LAB – CREATING FUNCTIONS THAT RETURN VALUES

OBJECTIVE

In this lab, you will create functions that return values.

PART 1

Open a terminal and switch to the lab directory

STEP 1: OPEN A TERMINAL WINDOW

Double-click the Terminal icon on the desktop to open the terminal window for use in this lab.

STEP 2: CHANGE DIRECTORY

Change to the directory **labs/prne/** in the user home directory, which holds the files for the course labs.

```
~$ cd labs/prne/
```

PART 2

Copy the required configuration for this lab from flash into the running-configuration of the CSR1000v and R2 routers.

STEP 1: COPY CONFIGURATION ON CSR1000V (R1)

In the CSR1000v router console, enter **privileged exec** mode and copy the additional configuration located in the **CONFIG5** file to the running-configuration using the commands:

```
CSR1kv> enable
CSR1kv# copy flash:CONFIG5 running-config
```

```
CSR1kv>enable
CSR1kv#copy flash:CONFIG5 running-config
Destination filename [running-config]?
348 bytes copied in 0.054 secs (6444 bytes/sec)
CSR1kv#
```

STEP 2: VERIFY CONFIGURATION ON CSR1000V (R1)

Verify the new running-configuration by using the command:

```
CSR1kv# show run | section router
```

NOTE:
To get the pipe (**|**) command, you may need to use the keys **Shift** + **#** rather than your local keyboard input.

```
CSR1kv#show run ¦ section router
router ospf 10
 network 172.16.1.0 0.0.0.3 area 1
 network 192.168.56.0 0.0.0.255 area 0
CSR1kv#
```

STEP 3: COPY CONFIGURATION ON R2

In the R2 router console, enter **privileged exec** mode and copy the additional configuration located in the **CONFIG6** file to the running-configuration using the commands:

```
R2> enable
R2# copy flash:CONFIG6 running-config
```

```
R2>enable
R2#copy flash:CONFIG6 running-config
Destination filename [running-config]?
923 bytes copied in 0.120 secs (7692 bytes/sec)
R2#
```

STEP 4: VERIFY CONFIGURATION ON R2

Verify the new running-configuration by using the command:

```
CSR1kv# show run | section router
```

NOTE:
To get the pipe (**|**) command, you may need to use the keys **Shift** + **#** rather than your local keyboard input.

```
R2#show run ¦ section router
router ospf 10
 network 10.76.98.0 0.0.0.255 area 2
 network 172.16.1.0 0.0.0.3 area 1
 network 192.0.2.0 0.0.0.15 area 2
 network 192.168.56.0 0.0.0.255 area 0
 network 198.51.100.0 0.0.0.7 area 2
 network 203.0.113.0 0.0.0.3 area 2
 default-information originate
R2#
```

PART 3

Open **Visual Studio Code**, create a new file and save it with a filename of **creating-functions-that-return-values-part-3.py**, ensuring to save the file in the **~/labs/prne/** directory.
This python application will:

- Create a function to connect to a device in your virtual network, returning the pexpect **session** object.
- Create another function to use the session object to read interface information from the device, returning the output of the **show interface summary** command.
- Your main code will call the connect function, call the show interface function, and display the output just as it was received from the device

STEP 1: CREATE FUNCTION TO CONNECT TO DEVICE

Create a function that takes the IP address, username, and password of the device as input parameters. The function should use pexpect to connect to the device and if successful, the function should return the pexpect **session** object. If unsuccessful, the function should return 0.

```
# Import required modules/packages/library
import pexpect

# The following code connects to a device
def connect(dev_ip, username, password):
    """
    Connects to device using pexpect
    :dev_ip: The IP address of the device we are connecting to
    :username: The username that we should use when logging in
    :password: The password that we should use when logging in
    =return: pexpect session object if successful, 0 otherwise
    """

    print('--- attempting to: telnet ' + dev_ip)
    session = pexpect.spawn('telnet ' + dev_ip, encoding='utf-8',
                            timeout=20)

    result = session.expect(['Username:', pexpect.TIMEOUT, pexpect.EOF])
    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0

    print('--- attempting to: username: ' + username)

    # Successfully got username prompt, logging with username
    session.sendline(username)

    result = session.expect(['Password:', pexpect.TIMEOUT, pexpect.EOF])
    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0

    print('--- attempting to: password: ' + password)

    # Successfully got password prompt, logging in with password
    session.sendline(password)
    session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

    return session  # return pexpect session object to caller
```

