Basic Logging

In this exercise, you will use basic logging functionality to add logging in to your network application.

Step 1

Modify the `main.py` application to send log output to a file named `prne.log`. Set the logging format to include the `timestamp` and set the logging level to `INFO`.

Answer

Navigate to the **~/Desktop/PRNE/section16/logging** folder and open `main.py` with a text editor. Add the following lines:

```
import logging

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

Step 2

In `main.py`, log messages for notable events, such as reading in a device, connecting to a device, getting interface information for a device, and getting routing information for a device. Note that for basic logging, you will need to put the module name into the log message – the logging function will not do that for you.

Answer

The following example shows logging an informational message after reading the device info from the CSV file. You should add similar logging messages for other events.

```
devices_list_in = read_devices_info(devices_filename)  # read CSV info for d

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

Step 3

In `main.py`, log messages for error events, such as failure to connect to a device or failure to get interface information.

Answer

The following example shows logging an error message if the application fails to connect to the device. You should add similar error messages for other events.

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

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

Step 4

Run your application and verify that log messages are being logged.

Answer

`prne.log` should contain messages similar to:

```
2016-02-12 09:10:33,762 main: read 3 devices from csv-devices
2016-02-12 09:10:33,762 main: created device: ios-01 IP: 10.30.30.1
2016-02-12 09:10:33,762 main: created device: ios-02 IP: 10.30.30.2
2016-02-12 09:10:33,762 main: created device: ios-03 IP: 10.30.30.3
2016-02-12 09:10:34,537 main: got interfaces data for: ios-01
2016-02-12 09:10:34,651 main: got routes data for: ios-01
2016-02-12 09:10:35,472 main: got interfaces data for: ios-02
2016-02-12 09:10:35,585 main: got routes data for: ios-02
2016-02-12 09:10:36,395 main: got interfaces data for: ios-03
2016-02-12 09:10:36,507 main: got routes data for: ios-03
```

