

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

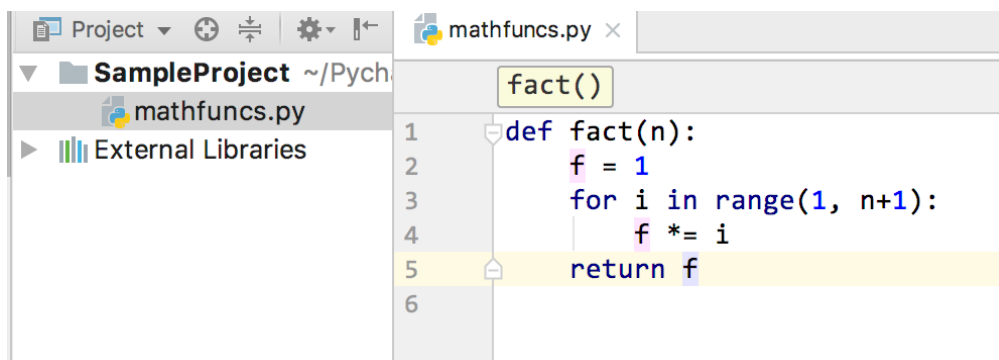
1. Python Code Files
2. packages

1. Python Code files:

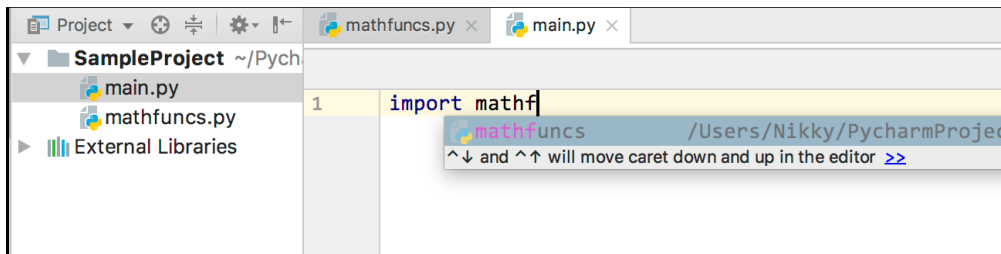
“Every python code file (‘.py’ file) is a module.”

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

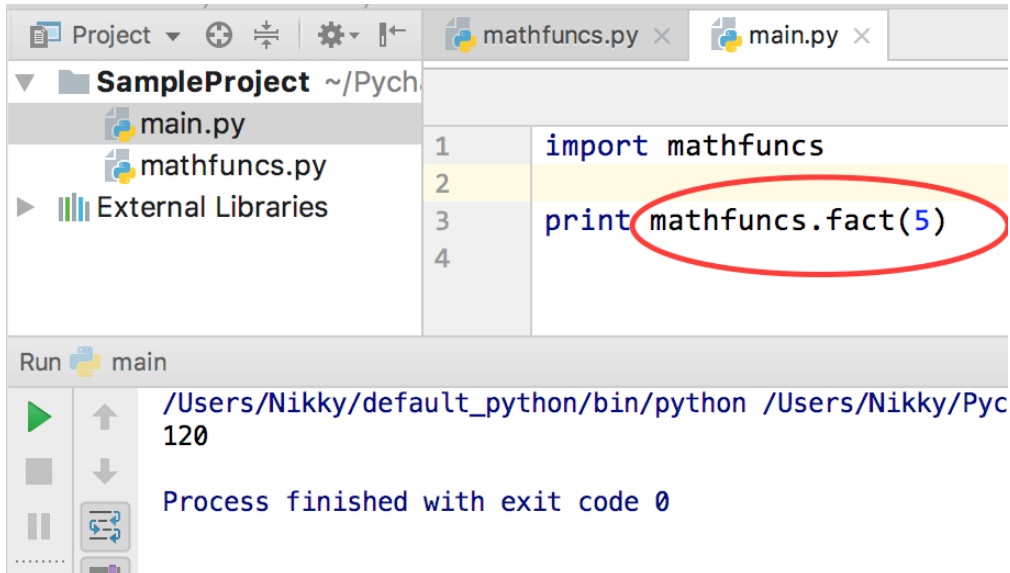
Let’s create a project, ‘SampleProject’ in pycharm. There is a python file ‘mathfuncs.py’ and we defined a function ‘fact’ in it.



