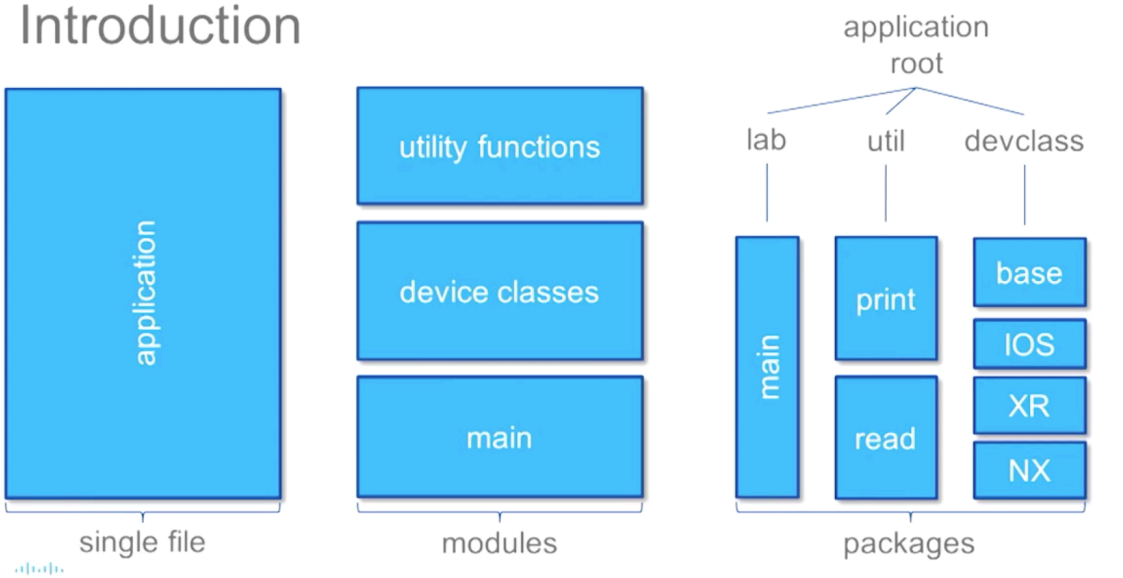
Introduction



Modules

- **Module files**

Normal Python files, typically with only function or class definitions

- **Calling files**

Callers reference Python modules using 'import' statement

util.py (module)

```
def read_device_info(...):
def print_device_info(...):
```

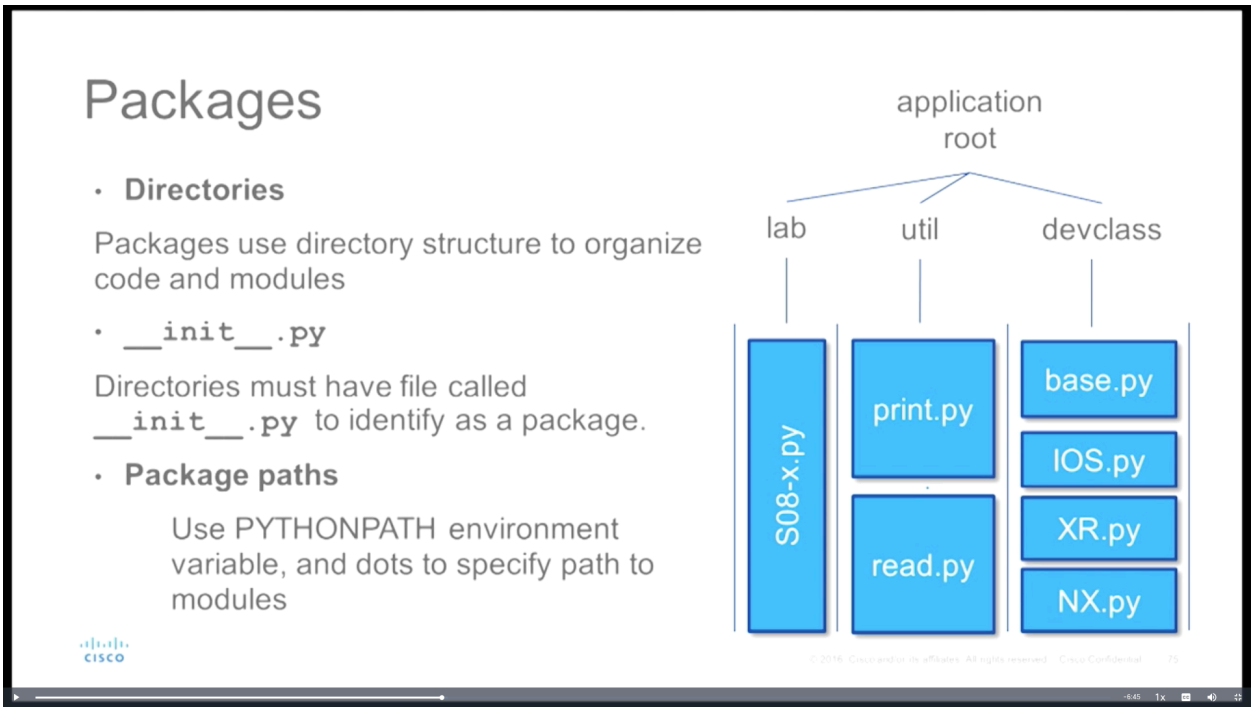
devclass.py (module)

```
class NetworkDevice() :
class IOSDevice(...) :
```

