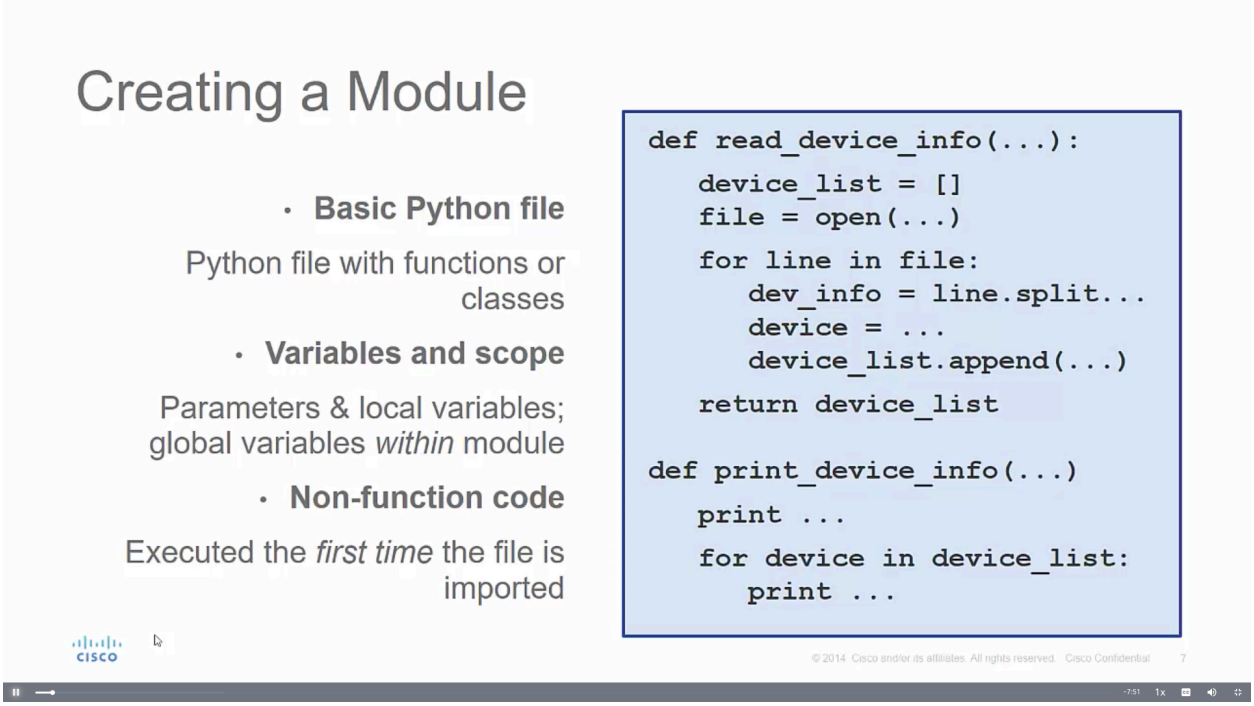


Python modules are simply files with Python code in them. The difference between a module and an application file is that the module file is intended to be called by the application. As such, the module file will contain functions and/or classes, typically related to each other in some way. For example, a module for utility routines, a module for a specific type of Python classes, a module for commonly used classes, and so on.

Module Data

When creating a Python module, especially with functions, it is important to consider carefully the variables you will be using in your functions. These variables will fall into three categories:

- **Parameters:** Parameters are the values passed to your function by the calling routine. Your formal parameters, which are enclosed in the parentheses in your function definition, are accessible to you and valid within that function only.
- **Local variables:** Local variables are those variables and data structures you define within your function. They are not seen by the calling routine, or by any other functions; they are local to your function alone.

Note

If you create a local variable, such as, a device list, and then you return that value to the caller using the return statement, then your local variable will be, in effect, passed back to the caller, and the calling function now has a reference to that data.

- **Global variables:** It is possible – though not necessarily a good idea – to define global variables for your module. These global variables are accessible and seen only by code within the module – calling code cannot see these variables. The scope of these variables is for the entire module.

If you create a local variable, such as a device list, and then you return that value to the caller using the return statement, then your local variable will be, in effect, passed back to the caller, and the calling function now has a reference to that data.

Classes have similar categories of data as in a module of functions, but since classes tend to have self-contained instance data, it can be less confusing. You can still have global data, however, which is accessible to all classes in the module.