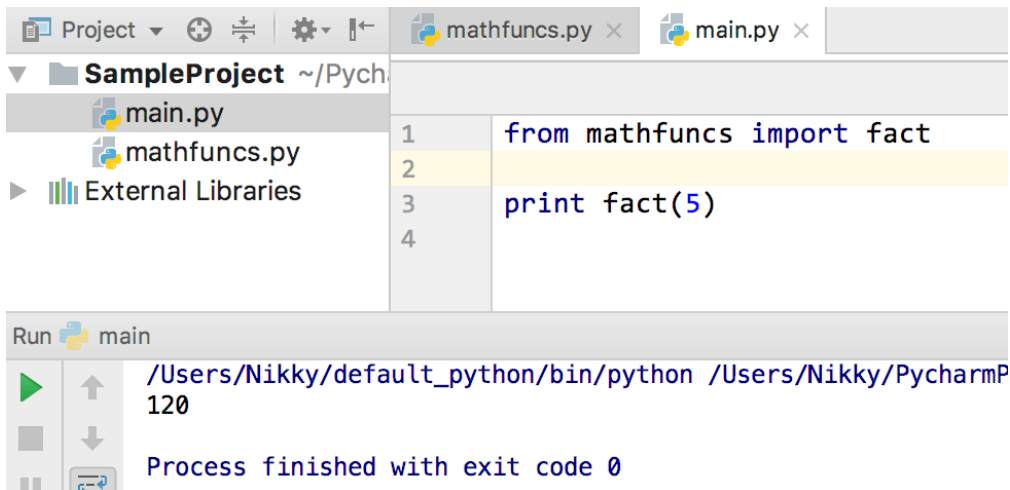
If we want to reuse the function ‘fact’, in any other python file, we have to import the file as module, using import statement.



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



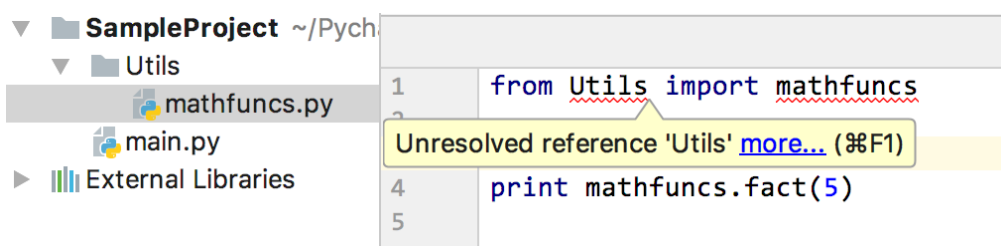
Another way of importing. 'from' keyword is used to import only specific functions in the module 'mathfuncs' without using module name.



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.

The screenshot shows the PyCharm IDE interface. On the left, the 'Project' view shows a folder named 'SampleProject' containing a sub-folder 'Utils'. Inside 'Utils', there is an '.__init__.py' file (highlighted in blue), 'mathfuncs.py', and 'main.py'. The 'Utils' folder is now a package. The main editor shows the code in 'main.py':

```
1 from Utils import mathfuncs
2
3 print mathfuncs.fact(5)
4
```

Below the editor, the 'Run' window shows the command executed: `/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProject` and the output: `120`. The status bar indicates 'Process finished with exit code 0'.

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,

The screenshot shows the PyCharm IDE interface with a more complex project structure. The 'Project' view shows 'SampleProject' containing 'Utils', which contains 'Shapes'. 'Shapes' contains '.__init__.py', 'cube.py', 'triangle.py', and 'mathfuncs.py'. The main editor shows the code in 'triangle.py':

```
1 # area of a triangle when 3 sides given
2 def area(a, b, c):
3     s = (a + b + c)/2.0
4     return (s*(s-1)*(s-b)*(s-c)) ** 0.5
5
```

The 'Run' window shows the command executed: `/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProjects/SampleProject/main.py` and the output: `factorial= 120`, `area of a triangle= 7.74596669241`, and `volume of a cube= 24`. The status bar indicates 'Process finished with exit code 0'.

`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`

The screenshot shows the PyCharm IDE interface. The 'Project' view shows the same structure as before. The main editor shows the code in 'main.py':

```
1 from Utils.Shapes.cube import volume
2 from Utils.Shapes.triangle import area
3 from Utils.mathfuncs import fact
4
5 print 'factorial=', fact(5)
6 print 'area of a triangle=', area(3, 4, 5)
7 print 'volume of a cube=', volume(2, 3, 4)
8
9
10
```

The 'Run' window shows the command executed: `/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProjects/SampleProject/main.py` and the output: `factorial= 120`, `area of a triangle= 7.74596669241`, and `volume of a cube= 24`. The status bar indicates 'Process finished with exit code 0'.

