Tabulate and Print Routes Per Interface

In this exercise, you will read IP route information from a device. You will allow a user to enter IP address prefixes and will display route information if the route was learned via OSPF.

Step 1

Connect to device R1 in the lab environment and run the `show ip route` command.

```
routes_list = [] # Create the list of routes


#-----------------------------------------------------------
# The following code connects to a device and dumps all
# routing information

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 'Timout or unexpected reply from device'
    exit()

# Successfully got password prompt, logging in with username
session.sendline('cisco')
result = session.expect('Password:')

# Successfully got password prompt, logging in with username
session.sendline('cisco')
result = session.expect('>')

# Must set terminal length to zero for long replies
print '--- setting terminal length to 0'
session.sendline('terminal length 0')
result = session.expect('>')

# Execute the 'show ip route' command to get routing info
print '--- executing: show ip route'
session.sendline('show ip route')
result = session.expect('>')
```

```
# Get output from ip route command
print '--- getting ip route command output'
show_ip_route_output = session.before

print ''
print 'IP route output'
print '-------------------------------------------------------'
print show_ip_route_output
print '-------------------------------------------------------'
print ''
```

Step 2

Using the output of the above show ip route command, create a list of destination IP address prefixes. For each destination, you will create a list of potential next hops, including next hop IP address, interface, and any other information you choose.

Answer

Note: It may be more efficient to use a dictionary here in a real-world situation, but since this module focuses on looping a list was used to show iterating through the list of routes.

```
# Get routing information into list
routes_list = show_ip_route_output.splitlines()
```

Step 3

Allow the user to input a destination IP address prefix.

Answer

```
while True: # Loop forever, until user terminates program

    # Request user to input the IP destination route prefix we will search f
    try:
        ip_address = raw_input('Enter IP destination address to find (Ctrl-C
    except KeyboardInterrupt:
        break;
```

Step 4

Go through the list of destination IP prefixes, searching for a match learned via OSPF. Print the route information if there is a match. If there is no match, print "--- Given route prefix not found ---'.

Answer

```
    # Set the pattern for matching OSPF routes
    route_pattern = re.compile('^O.{8}([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[

    # Loop through our devices looking for a match on IP address
    for route in routes_list:

        # Search for our route string, and continue to next iteration if not
        route_match = route_pattern.search(route)
        if not route_match: continue

        # Found our IP address, print out route information
        if route_match.group(1) == ip_address:
            route_info = route.split(',')
            print '  ---- Route:     ', route_info[0][5:].strip()
            print '  ---- Time:      ', route_info[1].strip()
            print '  ---- Interface: ', route_info[2].strip()
            print ''
            break

    else:  # We get here if we exhausted the device list, IP not found
        print '--- Given route prefix not found ---'

print '\n'
print 'Route search terminated.\n'

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