Module Code

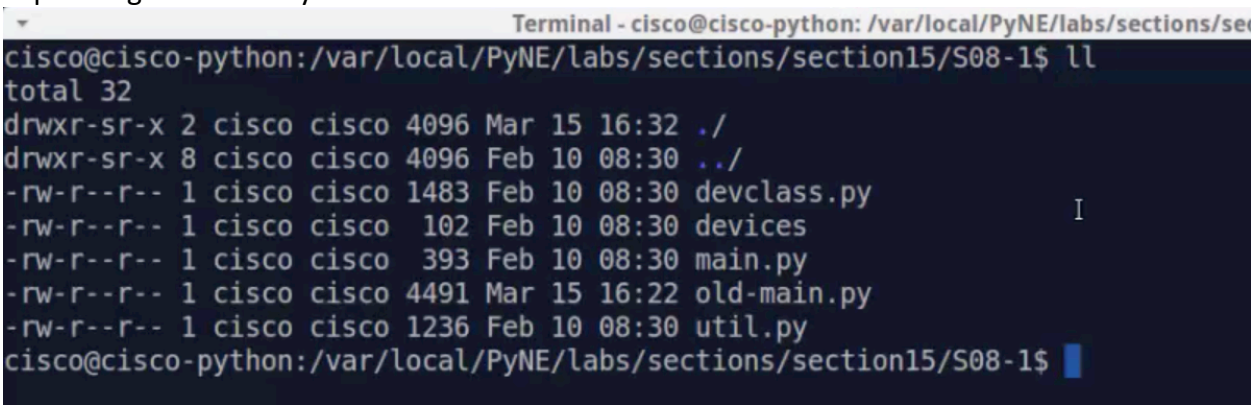
The main purpose of a module is to hold functions or classes that can be called by other pieces of code, either a main application or by code in another module which itself has been called by the main application. However, it is possible to define code in your modules which does not belong to any function or class – in other words, the code is not within any of your function or class definitions. This code can be considered global to the module; it is executed once, and only once, when the module is first imported by the application or by any other module.

If the purpose of your module is to hold some data, in addition to providing functions to be called, then having this type of module code might be useful to initialize that global data for the module. Using a module in this manner with a module containing only functions makes that module similar to a Python class, providing both data (like attributes) and functions (like methods). The major difference lies in the fact that for classes you will create separate instances, with their own instance data; this is not true for modules. The bulk of your code in a module will be located in your functions, but there may be some cases where non-function code may serve a useful purpose.

Modules with Classes

For modules with classes, the data is structured into class attributes, and the same rules apply within a class that is defined in a module, as with classes defined in the main application file itself. For modules of classes, you can have global variables as well, but object-oriented programming is intended to limit or remove the use of global variables, and should be avoided in modules as well.

Separating Code into Python Directories:



The old-main.py has become too large and impractical to work with.

We separated the base devclass into its own module:

```
import pexpect

#================================================================
#---- Class to hold information about a generic network device --------
class NetworkDevice():

    def __init__(self, name, ip, user='cisco', pw='cisco'):
        self.name = name
        self.ip_address = ip
        self.username = user
        self.password = pw

    def connect(self):
        self.session = None

    def get_interfaces(self):
        self.interfaces = '--- Base Device, does not know how to get interfaces ---'
```

```
#=============================================================
#---- Class to hold information about an IOS network device --------
class NetworkDeviceIOS(NetworkDevice):

    def __init__(self, name, ip, user='cisco', pw='cisco'):
        NetworkDevice.__init__(self, name, ip, user, pw)

    def connect(self):
        print '--- connecting IOS: telnet '+self.ip_address

        self.session = pexpect.spawn('telnet '+self.ip_address, timeout=20)
        result = self.session.expect(['Username:', pexpect.TIMEOUT])

        self.session.sendline(self.username)
        result = self.session.expect('Password:')

        # Successfully got password prompt, logging in with password
        self.session.sendline(self.password)
        self.session.expect('>')

    def get_interfaces(self):

        self.session.sendline('show interfaces summary')
        result = self.session.expect('>')

        self.interfaces = self.session.before
```

And also the Utility module:

```
import devclass

#=====================================================================
def read_devices_info(devices_file):

    devices_list = []

    file = open(devices_file,'r')
    for line in file:

        device_info = line.strip().split(',')

        # Create a device object with this data
        if device_info[1] == 'ios':

            device = devclass.NetworkDeviceIOS(device_info[0],device_info[2],
                                               device_info[3],device_info[4])

        else:
            device = devclass.NetworkDevice(device_info[0],device_info[2],
                                            device_info[3],device_info[4])

        devices_list.append(device)

    return devices_list


#=====================================================================
def print_device_info(device):

    print '------------------------------------------------------'
    print '    Device Name:      ',device.name
    print '    Device IP:        ',device.ip_address
    print '    Device username:  ',device.username,
    print '    Device password:  ',device.password

    print ''
    print '    Interfaces'
    print ''

    print device.interfaces
    print '------------------------------------------------------\n\n'
```

Finally the “main.py” or new python file that references the utilities module:

```
import util

#=====================================================================
# Main program: connect to device, show interface, display

devices_list = util.read_devices_info('devices')

for device in devices_list:

    print '==== Device ================================================================='

    device.connect()
    device.get_interfaces()
    util.print_device_info(device)
```