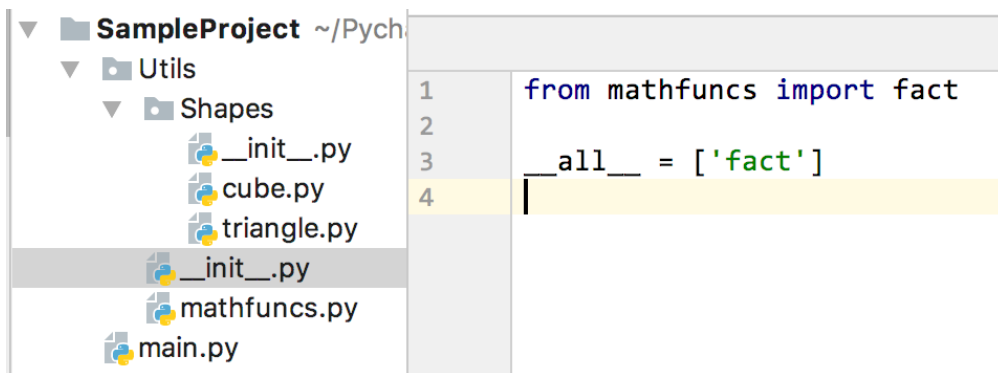
we have to use the long path name. Some developers do not want to expose the intermediate names, like Shapes, cube, triangle etc. What if, we could access all the functions directly from Utils namespace. If we export fact() to Utils name space we can directly access fact from Utils as below.

```

1  from Utils.Shapes.cube import volume
2  from Utils.Shapes.triangle import area
3  # from Utils.mathfuncs import fact
4  from Utils import fact
5
6  print 'factorial=', fact(5)
7  print 'area of a triangle=', area(3, 4, 5)
8  print 'volume of a cube=', volume(2, 3, 4)
9

```

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

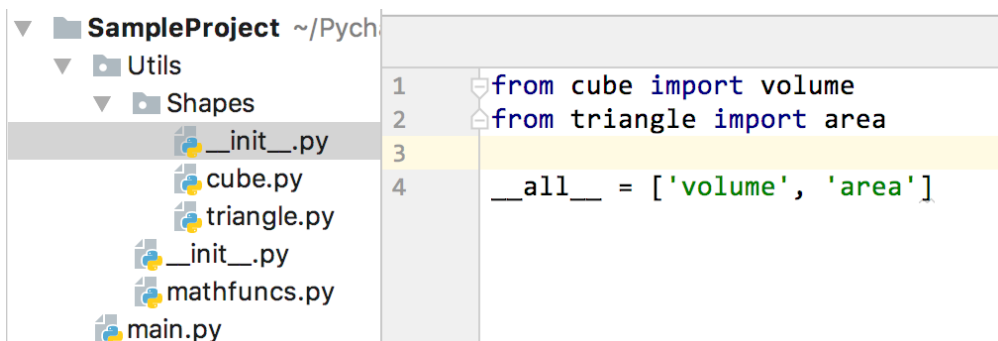


```

1  from mathfuncs import fact
2
3  __all__ = ['fact']
4

```

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**.

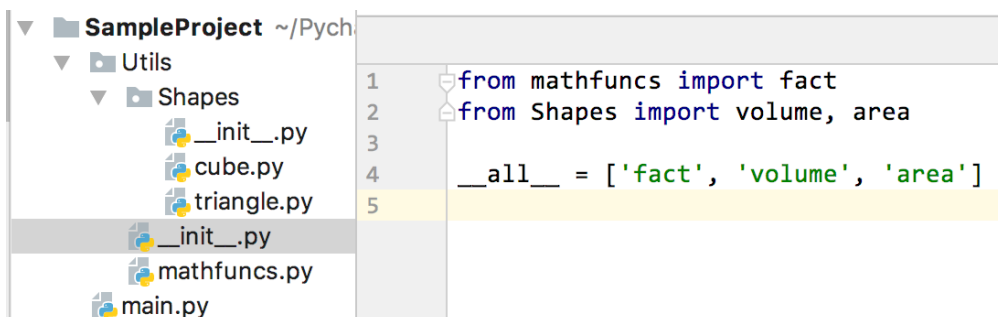


```

1  from cube import volume
2  from triangle import area
3
4  __all__ = ['volume', 'area']

