Step 1

Create a unit test module. In the module, create a string that will be used to create the test CSV device information file, which will be read by the `read_devices_info` function. The actual content is not important, but the string must match the format of device-name, OS version, IP address, username, password. You will also need to create a variable (a list of lists) that contains a value that should match the result returned by the `read_devices_info` function.

Answer

Open a text editor to create the unit test and create the initial test string.

```
import unittest
import filecmp
from pprint import pprint

import util

csv_test_input_filename  = 'csv-in-test-devices'
csv_test_output_filename = 'csv-out-test-devices'

# Input file for testing. Note the '\r\n', required for file comparison test
csv_input_file_string = \
         ('test-1,test-os,1.1.1.1,test-username1,test-password1\r\n'
          'test-2,test-os,1.1.1.2,test-username2,test-password2\r\n'
          'test-3,test-os,1.1.1.3,test-username3,test-password3\r\n'
          'test-4,test-os,1.1.1.4,test-username4,test-password4\r\n')
print '---- CSV Test Devices Input String ----------------------------------'
pprint(csv_input_file_string)

# Devices list for comparison - this is the data that should be created
# when the file above is read by our code under test.
csv_test_devices_list = \
         [['test-1','test-os','1.1.1.1','test-username1','test-password1'],
          ['test-2','test-os','1.1.1.2','test-username2','test-password2'],
          ['test-3','test-os','1.1.1.3','test-username3','test-password3'],
          ['test-4','test-os','1.1.1.4','test-username4','test-password4']]
print '---- CSV Test Devices List ------------------------------------------'
pprint(csv_test_devices_list)
```

Step 2

Create a unit test class that is called `TestUtil` which derives from `unittest.TestCase`.

Answer

```
class TestUtil(unittest.TestCase):
```

Step 3

Define your `setUp()` and `tearDown()` methods within your `TestUtil` class.

Answer

```
#============================================================================
class TestUtil(unittest.TestCase):

    def setUp(self):

        # Create the CSV file for testing
        with open(csv_test_input_filename, 'w') as file:
            file.write(csv_input_file_string)

    def tearDown(self):

        pass
```

Step 4

Define test cases within your `TestUtil` class for testing the reading of CSV data into Python lists

---

Answer

```
    def test_csv_read(self):

        print '\n**** Testing reading CSV file ***'

        # Test that we can correctly read in CSV values
        csv_devices_list = util.read_devices_info(csv_test_input_filename)
        self.assertEqual(csv_devices_list, csv_test_devices_list,
                                                "Failed read device list")
```

Step 5

Run your unit test by running your test module as you would any Python application. The output should look something like the following:

Answer

```
$ python test_util.py
---- CSV Test Devices Input String ----------------------------------------
'test-1,test-os,1.1.1.1,test-username1,test-password1\r\ntest-2,test-os,1.1.
---- CSV Test Devices List ------------------------------------------------
[['test-1', 'test-os', '1.1.1.1', 'test-username1', 'test-password1'],
 ['test-2', 'test-os', '1.1.1.2', 'test-username2', 'test-password2'],
 ['test-3', 'test-os', '1.1.1.3', 'test-username3', 'test-password3'],
 ['test-4', 'test-os', '1.1.1.4', 'test-username4', 'test-password4']]

**** Testing reading CSV file ***
.
**** Testing writing CSV file ***
.
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
$
```