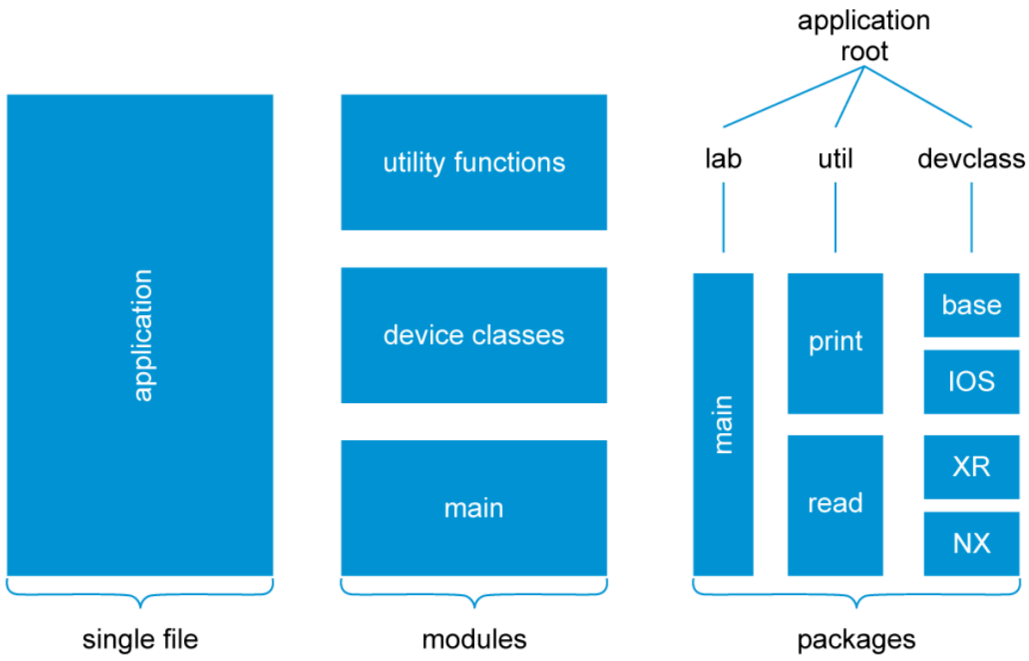
lab08-1 (main)

```
import util
import devclass

util.read_device_info(...)
```



Your network programmability code will eventually grow to the point where having single files holding your scripts and applications no longer makes sense. At this point, you will want to organize your code into Python modules and packages.



The image shows the three different ways in which Python applications can be structured.

Single Application File

On the left is a single application file. This format has the advantage of having all code located in the same file for easy access. However once your application becomes larger, the single file becomes crowded and can be confusing, hard to modify and even harder to maintain. Therefore, once your application begins to grow, you will want to separate the different pieces of functionality into modules.

Modules

A Python module is technically just a file with Python code in it. Your single application is technically a 'module'. However, when people speak of separating an application into modules, the idea is to take common pieces of functionality and organizing them together into their own, separate Python files. These files are then called Python modules, which are a collection of Python functions or classes, which are grouped into a single file.

Your application will reference these modules, and the functions or classes within them, using the Python import statement.

Packages

A Python package is a directory that contains Python modules. When your application becomes large enough to require more organization than can be provided by modules alone, you will want to break it apart into packages.

A directory becomes a Python package by having a special __init__.py file located within it. Every directory that is intended to be a package must have an __init__.py file in it. Your application will reference the modules within your packages in the normal way, using the Python import statement. Since you are running code from different directory, you will need to make sure that Python knows where to look for your packages, using the PYTHONPATH environment variable.

Modules and packages are important not only for organizing your own application, but for creating code that can be imported and called by other developers as well. The Python libraries that you have been using are all Python modules, hosting functions or classes that you will be using regularly in your network programming projects. So the value of modules and packages may go well beyond your own specific application.

Note: The remainder of these lessons will be referring to functions inside Python modules. It is also possible to define classes within modules – in fact, this is done in the examples. For purpose of clarity, this section will refer to functions, since that is the most common and simplest type of Python object to define in a module. If you read 'function', you should assume it to include classes as well.

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section15/S08-1$ cat devices
ios-01,ios,10.30.30.1,cisco,cisco
ios-02,ios,10.30.30.2,cisco,cisco
ios-03,ios,10.30.30.3,cisco,cisco
cisco@cisco-python:/var/local/PyNE/labs/sections/section15/S08-1$
```

Device Class 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
```

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'
```

Main 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)
~
```