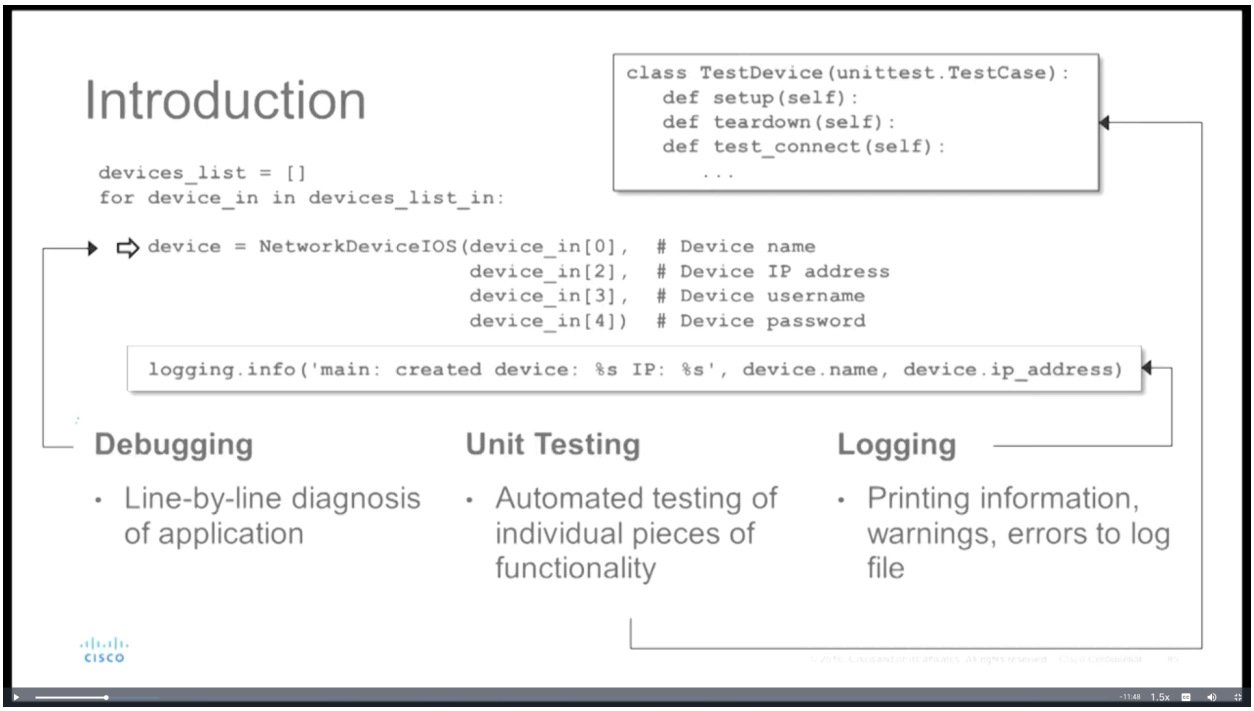

Regression Testing – Running unit tests against code to ensure that as newer code is added, it does not conflict with older code

When you have written your Python application, you will run it in order to determine if it is working as intended. Determining whether it is running correctly may involve merely looking at the output to see if the results are what was expected. Sometimes more thorough testing is required and this section will look into some mechanisms for taking testing to a deeper level. There are three important aspects of testing which will be considered:

- **Debugging:** Debugging an application means stopping execution in the middle of the application, and looking at the internal variables to see if they are as they should be. It can mean going through the code step-by-step in order to determine if it is operating as intended. It can mean tracing the thread of execution down into function calls in order to determine where and when a failure has occurred.
- **Unit testing:** Running your application and examining results is one way to determine a superficial level of quality. But a better way to test the completeness and quality of your application is via unit testing. Unit testing frameworks allow you to test the inputs and outputs of every facet of your code. Unit testing also provides a framework for creating automated tests through which you can insure quality across versions by performing *regressions* to make sure nothing that previously worked has been broken by the new code changes.
- **Logging:** When your application working in a live environment, you will not have the luxury of printing diagnostic messages to the console, because you typically will not be sitting at the console to observe them. Logging provides a mechanism for recording notable events (informational messages), potential issues (warning messages), problems (error messages), and failures (critical messages). These messages go into log files, which can later be examined to determine any issues that may be arising in your application's execution.

