LAB – USING CONDITIONALS WITH NETWORK DEVICES

OBJECTIVE

In this lab, you will use conditionals to perform typical network tasks, such as:

- Using an if statement to test for 'empty'
- Using a nested if statement
- Using an else statement
- Using an inline if statement

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

Open **Visual Studio Code**, create a new file and save it with a filename of **using-conditionals-with-network-devices-part-2.py**. Ensuring to save the file in the **~/labs/prne/** directory, as otherwise the code will require modification to find the associated files that are used. This python application will count how many routes are associated with each Gigabit Ethernet port from a **show ip route** command, using the file **ip-routes.txt** as the input for your application.

STEP 1: IMPORT MODULES/PACKAGES/LIBRARY

Import the pprint function to enable displaying the list nicely formatted and the re module to allow the use of regular expressions.

```
# Import required modules/packages/library
from pprint import pprint
import re
```

STEP 2: CREATE REGULAR EXPRESSION

Create a regular expression pattern to match the interface 'GigabitEthernetn/n/n/n' string.

```
# Create regular expression to match Gigabit interface names
gig_pattern = re.compile(r'(GigabitEthernet)(\d\/\d\/\d\/\d)')
```

STEP 3: CREATE DICTIONARY

Create a dictionary to hold the count of routes that will get forwarded out each interface.

```
# Create a dictionary to hold the count of routes
# forwarded out each interface
routes = {}
```

STEP 4: OPEN FILE FOR READING

Open the file **ip-routes.txt** for reading.

```
# Read all lines of IP routing information
file = open('ip-routes.txt', 'r')
for line in file:
```

STEP 5: MATCH LINE

On each line, check if there is a match for Gigabit Ethernet interfaces regular expression.

```
    # Match for Gigabit Ethernet
    match = gig_pattern.search(line)
```

STEP 6: INCREMENT COUNT IF MATCH

If there is a match, add one to the count of routes for that specific interface.

```
    # Check to see if we matched the Gig Ethernet string
    if match:
        intf = match.group(2)  # get the interface from the match
        routes[intf] = routes[intf]+1 if intf in routes else 1
```

STEP 7: DISPLAY HEADER

Display in the terminal a header.

```
# Display heading
print('')
print('Number of routes per interface')
print('-------------------------------')
```

STEP 8: DISPLAY THE ROUTE COUNT

Display in the terminal the counts of IP routes that are getting forwarded out each interface.

```
# Display the routes per Gigabit interface
pprint(routes)

# Display a blank line to make easier to read
print('')
```

STEP 9: CLOSE FILE

Close the file.

```
# Close the file
file.close()
```

STEP 10: 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 using-conditionals-with-network-devices-part-2.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 using-conditionals-with-network-devices-part-2.py

Number of routes per interface
------------------------------
{'0/0/0/0': 2, '0/0/0/1': 1, '0/0/0/2': 2}
```

PART 3

Open **Visual Studio Code**, create a new file and save it with a filename of **using-conditionals-with-network-devices-part-3.py**. Ensuring to save the file in the **~/labs/prne/** directory, as otherwise the code will require modification to find the associated files that are used. This python application will use if and elif statements to categorize and tabulate information about a set of devices, using the file **devices-06.txt** as the input for your application.

STEP 1: IMPORT PPRINT

Import the pprint function to enable displaying the list nicely formatted.

```
# Import required modules/packages/library
from pprint import pprint
```

STEP 2: DISPLAY HEADING

Display in the terminal a header.

```
# Display heading
print('')
print('Counts of different OS-types for all devices')
print('--------------------------------------------')
```

STEP 3: CREATE DICTIONARY

Create a dictionary of OS types, with the keys "Cisco IOS", "Cisco Nexus", "Cisco IOS-XR", and "Cisco IOS-XE".

```
# Create a dictionary of OS types
os_types = {'Cisco IOS':    {'count': 0, 'devs': []},
            'Cisco Nexus':  {'count': 0, 'devs': []},
            'Cisco IOS-XR': {'count': 0, 'devs': []},
            'Cisco IOS-XE': {'count': 0, 'devs': []}}
```

STEP 4: OPEN FILE FOR READING

Open the file **devices-06.txt** for reading.

```
# Read all lines of IP routing information
file = open('devices-06.txt', 'r')
for line in file:
```

STEP 5: CREATE LIST

Put the contents of the file into a list, splitting the string using the comma.

