Count Routes Per Interface

In this exercise, you will be using a text file that has information from a `show ip route` command. The file is **~/Desktop/PRNE/section09/ip-routes** and contains data that looks like:

```
Codes: C - connected, S - static, R - RIP, B - BGP, (>) - Diversion path
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type
C    10.11.13.0/24 is directly connected, 3d05h, MgmtEth0/0/CPU0/0
L    10.11.13.4/32 is directly connected, 3d05h, MgmtEth0/0/CPU0/0
S    10.16.0.0/16 [1/0] via 198.18.1.1, 3d05h
i L2 11.11.1.0/24 [115/10] via 58.0.0.21, 2d10h, GigabitEthernet0/0/0/0
S    11.11.2.0/24 is directly connected, 3d05h, Null0
i L2 11.11.3.0/24 [115/20] via 58.0.0.21, 2d10h, GigabitEthernet0/0/0/0
                  [115/20] via 49.0.0.30, 2d10h, GigabitEthernet0/0/0/2
i L2 11.11.4.0/24 [115/30] via 49.0.0.30, 2d10h, GigabitEthernet0/0/0/2
i L2 11.11.5.0/24 [115/10] via 48.0.0.27, 3d05h, GigabitEthernet0/0/0/1
```

You will count how many routes are associated with each GigabitEthernet port.

Note: You will be able to use several components of this lesson in this lab, in particular:

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

Step 1

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

Answer

```
from pprint import pprint
import re

# Create regular expression to match Gigabit interface names
gig_pattern = re.compile('(GigabitEthernet)([0-9]\/[0-9]\/[0-9]\/[0-9])')
```

Step 2

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

Answer

```
routes = {}
```

Step 3

Read the route information from the text file **~/Desktop/PRNE/section09/ip-routes**.

Answer

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

Note

Make sure you are in the **~/Desktop/PRNE/section09/** directory before running your script. This is because the **ip-routes** file the script is attempting to open is in current working directory.

Step 4

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

Answer

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

    # 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 5

When all lines have been read, print the counts of IP routes that are getting forwarded out each interface.

Answer

```
print ''
print 'Number of routes per interface'
print '-------------------------------'
pprint(routes)
print '' # Print final blank line
```

Step 6

Run your application and verify the output.

Answer

Your output should look similar to:

```
$ python ip-route.py

Number of routes per interface
-----------------------------------------
{'0/0/0/0': 14, '0/0/0/1': 6, '0/0/0/2': 20}
```

Tabulate OS Types

In this exercise, you will use `if` and `elif` statements to categorize and tabulate information about a set of devices. Use the **~/Desktop/PRNE/section09/devices** file as the input for your application.

Step 7

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

Answer

```
from pprint import pprint

# Print heading
print ''
print 'Counts of different OS-types for all devices'
print '============================================='

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 8

Read in the file of devices.

Answer

```
# Read all lines of device information from file
file = open('devices','r')
for line in file:

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

Note

Make sure you are in the **~/Desktop/PRNE/section09/** directory before running your script. This is because the **devices** file the script is attempting to open is in current working directory.

Step 9

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.

Answer

```
    # 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]

    name = device_info['name'] # get the device name
    os = device_info['os-type'] # get the OS-type for comparisons
```

Step 10

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.

Answer

```
    # 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 11

When all lines have been read print the results.

Answer

```
pprint(os_types)
print '' # Print final blank line

file.close() # Close the file since we are done with it
```

Step 12

Run your application and verify the output.

Answer

Your output should look similar to:

```
$ python os-types.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-xre']},
 'Cisco Nexus': {'count': 2, 'devs': ['d03-nx'. 'd04-nx']},
```

Tabulate OSPF Interfaces

In this exercise, you will read route info from a device. You will tabulate and print routes per interface.

```
import pexpect
from pprint import pprint
import re

# Print heading
print ''
print 'Interfaces, routes list, routes details'
print '--------------------------------------'

# Create regular expressions to match interfaces and OSPF
OSPF_pattern = re.compile('^O')
intf_pattern = re.compile('(GigabitEthernet)([0-9]\/[0-9])')

# Create regular expressions to match prefix and routes
prefix_pattern = re.compile('^O.{8}([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9
route_pattern = re.compile('via ([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1

# Connect to device and run 'show ip route' command
print '--- connecting telnet 10.30.30.1 with cisco/cisco'

session = pexpect.spawn('telnet 10.30.30.1', timeout=20)
result = session.expect(['Username:', pexpect.TIMEOUT])

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

# Successfully got username prompt, enter username
session.sendline('cisco')
result = session.expect('Password:')

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

Step 14

Read the route information using `show ip route.`

Answer

```
# Must set terminal length to zero for long replies
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, performing show ip route command
session.sendline('show ip route')
result = session.expect('>')

# Print out the output of the command, for comparison
print '--- show ip route output:'
show_ip_route_output = session.before
print show_ip_route_output

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

Step 15

Create a list of OSPF routes using the output of the `show ip route` command.

Answer

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

# 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:

        intf_match = intf_pattern.search( route ) # Match for Gigabit Ether

        # Check to see if we matched the Gig Ethernet string
        if intf_match:

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

Step 16

For every route created above, create a dictionary holding the destination IP address/subnet, and the next hop.

Answer

```
            if intf not in intf_routes: # If route list not yet created, do
                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 17

Print the data that you have tabulated.

Answer

```
pprint(intf_routes)
print '' # Print final blank line
```

```
Interfaces, routes list, routes details
---------------------------------------
--- connecting telnet 10.30.30.1 with cisco/cisco
--- 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 roo
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated root, % - next hop override

Gateway of last resort not set

      10.0.0.0/8 is variably subnetted, 11 subnets, 3 masks
C        10.1.1.0/24 is directly connected, GigabitEthernet0/1
L        10.1.1.1/32 is directly connected, GigabitEthernet0/1
C        10.1.2.0/30 is directly connected, GigabitEthernet0/2
L        10.1.2.1/32 is directly connected, GigabitEthernet0/2
C        10.1.3.0/30 is directly connected, GigabitEthernet0/3
L        10.1.3.1/32 is directly connected, GigabitEthernet0/3
O        10.2.3.0/30 [110/2] via 10.1.3.2, 00:04:14, GigabitEthernet0/3
                     [110/2] via 10.1.2.2, 00:04:14, GigabitEthernet0/2
O E2      10.11.12.0/24 [110/20] via 10.1.3.2, 00:04:14, GigabitEthernet0/3
                     [110/20] via 10.1.2.2, 00:04:24, GigabitEthernet0/2
C        10.30.30.1/32 is directly connected, Loopback0
O E2      10.30.30.2/32 [110/20] via 10.1.2.2, 00:04:24, GigabitEthernet0/2
O E2      10.30.30.3/32 [110/20] via 10.1.3.2, 00:04:14, GigabitEthernet0/3
r1
{'0/2': [{'next-hop': '10.1.2.2', 'prefix': '10.30.30.2/32'}],
 '0/3': [{'next-hop': '10.1.3.2', 'prefix': '10.2.3.0/30'},
         {'next-hop': '10.1.3.2', 'prefix': '10.11.12.0/24'},
         {'next-hop': '10.1.3.2', 'prefix': '10.30.30.3/32'}]}
```