Modules and Packages

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Python

Modules

- Allows to logically organize Python code.
- Grouping related code into a module makes the code easier to understand and use.
- A module is a Python object with arbitrarily named attributes that can bind and reference.
- 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.

Modules (Cont.)

- The Python code for a module named aname normally resides in a file namedaname.py.
- Here is an example of a simple module, **support.py** :

support | demo

```
1 def print_func (par):
2     print("Hello : ", par)
3
```

The import Statement

- Can use any Python source file as a module by executing an **`import`** statement in some other Python source file.
- The **`import`** has the following syntax :

```
import module1[, module2[, ….module]
```

The import Statement (Cont.)

- When the interpreter encounters an `import` statement, it imports the module if the module is present in the search path.
- A search path is a list of directories that the interpreter searches before importing a module.
- A module is loaded only once, regardless of the number of times it is imported.
- This prevents the module execution from happening repeatedly, if multiple imports occur.

The import Statement (Cont.)

- For example, to import the module **support.py**, need to put the following command at the top of the script :

support | demo

```
1 # Import module support
2
3 import support
4
5 # Now you can call defined function that module as follow
6 support.print_func('Zara')
7
```

The from… import Statement

- Python's `from` statement lets import specific attributes from a module into the current namespace.
- The `from...import` has the following syntax :

```
from modname import name1[, name2[, ... nameN]]
```

The from… import Statement (Cont.)

support | demo

```
1  def fib(n):
2      result = []
3      a, b = 0, 1
4      while b < n :
5          result.append(b)
6          a, b = b, a + b
7      return result
8
```

support | demo

```
1 # Import module support
2
3 from support import fib
4
5 print (fib(100))
6
```

The from...import * Statement

- It is also possible to import all the names from a module into the current namespace by using the following `import` statement :

```
from modname import *
```

- This provides an easy way to import all the items from a module into the current namespace.
- However, this statement should be used sparingly.

Executing Modules as Scripts

- Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.
- The code in the module will be executed, just as if imported it, but with the `__name__` set to `__main__`.

Executing Modules as Scripts

support | demo

```
1  def fib(n):
2      result = []
3      a, b = 0, 1
4      while b < n :
5          result.append(b)
6          a, b = b, a + b
7      return result
8
9  if __name__ == "__main__" :
10     f = fib(100)
11     print (f)
12
```

Locating Modules

- When import a module, the Python interpreter searches for the module in the following sequences :
  - The current directory.
  - If the module is not found, Python then searches each directory in the shell variable ***PYTHONPATH***.
  - If all else fails, Python checks the default path.
  - On UNIX, this default path is normally **`/usr/local/lib/python3/`**.

Locating Modules (Cont.)

- The module search path is stored in the system module `sys` as the `sys.path` variable.
- The `sys.path` variable contains the current directory, *PYTHONPATH*, and the installation-dependent default.

The PYTHONPATH Variable

- The ***PYTHONPATH*** is an environment variable, consisting of a list of directories.
- The syntax of ***PYTHONPATH*** is the same as that of the shell variable ***PATH***.
- Here is a typical ***PYTHONPATH*** from a Windows system :

```
set PYTHONPATH = c:\python36\lib;
```

- And here is a typical ***PYTHONPATH*** from a UNIX system :

```
set PYTHONPATH = /usr/local/lib/python
```

The dir( ) Function

- The `dir()` built-in function returns a sorted list of strings containing the names defined by a module.
- The list contains the names of all the modules, variables and functions that are defined in a module.
- The special string variable `__name__` is the module's name, and `__file__` is the filename from which the module was loaded.

The dir( ) Function (Cont.)

```
1 # Import built-in module math
2
3 import math
4
5 content = dir(math)
6
7 print (content)
```