```
    # Put device info into list
    device_info_list = line.strip().split(',')
```

STEP 6: CREATE DICTIONARY

For every OS type, create a dictionary with two items: a count of the number of devices of this OS type, and a list of device names of the devices of this device type. Also, for every device, determine the OS type.

```
    # Put device information into dictionary for this one device
    device_info = {}  # Create a dictionary of device info
    device_info['name'] = device_info_list[0]
    device_info['os-type'] = device_info_list[1]

    # Get the device name
    name = device_info['name']

    # Get the OS-type for comparisons
    os = device_info['os-type']
```

STEP 7: INCREMENT COUNTS

Depending on the OS type, increment the count of the correct OS type in your dictionary, and add the name of the device to the list of devices.

```
    # Based on the OS-type, increment the count and add name to list
    if os == 'ios':
        os_types['Cisco IOS']['count'] += 1
        os_types['Cisco IOS']['devs'].append(name)

    elif os == 'nx-os':
        os_types['Cisco Nexus']['count'] += 1
        os_types['Cisco Nexus']['devs'].append(name)

    elif os == 'ios-xr':
        os_types['Cisco IOS-XR']['count'] += 1
        os_types['Cisco IOS-XR']['devs'].append(name)

    elif os == 'ios-xe':
        os_types['Cisco IOS-XE']['count'] += 1
        os_types['Cisco IOS-XE']['devs'].append(name)

    else:
        print("   Warning: unknown device type:", os)
```

STEP 8: DISPLAY THE OS COUNT

Display in the terminal the OS types count nicely formatted.

```
# Display the OS types
pprint(os_types)

# Display a blank line to make easier to read
print('')
```

STEP 9: CLOSE FILE

Close the file.

```
# Close the file
file.close()
```

STEP 10: 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 using-conditionals-with-network-devices-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 using-conditionals-with-network-devices-part-3.py

Counts of different OS-types for all devices
--------------------------------------------
{'Cisco IOS': {'count': 2, 'devs': ['d01-is', 'd02-is']},
 'Cisco IOS-XE': {'count': 2, 'devs': ['d07-xe', 'd08-xe']},
 'Cisco IOS-XR': {'count': 2, 'devs': ['d05-xr', 'd06-xr']},
 'Cisco Nexus': {'count': 2, 'devs': ['d03-nx', 'd04-nx']}}
```

PART 4

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 5

Open **Visual Studio Code**, create a new file and save it with a filename of **using-conditionals-with-network-devices-part-4.py**. Ensuring to save the file in the **~/labs/prne/** directory. This python application will read route information from a device and then tabulate and display routes per interface.

STEP 1: IMPORT MODULES/PACKAGES/LIBRARY

Import the pprint function to enable displaying the list nicely formatted, the pexpect library to allow connecting to the device and the re module to allow the use of regular expressions.

```
# Import required modules/packages/library
import pexpect
from pprint import pprint
import re
```

STEP 2: DISPLAY HEADING

Display the heading for the output in the terminal.

```
# Display heading
print('')
print('Interfaces, routes list, routes details')
print('---------------------------------------')
```

STEP 3: CREATE REGULAR EXPRESSIONS

Create regular expressions to match the OSPF routes and the required interface type of GigabitEthernet.

```
# Create regular expressions to match interfaces and OSPF
OSPF_pattern = re.compile(r'O.+')
intf_pattern = re.compile(r'(GigabitEthernet)(\d)')
```

Create regular expressions to match the OSPF route prefix and length (if available) and also the next-hop for that route.

```
# Create regular expressions to match prefix and routes
prefix_pattern = re.compile(r'(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\/?\d?\d?)')
route_pattern = re.compile(r'via (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})')
```

STEP 4: CONNECT TO THE ROUTER

Create a session using the pexpect 'spawn' method to telnet into the router using a check to inform you if you are unable to login.

```
# Connect to device and run 'show ip route' command
print('--- connecting telnet 192.168.56.101 with prne/cisco123!')

session = pexpect.spawn('telnet 192.168.56.101', 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')
    exit()

# Enter username
session.sendline('prne')
result = session.expect('Password:')

# Enter password
session.sendline('cisco123!')
result = session.expect('>')
```