STEP 2: CREATE FUNCTION TO RUN COMMAND

Create a function that takes as input a pexpect session object, and performs a **show interface summary** command to receive interface information from the device. The function should return the output of the command.

```
# The following function gets and returns interface information
def show_int_summary(session):
    """
    Runs 'show int summary' command on device and returns
    output from device in a string

    :session: The pexpect session for communication with device

    =return: string of output from device
    """

    print('--- show interface summary command')
    session.sendline('show interface summary')
    result = session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

    print('--- getting interface command output')
    show_int_brief_output = session.before

    return show_int_brief_output
```

STEP 3: CREATE MAIN CODE

Your **main** code should call your connect function, passing in the actual IP address, username, and password, and receiving the session object in return. The main code should then call your show int brief function, passing in the session object, and receiving the output of the command in return.

```
# Main program: connect to device, show interface, display
if __name__ == '__main__':

    session = connect('192.168.56.101', 'prne', 'cisco123!')
    if session == 0:
        print('--- Session attempt unsuccessful, exiting.')
        exit()

    output_data = show_int_summary(session)

    print('')
    print('Show Interface Output')
    print('----------------------------------------------------')
    print('')

    print(output_data)

    # Close the session
    session.sendline('quit')
    session.kill(0)
```

STEP 4: SAVE, RUN AND VERIFY APPLICATION

Save you application and then run it from the terminal rather than from within visual studio code.

```
~/labs/prne$ python3 creating-functions-that-return-values-part-3.py
```

The output from your application will be displayed in your terminal window, verify that it is comparable to below.

```
devasc@labvm:~/labs/prne$ python3 creating-functions-that-return-values-part-3.py
--- attempting to: telnet 192.168.56.101
--- attempting to: username: prne
--- attempting to: password: cisco123!
--- show interface summary command
--- getting interface command output

Show Interface Output
-----------------------------------------------------

show interface summary

 *: interface is up
 IHQ: pkts in input hold queue     IQD: pkts dropped from input queue
 OHQ: pkts in output hold queue    OQD: pkts dropped from output queue
 RXBS: rx rate (bits/sec)          RXPS: rx rate (pkts/sec)
 TXBS: tx rate (bits/sec)          TXPS: tx rate (pkts/sec)
 TRTL: throttle count

  Interface                  IHQ       IQD       OHQ       OQD      RXBS      RXPS      TXBS      TXPS      TRTL
-------------------------------------------------------------------------------------------------------------------
* GigabitEthernet1             0         0         0         0         0         0         0         0         0
* GigabitEthernet2             0         0         0         0         0         0         0         0         0
CSR1kv
```

PART 4

Open **Visual Studio Code**, create a new file and save it with a filename of **creating-functions-that-return-values-part-4.py**, ensuring to save the file in the **~/labs/prne/** directory.

This python application will:

- Create a function to read in the device information from the file **devices-09.txt**.
- Create another function to connect to a device in your virtual network, returning the pexpect **session** object.
- Create another function to use the session object to read interface information from the device, returning the output of the **show interface summary** command.
- Your main code will call the connect function, call the show interface function, and display the output just as it was received from the device

STEP 1: CREATE FUNCTION TO READ INFORMATION

Create a function that reads the device information from the file **devices-09.txt** and return a list of devices, with information for each device stored in a dictionary.

```
# Import required modules/packages/library
import pexpect

# Reads the device information from the file
def read_devices_info(filename):

    devices_list = []

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

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

        device_info = {}
        device_info['name'] = device_info_list[0]
        device_info['ip'] = device_info_list[1]
        device_info['username'] = device_info_list[2]
        device_info['password'] = device_info_list[3]

        devices_list.append(device_info)

    return devices_list
```

STEP 2: CREATE FUNCTION TO CONNECT TO DEVICE

Create a function that takes the IP address, username, and password of the device as input parameters. The function should use pexpect to connect to the device and if successful, the function should return the pexpect **session** object. If unsuccessful, the function should return 0.

```
# The following code connects to a device
def connect(dev_ip, username, password):

    print('--- connecting IOS: telnet ' + dev_ip)

    session = pexpect.spawn('telnet ' + dev_ip, encoding='utf-8', timeout=20)
    result = session.expect(['Username:', pexpect.TIMEOUT, pexpect.EOF])

    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0
    print('--- attempting to: username: ' + username)

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

    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0

    print('--- attempting to: password: ' + password)

    # Successfully got password prompt, logging in with password
    session.sendline(password)
    session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0

    # Must set terminal length to zero for long replies, no pauses
    print('--- setting terminal length to 0')
    session.sendline('terminal length 0')
    result = session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0

    # return pexpect session object to caller
    return session
```

