Overview

`if` ✓	`else`	`elif`
• Normal `if`	• Normal `else`	• Simplified `else-if`
• Inline `if`	• Nested `else`	
• Nested `if`		
• Multiple comparisons		
• Using `if` in comprehensions		

```
if admin_state == 'up':
   # take appropriate link action
elif admin_state == 'down':
   # ignore link
else:
   # log message link in unknown state
```

When creating networking scripts, you will find that you almost always use comparisons such as >, <, == and regular expressions as part of a conditional statement such as IF-THEN-ELSE. When you are comparing two pieces of information, such as the version of IOS running on a router with a target version of IOS, you are typically making the comparison for the sole purpose of taking different actions based on the results of the comparison.

The primary conditional code statement in Python is the `if` statement.

There are three possible components of an `if` statement:

- `if`: Evaluates a comparison operation and if true, executes the associated code block.
- `else`: If the previous `if` comparison returned false, executes the associated `else` code block.
- `elif`: Combines an `else` statement with a subsequent `if` statement, which includes a new comparison associated with this new `if` clause.

If you are writing a script that checks the running version of IOS on all routers in the network, you may want the script to provide a list of routers that are running an IOS version that matches the one you specify.

```
if running_version == target_version:
# alert user that this router is running the version of  IOS we are looking for

else:
# the versions do not match, so check the next router
```

If you are trying to determine if the version of IOS running on a router, 15.4(2), is older than the version you are looking for, 15.5, you will need to perform a multi-step, complex comparison by chaining multiple IF statements together.

```
target_version_major = 15
target_version_minor = 5
running_version_major = 15
running_version_minor = 4

if running_version_major < target_version_major:
    # alert the user that this router is running an older IOS version

elif running_version_major > target_version_major:
    # do nothing since router is a running a newer version

elif running_version_major == target_version_major:
    # we need to compare the minor versions
```

If the major version numbers are the same, you will likely want to make use of a nested `if` statement by continuing the script with the following code:

```
    if running_version_minor < target_version_minor:
        # alert the user that this router is running an older IOS version

    elif running_version_minor > target_version_minor:
        # do nothing since this router is running a newer version

    else:
        # do nothing because this router must be running the same version
```

You will notice that the last statement used an `else` conditional. If the first two comparisons returned false, the two minor version numbers must match exactly.

Clean up the output from “show ip routes” and calc the number of routes per interface:

```
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])')

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

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

    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
    else:
        continue

print ''
print 'Number of routes per interface'
print '------------------------------'
pprint(routes)
```

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