`if` Statement

```
if condition:
    # code for if condition is true
```

Examples:

```
# Check if the software version is current
if version == current_version:

# Check if need to check status
if last_check[device] > status_interval:

# Check if the IP address is known
if ip_address in known_ip_addresses:
```


`if` Statement (more examples)

- Tests for non-empty data structures, e.g. a list with zero items
- Returns `false` if empty, `true` otherwise

Examples:

```
# Check there are available IP addresses
if available_ip_addresses:

# Check if we have any devices in our dictionary
if device_dictionary:

# Check if our regex found the desired version pattern
if version_pattern.search(version_output):
```

A simple `if` statement involves only the `if` itself, followed by the condition, followed by a code block that will get executed if the condition evaluates to true. Networking use cases of `if` statements include:

- Comparing if the software version of a device is the correct value. For example, equal to the current version on which all devices should be running.

```
# Check if the software version is current
if version == current_version:
```

  This example would be useful for a device software management application attempting to make sure that all devices in the network are kept up to date on the correct version of software.
- Comparing if the last time a device status was checked is greater than the configured status interval.

```
# Check if need to check status
if last_check[device] > status_interval:
```

  This example would be useful for a network management application that attempts to maintain reasonably current status for all devices in the network.

- Comparing if the current IP address is valid.

```
# Check if the IP address is known
if ip_address in known_ip_addresses:
```

  This example is a membership test which checks the IP address against the set 'known_ip_addresses'. This type of test would be useful in an IP address diagnostic application, where an event has detected some type of issue with a particular IP address, and the application is looking to see if there is more information related to this IP address (for example, the user who is currently associated with this IP address).

This type of test may also be useful in an IP address management application, where IP addresses would be tested against a pool of unused IP addresses.

With Python, it is also possible to test against whether the data structure is empty or not. The following examples show testing for empty:

- Checking the list of available IP addresses, to see if anything is left in the pool.

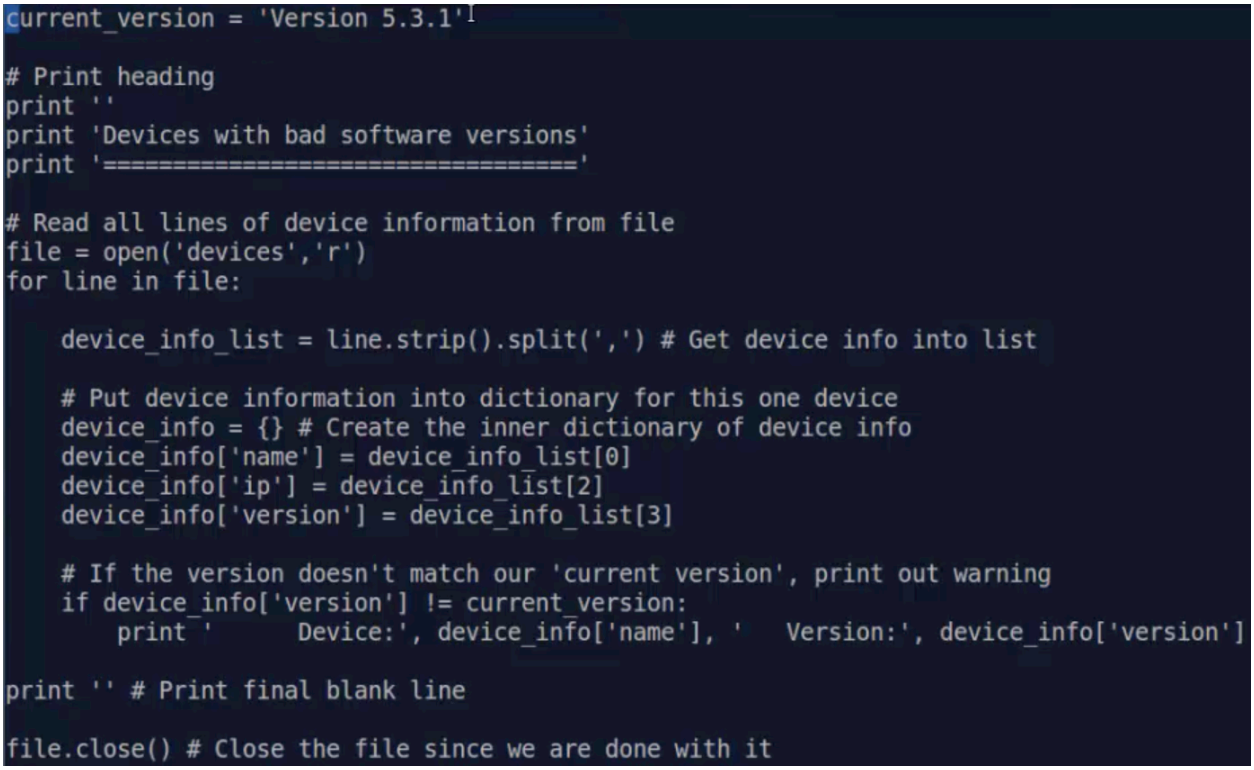
```
    # Check there are available IP addresses
    if available_ip_addresses:
```

- Checking to see if there are any devices in the dictionary which could be useful when determining if a discovery process has yielded any positive results.

```
    # Check if there are any devices in the dictionary
    if device_dictionary:
```

- Checking to see if a regular expression searching for a version string has been successful or not.

```
    # Check if regex found the desired version pattern
    if version_pattern.search(version_output):
```



```
current_version = 'Version 5.3.1'

# Print heading
print ''
print 'Devices with bad software versions'
print '================================='

# 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

    # Put device information into dictionary for this one device
    device_info = {} # Create the inner dictionary of device info
    device_info['name'] = device_info_list[0]
    device_info['ip'] = device_info_list[2]
    device_info['version'] = device_info_list[3]

    # If the version doesn't match our 'current version', print out warning
    if device_info['version'] != current_version:
        print '      Device:', device_info['name'], '   Version:', device_info['version']

print '' # Print final blank line

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

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section08$ python S04-1-bad-version.py

Devices with bad software versions
==================================
      Device: d02-is    Version: Version 4.22.18
      Device: d05-xr    Version: Version 4.16.9
      Device: d06-xr    Version: Version 5.3.0
      Device: d07-xe    Version: Version 4.16.0
      Device: d08-xe    Version: Version 5.3.0
```