STEP 3: CREATE FUNCTION TO RUN COMMAND

Create a function that takes as input a pexpect session object, and performs a **show interface summary** command to receive interface information from the device. The function should return the output of the command.

```
# The following function gets and returns interface information
def show_int_summary(session):

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

    # Check for failure
    if result != 0:
        print('--- Timeout or unexpected reply from device')
        return 0

    show_int_summary_output = session.before

    return show_int_summary_output
```

STEP 4: CREATE A FUNCTION TO DISPLAY INFORMATION

Create a function which displays information about a device, including device info (name, IP, username, password), and the interface information for the device.

```
# The following function prints device information
def print_device_info(device_info, show_int_output):

    print('-----------------------------------------------------')
    print('    Device Name:      ', device_info['name'])
    print('    Device IP:        ', device_info['ip'])
    print('    Device username:  ', device_info['username'],)
    print('    Device password:  ', device_info['password'])

    print('')
    print('    Show Interface Output')
    print('')

    print(show_int_output)
    print('-----------------------------------------------------')
```

STEP 5: CREATE MAIN CODE

Create main application code that reads device information from the **devices-09.txt** file, and then iterates through the list of devices, using your created functions to connect to each one, read interface information, and displays nicely formatted output for each device.

```
# Main program: connect to device, show interface, display
if __name__ == '__main__':

    devices_list = read_devices_info('devices-09.txt')

    for device_info in devices_list:

        session = connect(device_info['ip'],
                          device_info['username'],
                          device_info['password'])
        if session == 0:
            print('--- Session attempt unsuccessful ---')
            continue

        show_int_output = show_int_summary(session)

        print_device_info(device_info, show_int_output)

        session.sendline('quit')
        session.kill(0)
```

STEP 6: SAVE, RUN AND VERIFY APPLICATION

Save you application and then run it from the terminal rather than from within visual studio code.

```
~/labs/prne$ python3 creating-functions-that-return-values-part-4.py
```

The output from your application will be displayed in your terminal window, verify that it is comparable to below.

```
devasc@labvm:~/labs/prne$ python3 creating-functions-that-return-values-part-4.py
--- connecting IOS: telnet 192.168.56.101
--- attempting to: username: prne
--- attempting to: password: cisco123!
--- setting terminal length to 0
-----------------------------------------------------
    Device Name:       CSR1kv
    Device IP:         192.168.56.101
    Device username:   prne
    Device password:   cisco123!

    Show Interface Output

show interface summary

 *: interface is up
 IHQ: pkts in input hold queue     IQD: pkts dropped from input queue
 OHQ: pkts in output hold queue    OQD: pkts dropped from output queue
 RXBS: rx rate (bits/sec)          RXPS: rx rate (pkts/sec)
 TXBS: tx rate (bits/sec)          TXPS: tx rate (pkts/sec)
 TRTL: throttle count

  Interface                  IHQ       IQD       OHQ       OQD      RXBS      RXPS      TXBS      TXPS      TRTL
-----------------------------------------------------------------------------------------------------------------
* GigabitEthernet1             1         0         0         0         0         0         0         0         0
* GigabitEthernet2             0         0         0         0         0         0         0         0         0
CSR1kv
-----------------------------------------------------
--- connecting IOS: telnet 192.168.56.130
--- attempting to: username: prne
--- attempting to: password: cisco123!
--- setting terminal length to 0
-----------------------------------------------------
    Device Name:       R2
    Device IP:         192.168.56.130
    Device username:   prne
    Device password:   cisco123!

    Show Interface Output

show interface summary

 *: interface is up
 IHQ: pkts in input hold queue     IQD: pkts dropped from input queue
 OHQ: pkts in output hold queue    OQD: pkts dropped from output queue
 RXBS: rx rate (bits/sec)          RXPS: rx rate (pkts/sec)
 TXBS: tx rate (bits/sec)          TXPS: tx rate (pkts/sec)
 TRTL: throttle count

  Interface                  IHQ       IQD       OHQ       OQD      RXBS      RXPS      TXBS      TXPS      TRTL
-----------------------------------------------------------------------------------------------------------------
* GigabitEthernet1             0         0         0         0         0         0         0         0         0
* GigabitEthernet2             0         0         0         0         0         0         0         0         0
* Loopback0                    0         0         0         0         0         0         0         0         0
* Loopback1                    0         0         0         0         0         0         0         0         0
* Loopback2                    0         0         0         0         0         0         0         0         0
* Loopback3                    0         0         0         0         0         0         0         0         0
R2
-----------------------------------------------------
```

PART 5 (OPTIONAL BUT HIGHLY RECOMMENDED)

As this lab is completed in NETLAB+ and your code files will be erased when the reservation ends, it is advisable to save your files in GitHub under your repository for this course.