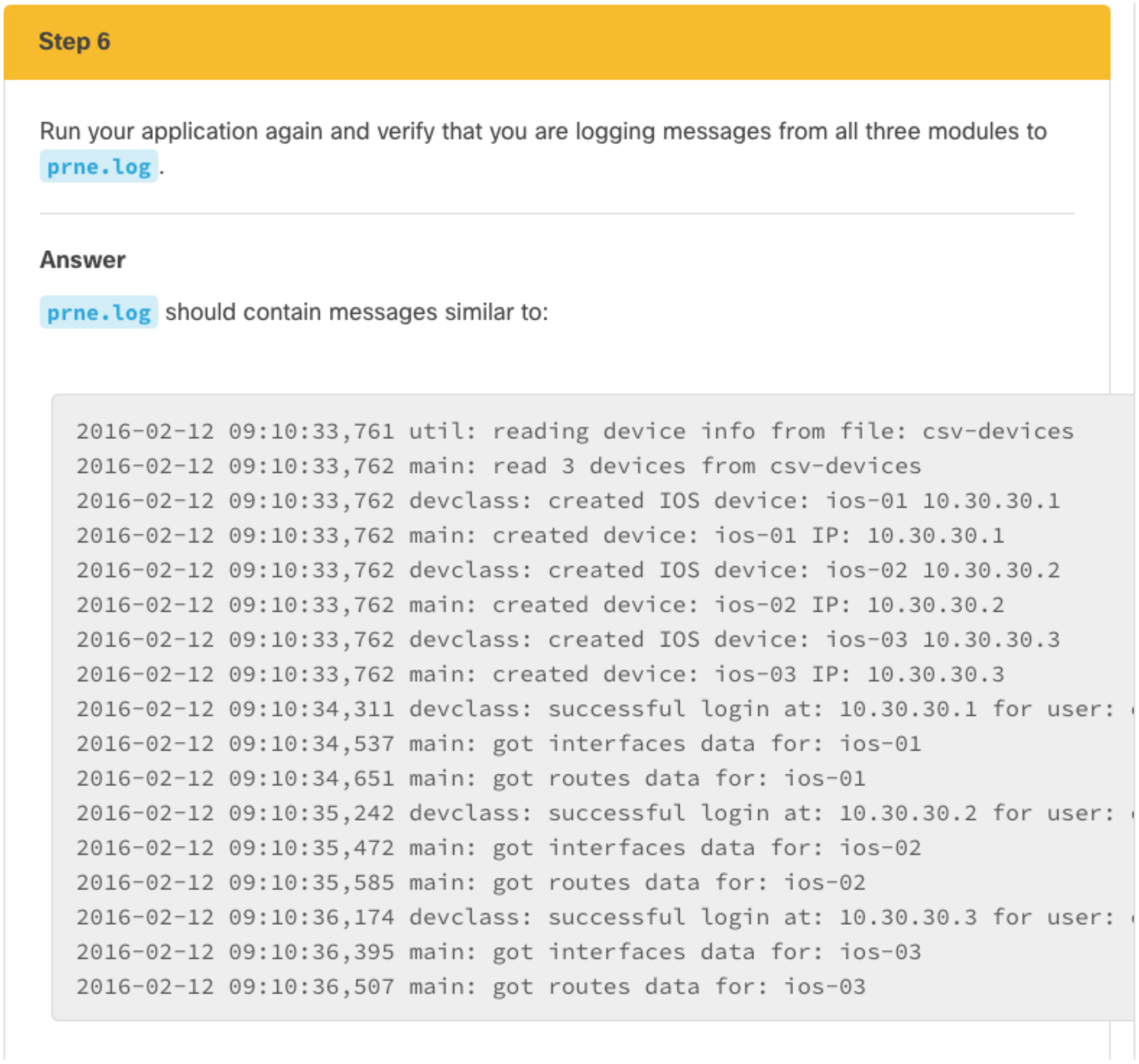
Step 5

Add logging to `util.py` and `devclass.py`, making sure that the messages get logged in to your main log file as they should.

Answer

The following example shows logging an info message when the `util.py` module reads device info. You should add similar logging messages for other events in `util.py` and in `devclass.py`.

```
def read_devices_info(devices_file):

    devices_list = []

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



Step 6

Run your application again and verify that you are logging messages from all three modules to `prne.log`.

Answer

`prne.log` should contain messages similar to:

```
2016-02-12 09:10:33,761 util: reading device info from file: csv-devices
2016-02-12 09:10:33,762 main: read 3 devices from csv-devices
2016-02-12 09:10:33,762 devclass: created IOS device: ios-01 10.30.30.1
2016-02-12 09:10:33,762 main: created device: ios-01 IP: 10.30.30.1
2016-02-12 09:10:33,762 devclass: created IOS device: ios-02 10.30.30.2
2016-02-12 09:10:33,762 main: created device: ios-02 IP: 10.30.30.2
2016-02-12 09:10:33,762 devclass: created IOS device: ios-03 10.30.30.3
2016-02-12 09:10:33,762 main: created device: ios-03 IP: 10.30.30.3
2016-02-12 09:10:34,311 devclass: successful login at: 10.30.30.1 for user:
2016-02-12 09:10:34,537 main: got interfaces data for: ios-01
2016-02-12 09:10:34,651 main: got routes data for: ios-01
2016-02-12 09:10:35,242 devclass: successful login at: 10.30.30.2 for user:
2016-02-12 09:10:35,472 main: got interfaces data for: ios-02
2016-02-12 09:10:35,585 main: got routes data for: ios-02
2016-02-12 09:10:36,174 devclass: successful login at: 10.30.30.3 for user:
2016-02-12 09:10:36,395 main: got interfaces data for: ios-03
2016-02-12 09:10:36,507 main: got routes data for: ios-03
```

Your complete application should look similar to:

main.py

```
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 d

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

# Iterate through all devices from the file, creating device objects for eac
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_addres
    print '----- device: name: ',device.name,' IP: ',device.ip_address

    devices_list.append(device)
```

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

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

#==== 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.

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

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

        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 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 interfac
            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 interfac
            return 0

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

Advanced Logging

In this exercise, you will use advanced logging functionality to add logging to your network application, using individual loggers for each module, and using a rotating log file scheme.

Step 7

Import the required logging modules into your main module.

Answer

To use advanced logging features you must import logging handlers in addition to logging.

```
import logging
import logging.handlers
```

Step 8

Create a logger in `main.py` using the `getLogger()` function. Set the logging level to INFO.

Answer

Remember that your naming convention must be consistent between your modules `main.py`, and the called modules `util.py` and `devclass.py`.