```

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

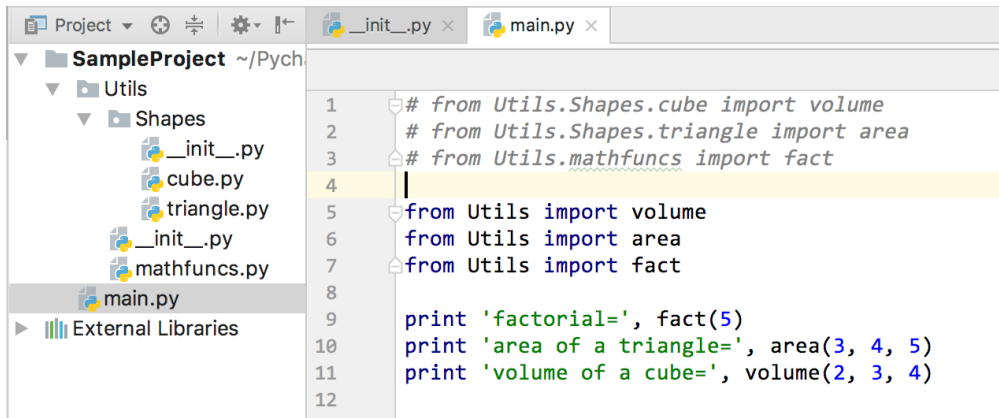


```

1  from mathfuncs import fact
2  from Shapes import volume, area
3
4  __all__ = ['fact', 'volume', 'area']
5

```

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.



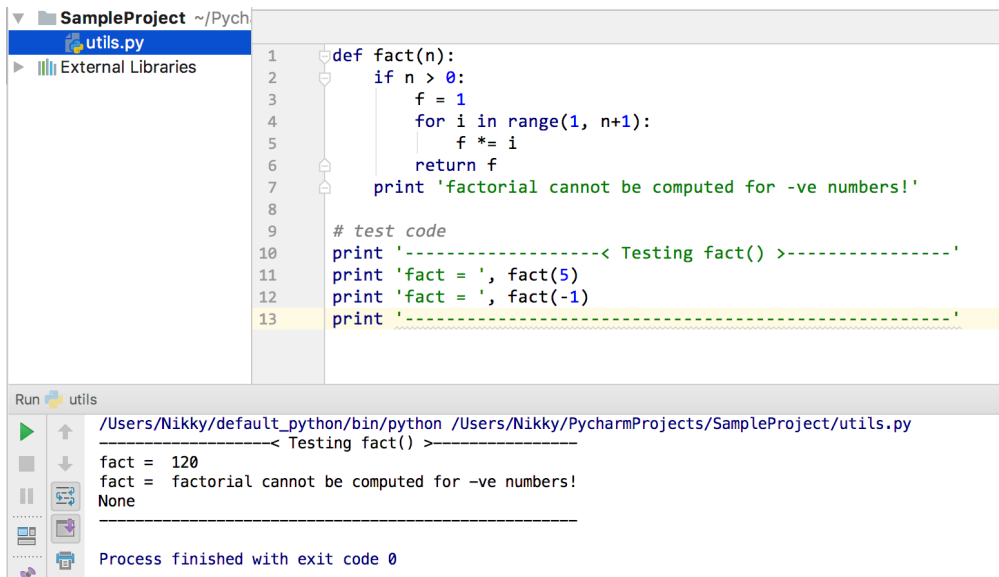
```

1  # from Utils.Shapes.cube import volume
2  # from Utils.Shapes.triangle import area
3  # from Utils.mathfuncs import fact
4
5  from Utils import volume
6  from Utils import area
7  from Utils import fact
8
9  print 'factorial=', fact(5)
10 print 'area of a triangle=', area(3, 4, 5)
11 print 'volume of a cube=', volume(2, 3, 4)
12

```

3. Preventing execution of unwanted code

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



```

1  def fact(n):
2      if n > 0:
3          f = 1
4          for i in range(1, n+1):
5              f *= i
6          return f
7      print 'factorial cannot be computed for -ve numbers!'
8
9  # test code
10 print '<-----< Testing fact() >----->'
11 print 'fact = ', fact(5)
12 print 'fact = ', fact(-1)
13 print '<----->'