STEP 5: DISPLAY ROUTE INFORMATION

Once logged in, due to the length of the expected output, the terminal must be set to not pause output on the terminal. Then run the **show ip route** command and display the output in the terminal.

```
# 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('>')

# Run the 'show ip route' commanmd on device
print('--- successfully logged into device, running show ip route
command')
session.sendline('show ip route')
result = session.expect('>')

# Display the output of the command, for comparison
print('--- show ip route output:')
show_ip_route_output = session.before
print(show_ip_route_output)
```

STEP 6: SPLIT INPUT INTO LINES

Split the output from the show ip route command into separate lines.

```
# Get the output from the command into a list of lines from the output
routes_list = show_ip_route_output.splitlines()
```

STEP 7: CREATE LIST OF ROUTES

From the separated lines from the show ip route command, create a list that holds only the OSPF routes and create a dictionary holding the destination IP address/subnet (prefix/length) and the next-hop.

```
# Create dictionary to hold number of routes per interface
intf_routes = {}

# Go through the list of routes to get routes per interface
for route in routes_list:

    OSPF_match = OSPF_pattern.search(route)
    if OSPF_match:

        # Match for GigabitEthernet interfaces
        intf_match = intf_pattern.search(route)

        # Check to see if we matched the GigabitEthernet interfaces string
        if intf_match:

            # Get the interface from the match
            intf = intf_match.group(2)

            # If route list not yet created, do so now
            if intf not in intf_routes:
                intf_routes[intf] = []

            # Extract the prefix (destination IP address/subnet)
            prefix_match = prefix_pattern.search(route)
            prefix = prefix_match.group(1)

            # Extract the route
            route_match = route_pattern.search(route)
            next_hop = route_match.group(1)

            # Create dictionary for this route,
            # and add it to the list
            route = {'prefix': prefix, 'next-hop': next_hop}
            intf_routes[intf].append(route)
```

STEP 8: DISPLAY THE DATA

Display in the terminal nicely formatted, the interfaces with the next-hop and prefix for every OSPF route.

```
# Display a blank line to make easier to read
print('')

# Display a title
print('OSPF routes out of GigabitEthernet interfaces:')

# Display the GigabitEthernet interfaces routes
pprint(intf_routes)

# Display a blank line to make easier to read
print('')
```

STEP 9: CLOSE FILE

Close the file.

```
# Close the file
file.close()
```

STEP 10: 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 using-conditionals-with-network-devices-part-5.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 using-conditionals-with-network-devices-part-5.py

Interfaces, routes list, routes details
---------------------------------------
--- connecting telnet 192.168.56.101 with prne/cisco123!
--- setting terminal length to 0
--- successfully logged into device, performing show ip route command
--- show ip route output:
show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is 172.16.1.2 to network 0.0.0.0

O*E2  0.0.0.0/0 [110/1] via 172.16.1.2, 00:20:47, GigabitEthernet2
      10.0.0.0/24 is subnetted, 1 subnets
O IA     10.76.98.0 [110/910] via 192.168.56.130, 00:20:34, GigabitEthernet1
      172.16.0.0/16 is variably subnetted, 2 subnets, 2 masks
C        172.16.1.0/30 is directly connected, GigabitEthernet2
L        172.16.1.1/32 is directly connected, GigabitEthernet2
      192.0.2.0/28 is subnetted, 1 subnets
O IA     192.0.2.0 [110/910] via 192.168.56.130, 00:20:34, GigabitEthernet1
      192.168.56.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.56.0/24 is directly connected, GigabitEthernet1
L        192.168.56.101/32 is directly connected, GigabitEthernet1
      198.51.100.0/29 is subnetted, 1 subnets
O IA     198.51.100.0 [110/910] via 192.168.56.130, 00:20:34, GigabitEthernet1
      203.0.113.0/30 is subnetted, 1 subnets
O IA     203.0.113.0 [110/910] via 192.168.56.130, 00:20:34, GigabitEthernet1
CSR1kv

OSPF routes out of GigabitEthernet interfaces:
{'1': [{'next-hop': '192.168.56.130', 'prefix': '10.76.98.0'},
       {'next-hop': '192.168.56.130', 'prefix': '192.0.2.0'},
       {'next-hop': '192.168.56.130', 'prefix': '198.51.100.0'},
       {'next-hop': '192.168.56.130', 'prefix': '203.0.113.0'}],
 '2': [{'next-hop': '172.16.1.2', 'prefix': '0.0.0.0/0'}]}

devasc@labvm:~/labs/prne$
```

PART 6 (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.