```
logger = logging.getLogger('main')
logger.setLevel(logging.INFO)
```

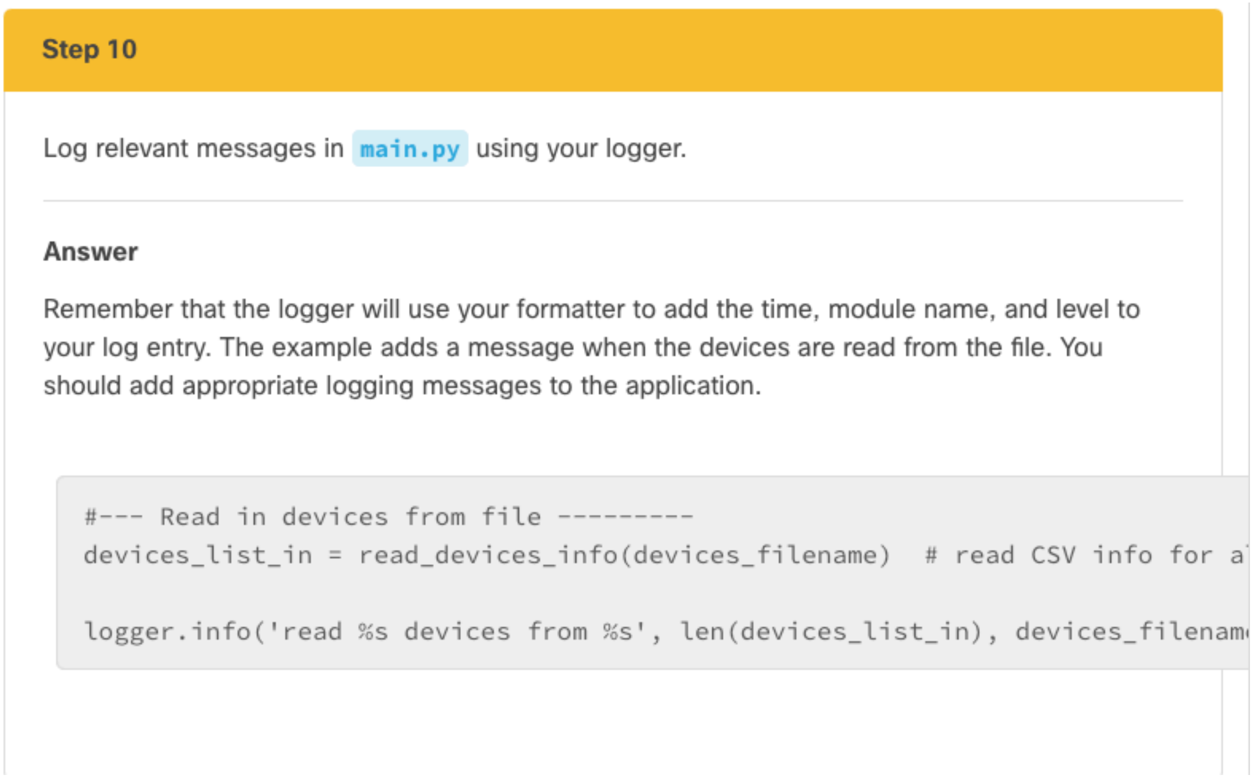
Step 9

In `main.py`, create a handler using a rotating file scheme. For the handler, create and set a formatter to log time, module name, logging level, and message.

Answer

Remember to add the handler to the logger.

```
handler = logging.handlers.RotatingFileHandler('main.log',
                                               maxBytes=20000,backupCount=4
handler.setFormatter(logging.Formatter(
                     '%(asctime)s - %(name)s - %(levelname)s - %(message)s')

logger.addHandler(handler)
```



Step 10

Log relevant messages in `main.py` using your logger.

Answer

Remember that the logger will use your formatter to add the time, module name, and level to your log entry. The example adds a message when the devices are read from the file. You should add appropriate logging messages to the application.

```
#--- Read in devices from file ---------
devices_list_in = read_devices_info(devices_filename)  # read CSV info for a

logger.info('read %s devices from %s', len(devices_list_in), devices_filenam
```

Step 11

Create loggers in `util.py` and `devclass.py`.

Answer

Remember that you only have to import logging. Handlers have been created in your parent logger which you created in `main.py`. Also remember that you must create your logger with a name that uses dot notation in order for Python to know that you are inheriting characteristics from the logger created in `main.py`.