```

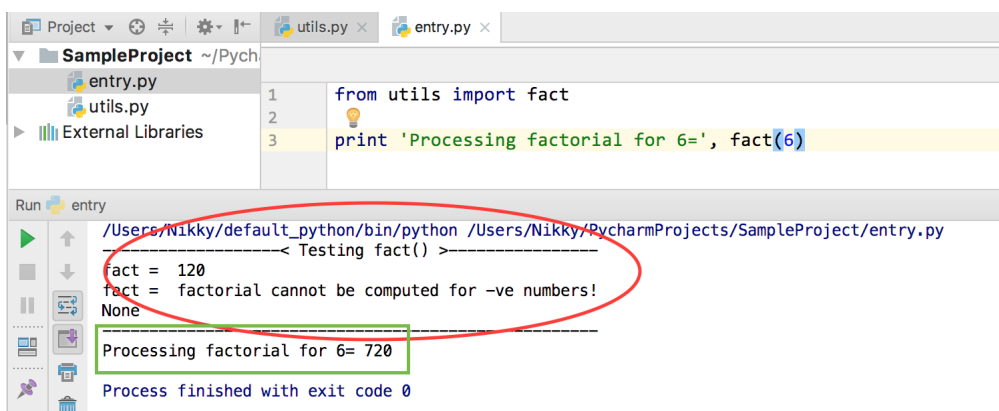
Run utils

```

/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProjects/SampleProject/utils.py
<-----< Testing fact() >----->
fact = 120
fact = factorial cannot be computed for -ve numbers!
None
Process finished with exit code 0

```

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.



```

1  from utils import fact
2
3  print 'Processing factorial for 6=', fact(6)

```

Run entry

```

/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProjects/SampleProject/entry.py
<-----< Testing fact() >----->
fact = 120
fact = factorial cannot be computed for -ve numbers!
None
Processing factorial for 6= 720
Process finished with exit code 0

```

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.

```

# entry.py
1 from utils import fact
2
3 print 'Processing factorial for 6=', fact(6)

# utils.py
fact()
1 def fact(n):
2     if n > 0:
3         f = 1
4         for i in range(1, n+1):
5             f *= i
6         return f
7     print 'factorial cannot be computed for -ve numbers!'
8
9 if __name__ == '__main__':
10     # test code
11     print '-----< Testing fact() >-----'
12     print 'fact = ', fact(5)
13     print 'fact = ', fact(-1)
14     print '-----'

Run entry
/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProjects/SampleProject/entry.py
Processing factorial for 6= 720
Process finished with exit code 0

```

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.

```

# entry.py
1 if __name__ == '__main__':
2     from utils import fact
3     print '__name__ in entry.py: ', __name__
4
5     if __name__ == '__main__':
6         print 'Processing factorial for 6=', fact(6)

# utils.py
1
2 print '__name__ in utils.py: ', __name__
3
4

Run entry
/Users/Nikky/default_python/bin/python /Users/Nikky/PycharmProjects/SampleProject/entry.py
__name__ in utils.py: utils
__name__ in utils.py: __main__
Processing factorial for 6= 720
Process finished with exit code 0

```

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:

```

file1.py
1 from file2 import dodo
2
3 def foo():
4     print "I'm foo using dodo()"
5     dodo()
6
7 def bar():
8     print "I'm bar"
9
10 if __name__ == '__main__':
11     foo()

file2.py
1 from file1 import bar
2
3 def toto():
4     print "I'm toto using bar()"
5     bar()
6
7 def dodo():
8     print "I'm dodo"

```

```

Run file1
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/munna/PycharmProjects/sample_pro/file1.py
Traceback (most recent call last):
  File "/Users/munna/PycharmProjects/sample_pro/file1.py", line 1, in <module>
    from file2 import dodo
  File "/Users/munna/PycharmProjects/sample_pro/file2.py", line 1, in <module>
    from file1 import bar
  File "/Users/munna/PycharmProjects/sample_pro/file1.py", line 1, in <module>
    from file2 import dodo
  File "/Users/munna/PycharmProjects/sample_pro/file1.py", line 1, in <module>
    from file2 import dodo
ImportError: cannot import name dodo
Process finished with exit code 1

```

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.

```

file1.py
1
2
3 def foo():
4     from file2 import dodo
5     print "I'm foo using dodo()"
6     dodo()
7
8 def bar():
9     print "I'm bar"
10
11 if __name__ == '__main__':
12     foo()

file2.py
1
2
3 def toto():
4     from file1 import bar
5     print "I'm toto using bar()"
6     bar()
7
8 def dodo():
9     print "I'm dodo"

```

```

Run file1
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/munna/PycharmProjects/sample_pro/file1.py
I'm foo using dodo()
I'm dodo
Process finished with exit code 0

```

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

```

sample_pro > file2.py
file1.py
1
2
3 def __foo():
4     print "I'm foo"
5
6
7 def bar():
8     print "I'm bar"
9

file2.py
1 from file1 import *
2
3
4 print bar()
5 print __foo()

```

```

Run file2
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/munna/PycharmProjects/sample_pro/file2.py
I'm bar
Traceback (most recent call last):
  File "/Users/munna/PycharmProjects/sample_pro/file2.py", line 5, in <module>
    print __foo()
NameError: name '__foo' is not defined
Process finished with exit code 1

```

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?