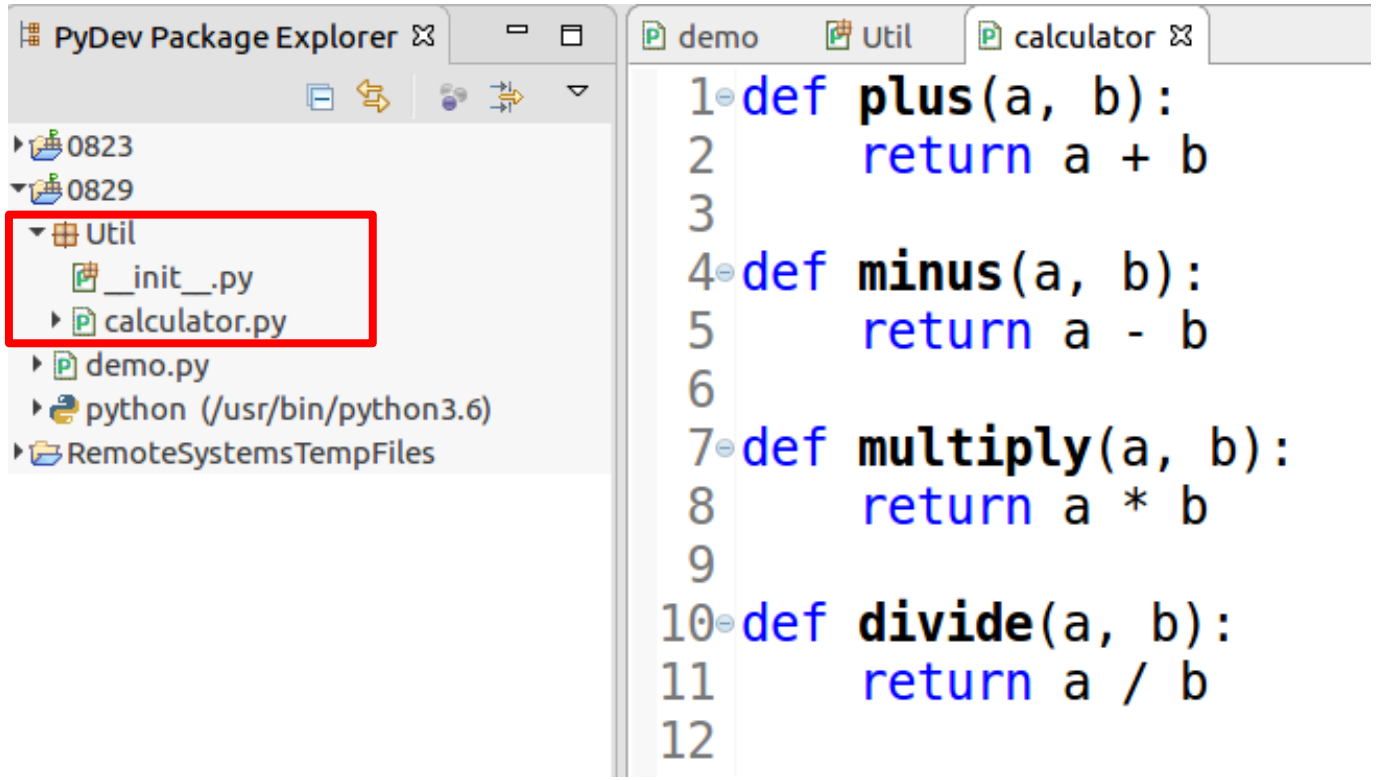
Console

<terminated> demo.py [/usr/bin/python3.6]

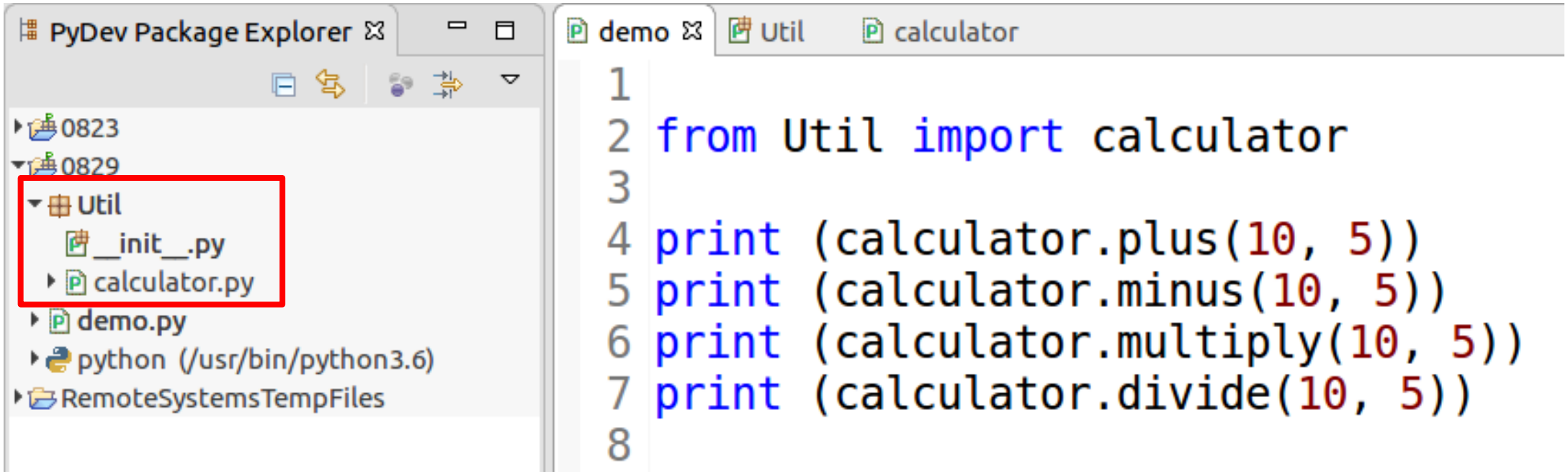
```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
```

Packages

- 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.



Packages


PyDev Package Explorer
0823
0829
Util
__init__.py
calculator.py
demo.py
python (/usr/bin/python3.6)
RemoteSystemsTempFiles
demo
Util
calculator
1
2 from Util import calculator
3
4 print (calculator.plus(10, 5))
5 print (calculator.minus(10, 5))
6 print (calculator.multiply(10, 5))
7 print (calculator.divide(10, 5))
8