The complete application should look similar to:

```
import unittest
import filecmp
from pprint import pprint

import util

csv_test_input_filename  = 'csv-in-test-devices'
csv_test_output_filename = 'csv-out-test-devices'

# Input file for testing. Note the '\r\n', required for file comparison test
csv_input_file_string = \
         ('test-1,test-os,1.1.1.1,test-username1,test-password1\r\n'
          'test-2,test-os,1.1.1.2,test-username2,test-password2\r\n'
          'test-3,test-os,1.1.1.3,test-username3,test-password3\r\n'
          'test-4,test-os,1.1.1.4,test-username4,test-password4\r\n')
print '---- CSV Test Devices Input String ---------------------------------'
pprint(csv_input_file_string)

# Devices list for comparison - this is the data that should be created
# when the file above is read by our code under test.
csv_test_devices_list = \
         [['test-1','test-os','1.1.1.1','test-username1','test-password1'],
          ['test-2','test-os','1.1.1.2','test-username2','test-password2'],
          ['test-3','test-os','1.1.1.3','test-username3','test-password3'],
          ['test-4','test-os','1.1.1.4','test-username4','test-password4']]
print '---- CSV Test Devices List -----------------------------------------'
pprint(csv_test_devices_list)
```

```
#==============================================================================
class TestUtil(unittest.TestCase):

    def setUp(self):

        # Create the CSV file for testing
        with open(csv_test_input_filename, 'w') as file:
            file.write(csv_input_file_string)

    def test_csv_read(self):

        print '\n**** Testing reading CSV file ***'

        # Test that we can correctly read in CSV values
        csv_devices_list = util.read_devices_info(csv_test_input_filename)
        self.assertEqual(csv_devices_list, csv_test_devices_list,
                                                    "Failed read device list



    def test_csv_write(self):

        print '\n**** Testing writing CSV file ***'

        # Test that the output CSV is correct
        util.write_devices_info(csv_test_output_filename, csv_test_devices_l
        self.assertTrue(filecmp.cmp(csv_test_input_filename,
                                    csv_test_output_filename), "Failed CSV w

    def tearDown(self):

        pass

if __name__ == '__main__':
    unittest.main()
```

Step 6

Create a unit test module for testing the `devclass` module.

Answer

As in the previous procedure, you will create your unit test module, import `unittest`, and create your test class which inherits from `unittest.TestCase` and has `setUp()` and `tearDown()` methods.

```
import unittest

class TestDevice(unittest.TestCase):

    def setUp(self):

    def tearDown(self):

if __name__ == '__main__':
    unittest.main()
```

Step 7

Define global variables that you will use for communicating with one device in your lab environment. For example, name of 'test-device', IP address 10.30.30.1, username 'cisco', password 'cisco'.

Answer

```
# Device information for the actual device we will be testing against
device_name = 'test-device'
device_ip   = '10.30.30.1'
device_user = 'cisco'
device_pw   = 'cisco'
```

Step 8

During your setup for each test, create a device object based on the global values defined previously.

Answer

```
# Create device for testing login, connectivity, etc.
self.device = NetworkDeviceIOS(device_name,device_ip,
                               device_user,device_pw)
```

Step 9

Write unit tests to test the device object's attributes, and its ability to execute `connect()`, `get_interfaces()`, and `get_routes()`. Note that in the test, you should be comparing not only whether the command completed successfully, but also whether the data returned from the command (for `show interfaces` and `show routes`) seems to be correct. You cannot check all the returned string, but you need to check something to indicate that the data returned is truly the output of that specific command.

```
def test_attributes(self):

        print '\n**** Testing device object creation and values ****'

        # Test to make sure that the device object has correct values as set
        self.assertEqual(self.device.name,device_name,"Incorrect device name
        self.assertEqual(self.device.ip_address,device_ip,"Incorrect IP addr
        self.assertEqual(self.device.username,device_user,"Incorrect usernam
        self.assertEqual(self.device.password, device_pw,"Incorrect password

    def test_connect(self):

        print '\n**** Testing device object connectivity to real device ****

        # Test that we can connect to the device (login)
        session = self.device.connect()
        self.assertNotEqual(session, 0, "Failed connection to device")

        self.device.disconnect()  # Clean up the session

    def test_show_interfaces(self):

        print '\n**** Testing device show interfaces command ****'

        # First must connect to the device
        session = self.device.connect()
        self.assertNotEqual(session, 0, "Failed connection to device")

        # Set terminal length to 0 for long replies
        self.assertTrue(self.device.set_terminal_length())

        # Run show interfaces command
        intf_output = self.device.get_interfaces()
```

```
        # Test the command ran successfully
        self.assertNotEqual(intf_output, 0, "Failed show interfaces command"

        # Test the data returned from the command
        # Note we test for 'Loopback', which will be present always
        self.assertNotEqual(len(intf_output), 0, "Show interfaces: no data")
        self.assertNotEqual(intf_output.find('Loopback'), -1,
                                          "Show interfaces: incorrect data")

        self.device.disconnect()  # Clean up the session

    def test_show_routes(self):

        print '\n**** Testing device show routes command ****'

        # First must connect to the device
        session = self.device.connect()
        self.assertNotEqual(session, 0, "Failed connection to device")

        # Set terminal length to 0 for long replies
        self.assertTrue(self.device.set_terminal_length())

        # Run show ip route command
        routes_output = self.device.get_routes()

        # Test the command ran successfully
        self.assertNotEqual(routes_output, 0, "Failed show ip route command"

        # Test the data returned from the command
        # Note we test for 'OSPF' just part of the legend for the command ou
        self.assertNotEqual(len(routes_output), 0, "Show ip routes: no data"
        self.assertNotEqual(routes_output.find('OSPF'), -1,
                                            "Show ip routes: incorrect data"


        self.device.disconnect()  # Clean up the session
```