The chronology of these testing mechanisms generally follows the order in which they are presented here:

- **Development Phase:** Debugging is done during development, when you can open up your code to determine the internal defects that cause your program to misbehave.
- **Testing Phase:** Unit testing is built as a framework around your application, in part to insure that operational code does not get broken on subsequent builds and revisions of your application. Note that in software development, the idea of 'Test Driven Development' (TDD) has gained some popularity, wherein unit tests are written *first*, and the actual code to implement the feature or function is implemented in response to the unit tests that have been created. The unit tests that are described in this section can be used for TDD and for the more traditional develop-then-test model of creating software.
- **Production Phase:** Logging is put into your application for use primarily when it has been completed and is operating in 'production', meaning in a real-world environment. At this phase, it is difficult to go into debug mode, and writing and running tests is often not possible. Logging is the best way to understand the problem, in order to address it in some manner.

```
import logging

from util import read_devices_info
from devclass import NetworkDeviceIOS

#=====================================================================
devices_filename = 'csv-devices'

logging.basicConfig(filename='prne.log',
                    format='%(asctime)s %(message)s',
                    level=logging.INFO)

devices_list_in = read_devices_info(devices_filename)  # read CSV info for all devices

logging.info('main: read %s devices from %s', len(devices_list_in), devices_filename)

# Iterate through all devices from the file, creating device objects for each
devices_list = []
for device_in in devices_list_in:
```

```
    device = NetworkDeviceIOS(device_in[0],  # Device name
                              device_in[2],  # Device IP address
                              device_in[3],  # Device username
                              device_in[4])  # Device password

    logging.info('main: created device: %s IP: %s', device.name, device.ip_address)
    print '----- device: name: ',device.name,' IP: ',device.ip_address

    devices_list.append(device)

# Iterate through all devices, connecting and getting interface and routing info
for device in devices_list:

    session = device.connect()
    if session == 0:
        logging.error('main: unable to connect: %s, %s', device.name, device.ip_address)
        continue

    device.set_terminal_length()
    interfaces = device.get_interfaces()
    if interfaces == 0 or len(interfaces) == 0:
        logging.error('main: get interfaces failed')
        continue

    print '--------------- device: name: ',device.name,' IP: ',device.ip_address
    print 'interfaces:', interfaces

    logging.info('main: got interfaces data for: %s', device.name)

    routes = device.get_routes()
    if routes == 0 or len(routes) == 0:
        logging.error('main: get routes failed')
        continue

    print '--------------- device: name: ',device.name,' IP: ',device.ip_address
    print 'routes:', routes

    logging.info('main: got routes data for: %s', device.name)
```

Util.py

```
import csv
from pprint import pprint

from devclass import NetworkDevice
from devclass import NetworkDeviceIOS

import logging

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

    devices_list = []

    logging.info('util: reading device info from file: %s', devices_file)

    file = open(devices_file,'r')   # Open the CSV file
    csv_devices = csv.reader(file)  # Create the CSV reader for file

    # Use list comprehension to put CSV data into list of lists
    return [dev_info for dev_info in csv_devices]

#=============================================================================
#=============================================================================
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 '----------------------------------------------------'

#=============================================================================
def write_devices_info(devices_file, devices_out_list):

    logging.info('util: writing device info to file: %s', devices_file)

    # Use CSV library to output our list of lists to a CSV file
    with open(devices_file, 'w') as file:
        csv_out = csv.writer(file)
        csv_out.writerows(devices_out_list)
```

