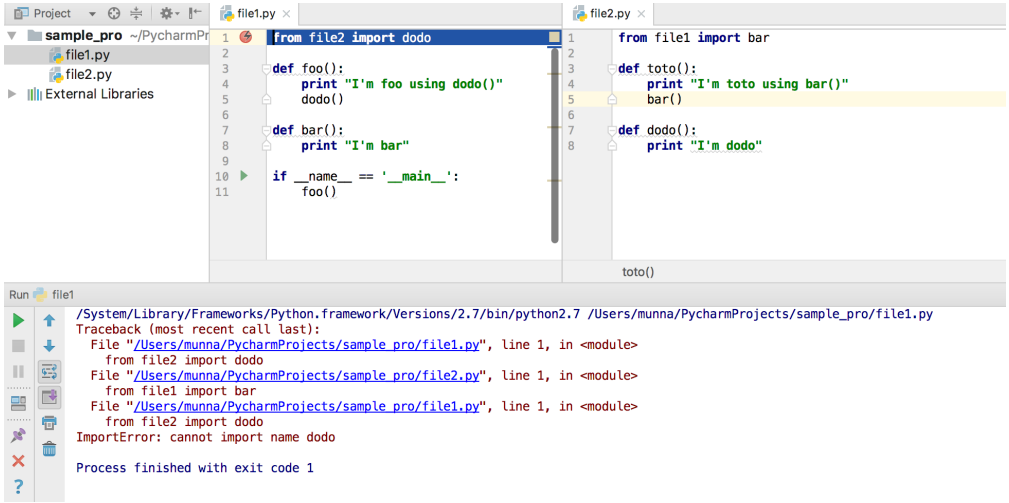


Global variable, **name**'s value is '**main**' in the start-up module of every project. In all other modules **name** value is set to its module name. Now if we execute entry.py we do not get the unwanted output. Let's run entry.py and print **name** value in both the modules. Check the output.

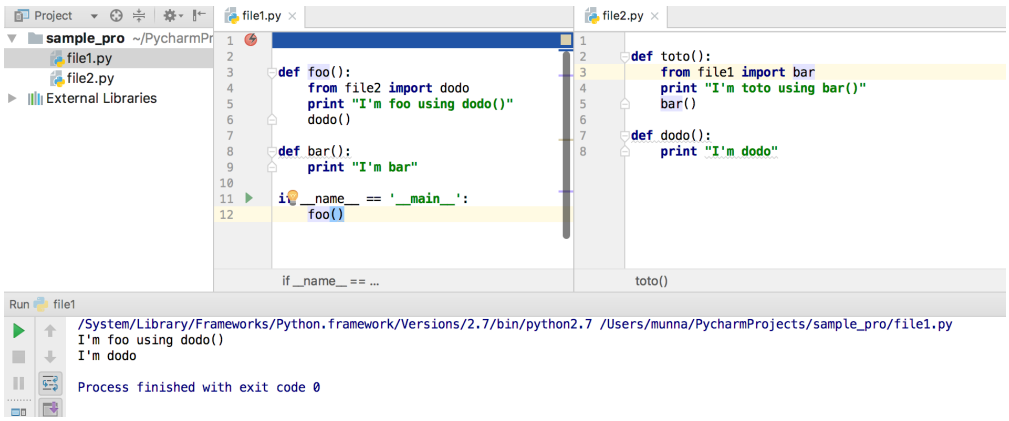


It is a good practice to keep all the global statements, which are not part of any function or class scope, inside if **name** == '**main**': block, which prevents unwanted code execution.

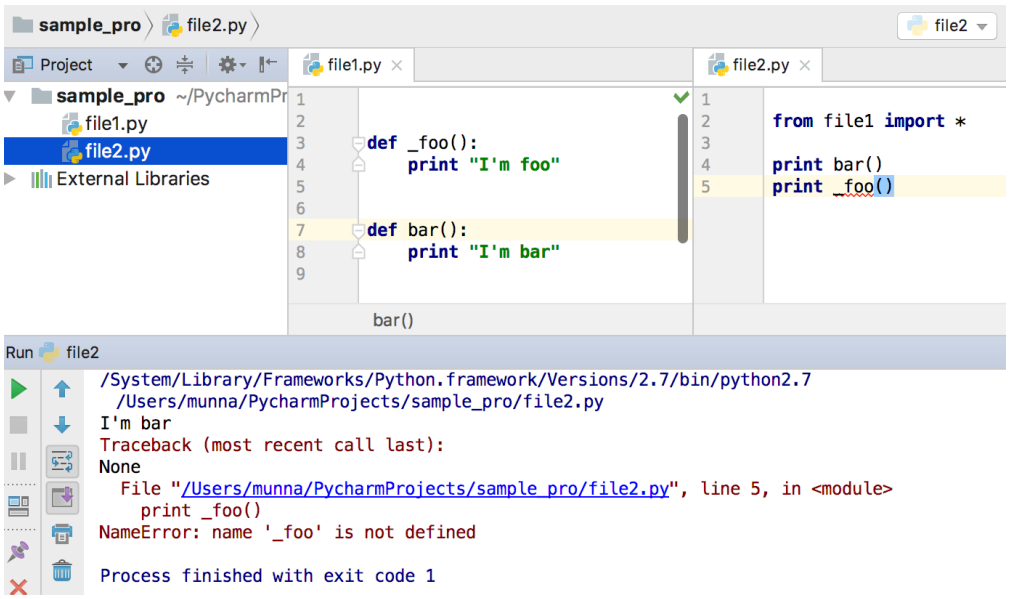
4. Recursive imports:



In the below example. file1.py has foo() and bar() functions. file2.py has toto() and dodo() functions. When file1.py import dodo(), file2.py import bar() we get a recursive imports problem as below. To avoid this problem, we should narrow the scope of imports.



Keep 'Import bar' statement inside the toto() function of file2.py and similarly , keep 'import dodo' statement inside foo() function as below.



5. Hiding symbols from import *

We can hide functions, classes any identifiers from import *, by prefixing with '-' (underscore). Look at the above code, file2.py trying to import everything from file1.py, but failed to import __foo(), as it is prefixed with underscore.

Interview Questions:

1. What is **name**
2. What is the use of **all**
3. How do you implement import *
4. How to avoid recursive imports
5. What is namespace in python
6. Difference between package and folder?
7. What is a module in python?