Step 10

Execute your unit test and verify that the tests ran successfully.

Answer

Your output should look similar to the following:

```
$ python test_devclass.py

**** Testing device object creation and values ****
.
**** Testing device object connectivity to real device ****
--- connecting IOS: telnet: cisco/cisco
.
**** Testing device show interfaces command ****
--- connecting IOS: telnet: cisco/cisco
.
**** Testing device show routes command ****
--- connecting IOS: telnet: cisco/cisco
.
----------------------------------------------------------------------
Ran 4 tests in 2.351s

OK
$
```

Complete App:

```
import unittest
from pprint import pprint

from devclass import NetworkDeviceIOS

# Device information for the actual device we will be testing against
device_name = 'test-device'
device_ip   = '10.30.30.1'
device_user = 'cisco'
device_pw   = 'cisco'

class TestDevice(unittest.TestCase):


    def setUp(self):

        # Create device for testing login, connectivity, etc.
        self.device = NetworkDeviceIOS(device_name,device_ip,device_user,dev

    def test_attributes(self):

        print '\n**** Testing device object creation and values ****'

        # Test to make sure that the device object has correct values as set
        self.assertEqual(self.device.name, device_name, "Incorrect device na
        self.assertEqual(self.device.ip_address, device_ip, "Incorrect IP ad
        self.assertEqual(self.device.username, device_user, "Incorrect usern
        self.assertEqual(self.device.password, device_pw, "Incorrect passwor

    def test_connect(self):

        print '\n**** Testing device object connectivity to real device ****
```

```
        # Test that we can connect to the device (login)
        session = self.device.connect()
        self.assertNotEqual(session, 0, "Failed connection to device")

        self.device.disconnect()  # Clean up the session


    def test_show_interfaces(self):

        print '\n**** Testing device show interfaces command ****'

        # First must connect to the device
        session = self.device.connect()
        self.assertNotEqual(session, 0, "Failed connection to device")

        # Set terminal length to 0 for long replies
        self.assertTrue(self.device.set_terminal_length())

        # Run show interfaces command
        intf_output = self.device.get_interfaces()

        # Test the command ran successfully
        self.assertNotEqual(intf_output, 0, "Failed show interfaces command"

        # Test the data returned from the command
        # Note we test for 'Loopback', which will be present always
        self.assertNotEqual(len(intf_output), 0, "Show interfaces: no data")
        self.assertNotEqual(intf_output.find('Loopback'), -1,
                                                "Show interfaces: incorrect d

        self.device.disconnect()  # Clean up the session
```

```
    def test_show_routes(self):

        print '\n**** Testing device show routes command ****'

        # First must connect to the device
        session = self.device.connect()
        self.assertNotEqual(session, 0, "Failed connection to device")

        # Set terminal length to 0 for long replies
        self.assertTrue(self.device.set_terminal_length())

        # Run show ip route command
        routes_output = self.device.get_routes()

        # Test the command ran successfully
        self.assertNotEqual(routes_output, 0, "Failed show ip route command"

        # Test the data returned from the command
        # Note we test for 'OSPF' just part of the legend for the command ou
        self.assertNotEqual(len(routes_output), 0, "Show ip routes: no data"
        self.assertNotEqual(routes_output.find('OSPF'), -1,
                                                "Show ip routes: incorrec

        self.device.disconnect()  # Clean up the session

    def tearDown(self):

        pass

if __name__ == '__main__':
    unittest.main()
```