```
util.py

import logging
logger = logging.getLogger('main.util')

devclass.py

import logging
logger = logging.getLogger('main.devclass')
```

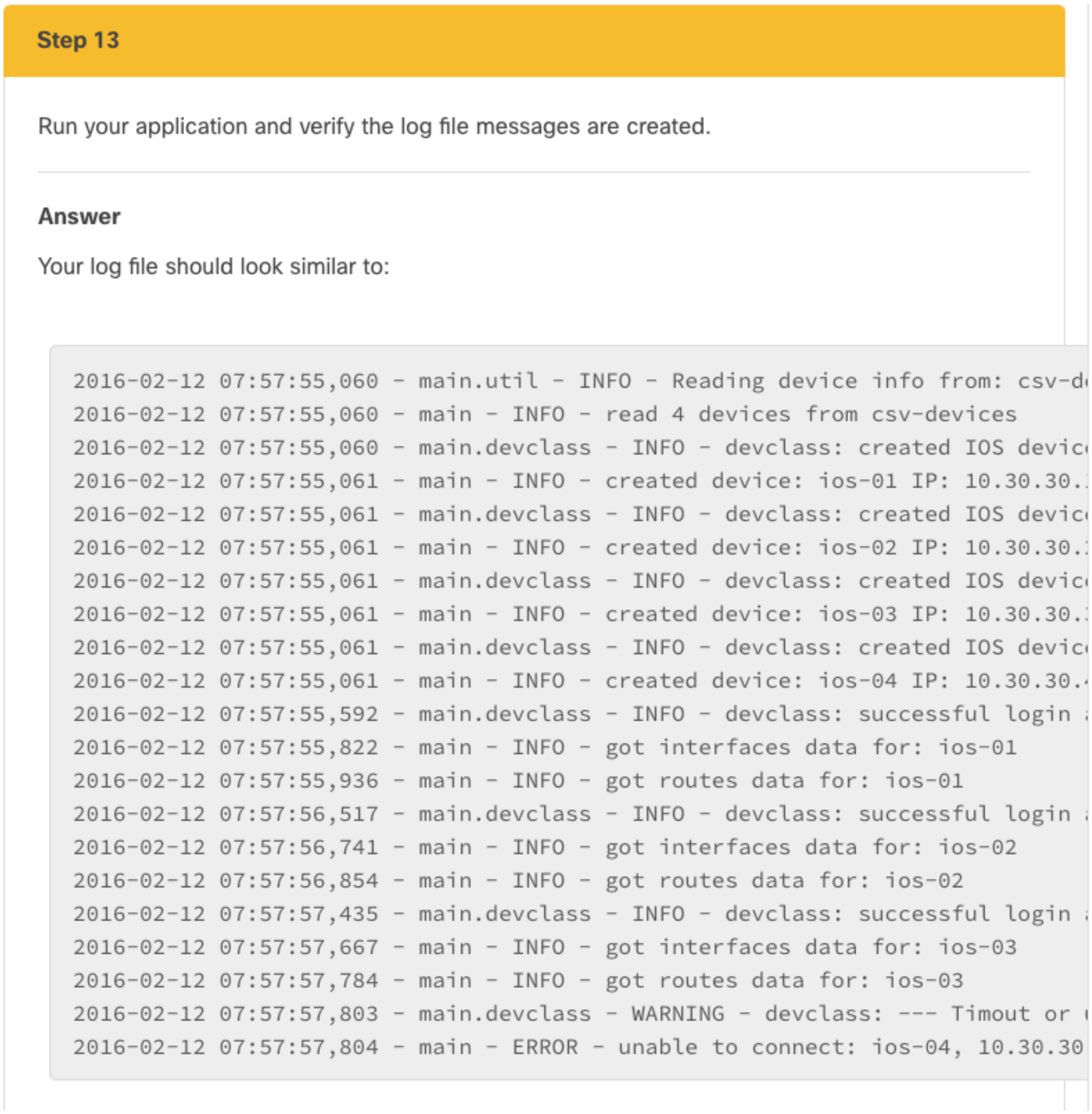
Step 12

Log relevant messages in `util.py` and `devclass.py` using your loggers for those modules.

Answer

The example shows an info message in `util.py` and a warning message in `devclass.py`. You should add appropriate logging throughout the application.