Devclass.py

```
import pexpect
import logging

#---- 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)
        logging.info('devclass: created IOS device: %s %s', name, ip)

    #-----------------------------------------------------------------------------
    def connect(self):
        print '--- connecting IOS: telnet: '+self.username+'/'+self.password

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

        # Check for failure
        if result != 0:
            logging.warn('devclass: --- Timout or unexpected reply from device')
            return 0

        # Successfully got username prompt, logging in with password
        self.session.sendline(self.username)
        result = self.session.expect(['Password:', pexpect.TIMEOUT])

        # Check for username failure
        if result != 0:
            logging.warn('devclass: --- Timeout or unexpected username reply from device')
            return 0

        # Successfully got password prompt, finish log in with password
        self.session.sendline(self.password)
        result = self.session.expect(['>', 'invalid', pexpect.TIMEOUT])

        # Check for password failure
        if result != 0:
            logging.warn('devclass: --- Timeout or unexpected password reply from device')
            return 0

        logging.info('devclass: successful login at: %s for user: %s',
                     self.ip_address,self.username)

        return self.session
```

```
    #-------------------------------------------------------------------------
    def set_terminal_length(self):

        # Must set terminal length to zero for long replies
        self.session.sendline('terminal length 0')
        result = self.session.expect(['>', pexpect.TIMEOUT])

        if result != 0:
            logging.warn('devclass: --- Timeout or unexpected terminal length reply')
            return False

        else: return True

#-------------------------------------------------------------------------
def get_interfaces(self):

    self.session.sendline('show interfaces summary')
    result = self.session.expect(['>', pexpect.TIMEOUT])

    print 'show interfaces summary result: ',result

    if result != 0:
        logging.warn('--- Timeout or unexpected reply from show interfaces')
        return 0

    self.interfaces = self.session.before
    return self.interfaces

#-------------------------------------------------------------------------
def get_routes(self):

    self.session.sendline('show ip route')
    result = self.session.expect(['>', pexpect.TIMEOUT])

    print 'show interfaces summary result: ',result

    if result != 0:
        logging.warn('--- Timeout or unexpected reply from show interfaces')
        return 0

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

Run main.py then check if log file created:

```
-rw-rw-r-- 1 cisco cisco 1161 Mar 30 10:18 prne.log
-rw-r--r-- 1 cisco cisco 1405 Feb 10 08:30 util.py
-rw-r--r-- 1 cisco cisco 1594 Mar 30 10:18 util.pyc
cisco@cisco-python:/var/local/PyNE/labs/sections/section22/S11-3-logging$ cat prne.log
```

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section22/S11-3-logging$ cat prne.log
2016-03-30 10:18:37,417 util: reading device info from file: csv-devices
2016-03-30 10:18:37,417 main: read 3 devices from csv-devices
2016-03-30 10:18:37,417 devclass: created IOS device: ios-01 10.30.30.1
2016-03-30 10:18:37,417 main: created device: ios-01 IP: 10.30.30.1
2016-03-30 10:18:37,417 devclass: created IOS device: ios-02 10.30.30.2
2016-03-30 10:18:37,417 main: created device: ios-02 IP: 10.30.30.2
2016-03-30 10:18:37,418 devclass: created IOS device: ios-03 10.30.30.3
2016-03-30 10:18:37,418 main: created device: ios-03 IP: 10.30.30.3
2016-03-30 10:18:37,956 devclass: successful login at: 10.30.30.1 for user: cisco
2016-03-30 10:18:38,176 main: got interfaces data for: ios-01
2016-03-30 10:18:38,290 main: got routes data for: ios-01
2016-03-30 10:18:38,889 devclass: successful login at: 10.30.30.2 for user: cisco
2016-03-30 10:18:39,124 main: got interfaces data for: ios-02
2016-03-30 10:18:39,256 main: got routes data for: ios-02
2016-03-30 10:18:39,851 devclass: successful login at: 10.30.30.3 for user: cisco
2016-03-30 10:18:40,200 main: got interfaces data for: ios-03
2016-03-30 10:18:40,344 main: got routes data for: ios-03
```