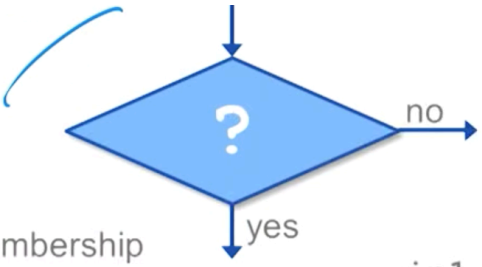
Introduction



Comparison operators

- Equality, Inequality, Membership
- Operators: `==`, `!=`, `<`, `<=`, `>`, `>=`, `in`

Comparison data

- Simple: numbers, strings
- Complex: lists, dictionaries, tuples, etc.

```
ip1      ==      ip2
10.3.21.5  <->  10.3.21.5
```

```
ip_list_1   <   ip_list_2
10.3.21.5  <->  10.3.21.5
10.3.21.6  <->  10.3.21.6
10.3.21.3  <->  10.3.21.7
```

Regular Expressions

Regular Expressions:

- Matching CLI output to extract relevant parts.
- Set pattern, perform match, group results

```
r1>show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mo
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF i
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA extern
       o - ODR, P - periodic downloaded static route, H - NH
       a - application route
       + - replicated route, % - next hop override

Gateway of last resort is 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
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, 03:10:42, GigabitEt
                     [110/2] via 10.1.2.2, 01:44:38, GigabitEt
O E2     10.11.12.0/24 [110/20] via 10.1.3.2, 03:10:42, Gigabi
                       [110/20] via 10.1.2.2, 01:44:38, Gigabi
C        10.30.30.1/32 is directly connected, Loopback0
```

With network scripting, it is common to want to compare two pieces of information. For example, you may create a script that checks the version of IOS software running on every router in the network and generates a list of routers running a specific IOS version. This script could be used to quickly identify all the routers running a specific version of IOS so they can be scheduled for upgrades.

To accomplish this task, your script will log in to each router, issue a `show version` command, and process the output.

```
r1#show version
Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.4(2)T1, REl
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2014 by Cisco Systems, Inc.
Compiled Thu 26-Jun-14 15:58 by prod_rel_team
```

The first thing your script needs to do is process each line of the `show version` output to find the version number, which in this example is 15.4(2)T1. In order to find this string of characters within the output, you will use a type of comparison match called a regular expression. Regular expressions are used to match a specific pattern of characters in a string of characters. The code to do this match would be:

```
version_pattern = re.compile('Version ([0-9]*.[0-9]*\([0-9]\))')
version = version_pattern.search(version_output)
running_version = version.group(1)
```

Once you have extracted the version number and stored it in the variable running_version, you will need to compare it against the IOS version number you are looking for. For this, you'll use a simple comparison match. If the IOS version number you're looking for is stored in the variable target_version and the IOS version number of the router you just issued the 'show version' command on is stored in the variable running_version, the simple comparison would be:

```
if target_version == running_version
```

The == symbol compares the two variables as strings and returns a value of true if they match.

In this lesson, you will learn about the various ways that Python objects can be compared and evaluated against each other.

Python provides the traditional comparison operators that are used in many different contexts, including equal ('=='), not equal ('!='), less than ('<'), greater than ('>'), less than or equal to ('<='), greater than or equal to ('>=').

You can perform arithmetic comparisons on numbers using the comparison operators ==, !=, <, >, <=, and >=. String comparisons are performed lexicographically, basically meaning in dictionary or alphabetical order.

Python also allows for testing of membership, using the `in` operator. The most obvious example would be for testing membership in a set. Less obvious is the fact that you can also test for membership in a list, tuple, or as a key in a dictionary. For strings, `in` tests for the presence of a substring within a string.

Lists and tuples are compared item by item. Dictionaries have their keys sorted, then are compared for both key and value.

Comparisons for sets use the membership operator 'in'.

Example –

```
cisco@cisco-python:/var/local/PyNE/labs/sections/section08$ cat devices
d01-is,ios,Mgmt:10.3.21.5,Version 5.3.1,cisco,cisco
d02-is,ios,Mgmt:10.3.21.6,Version 4.22.18,cisco,cisco
d03-nx,nx-os,Mgmt:10.3.21.7,Version 5.3.1,cisco,cisco
d04-nx,nx-os,Mgmt:10.3.21.8,Version 5.3.1,cisco,cisco
d05-xr,ios-xr,Mgmt:10.3.21.9,Version 4.16.9,cisco,cisco
d06-xr,ios-xr,Mgmt:10.3.21.10,Version 5.3.0,cisco,cisco
d07-xe,ios-xe,Mgmt:10.3.21.19,Version 4.16.0,cisco,cisco
d08-xe,ios-xe,Mgmt:10.3.21.22,Version 5.3.0,cisco,cisco
```

```
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
```