```
util.py

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

    logger.info('Reading device info from: %s', devices_file)


devclass.py

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

        # Check for failure
        if result != 0:
            logger.warn('devclass: --- Timout or unexpected reply from devic
            return 0
```



Step 13

Run your application and verify the log file messages are created.

Answer

Your log file should look similar to:

```
2016-02-12 07:57:55,060 - main.util - INFO - Reading device info from: csv-d
2016-02-12 07:57:55,060 - main - INFO - read 4 devices from csv-devices
2016-02-12 07:57:55,060 - main.devclass - INFO - devclass: created IOS devic
2016-02-12 07:57:55,061 - main - INFO - created device: ios-01 IP: 10.30.30.
2016-02-12 07:57:55,061 - main.devclass - INFO - devclass: created IOS devic
2016-02-12 07:57:55,061 - main - INFO - created device: ios-02 IP: 10.30.30.
2016-02-12 07:57:55,061 - main.devclass - INFO - devclass: created IOS devic
2016-02-12 07:57:55,061 - main - INFO - created device: ios-03 IP: 10.30.30.
2016-02-12 07:57:55,061 - main.devclass - INFO - devclass: created IOS devic
2016-02-12 07:57:55,061 - main - INFO - created device: ios-04 IP: 10.30.30.
2016-02-12 07:57:55,592 - main.devclass - INFO - devclass: successful login
2016-02-12 07:57:55,822 - main - INFO - got interfaces data for: ios-01
2016-02-12 07:57:55,936 - main - INFO - got routes data for: ios-01
2016-02-12 07:57:56,517 - main.devclass - INFO - devclass: successful login
2016-02-12 07:57:56,741 - main - INFO - got interfaces data for: ios-02
2016-02-12 07:57:56,854 - main - INFO - got routes data for: ios-02
2016-02-12 07:57:57,435 - main.devclass - INFO - devclass: successful login
2016-02-12 07:57:57,667 - main - INFO - got interfaces data for: ios-03
2016-02-12 07:57:57,784 - main - INFO - got routes data for: ios-03
2016-02-12 07:57:57,803 - main.devclass - WARNING - devclass: --- Timout or
2016-02-12 07:57:57,804 - main - ERROR - unable to connect: ios-04, 10.30.30
```

Your application should look similar to:

main.py

```
import logging
import logging.handlers

from util import read_devices_info
from devclass import NetworkDeviceIOS

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

#--- Set up logging --------------------
logger = logging.getLogger('main')
logger.setLevel(logging.INFO)

handler = logging.handlers.RotatingFileHandler('main.log',maxBytes=20000,bac
handler.setFormatter(logging.Formatter('%(asctime)s - %(name)s - %(levelname

logger.addHandler(handler)

#--- Read in devices from file ---------
devices_list_in = read_devices_info(devices_filename)  # read CSV info for a

logger.info('read %s devices from %s', len(devices_list_in), devices_filenam

# Iterate through all devices from the file, creating device objects for eac
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
    logger.info('created device: %s IP: %s', device.name, device.ip_address)

    devices_list.append(device)

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

    session = device.connect()
    if session == 0:
        logger.error('unable to connect: %s, %s', device.name, device.ip_add
        continue

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

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

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

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

util.py

```
import csv
from pprint import pprint

import logging

from devclass import NetworkDevice
from devclass import NetworkDeviceIOS

logger = logging.getLogger('main.util')

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

    logger.info('Reading device info from: %s', devices_file)

    devices_list = []

    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):

    logger.info('Writing device info to: %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

logger = logging.getLogger('main.devclass')

#---- 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 interfa

#==== 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)
        logger.info('devclass: created IOS device: %s %s', name, ip)
```

```
#---------------------------------------------------------------------------
    def connect(self):
        print '--- connecting IOS: telnet: '+self.ip_address+' for: '+self.u

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

        # Check for failure
        if result != 0:
            logger.warn('devclass: --- Timout or unexpected reply from devic
            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:
            logger.warn('devclass: --- Timeout or unexpected username reply
            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:
            logger.warn('devclass: --- Timeout or unexpected password reply
            return 0

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

        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:
            logger.warn('devclass: --- Timeout or unexpected terminal length
            return False

        else: return True
```