Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, April 19, 2016 12:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Zero-Touch ASA Upgrade using Python

Ricky

***** This article is part of my periodic mailings on Python and Network Automation. In these articles, I try to provide information that is useful to network engineers who are automating tasks in their environment. *****

***** You can also read this article here *****

Because of Cisco's recent IKE vulnerability, I have some Cisco ASAs that need upgraded. One of these ASAs is in my lab environment and I thought it would be interesting to upgrade this ASA programmatically.

This lab ASA is currently running an old operating system (*cough*, *cough*, 8.0(4)32...yes, I know it's old). In order to get started, I created a virtualenv on one of my AWS servers and then installed Netmiko 0.4.1. This AWS server has SSH access into the ASA.

Through a process of iterative testing, I wrote the following code.

Let's look at this code in some more detail:

```
def main():
    """Script to upgrade a Cisco ASA."""
    ip_addr = raw_input("Enter ASA IP address: ")
    my_pass = getpass()
    start_time = datetime.now()

    net_device = {
        'device_type': 'cisco_asa',
        'ip': ip_addr,
        'username': 'admin',
        'password': my_pass,
        'secret': my_pass,
        'port': 22,
    }
```

Here I prompt for an ip address and a password. I then create a dictionary representing the device's attributes.

This net_device dictionary is then passed into Netmiko and the SSH connection to the device is thus established (using the Netmiko ConnectHandler method). After that a few variables related to the program are initialized:

```
    print "\nLogging in to ASA"
    ssh_conn = ConnectHandler(**net_device)
    print

    dest_file_system = 'disk0:'
    source_file = 'test1.txt'
    dest_file = 'test1.txt'
    alt_dest_file = 'asa825-59-k8.bin'
    scp_changed = False
```

Note, for testing purposes I used a much smaller file 'test1.txt'. This allowed me to test and debug the program much more rapidly. I did at one point transfer 'asa825-59-k8.bin' to the ASA using Python so it is already on the ASA.

Up until this point, I have mostly just initialized things. Now, let's start doing some work:

```
    with FileTransfer(ssh_conn, source_file=source_file,
                      dest_file=dest_file,
                      file_system=dest_file_system) as scp_transfer:

        if not scp_transfer.check_file_exists():
            if not scp_transfer.verify_space_available():
                raise ValueError("Insufficient space available on remote
device")
```

Here I create a FileTransfer object named 'scp_transfer'. FileTransfer is a Netmiko class that I created to demonstrate secure copy on Cisco IOS devices. It also works, however, on Cisco ASAs.

What does the FileTranfer class do? It uses secure copy to transfer a file to the remote device. It accomplishes this by creating two channels—an SSH control channel and a SCP channel to transfer the file. Additionally, FileTransfer has methods that allow you to perform verifications. As you can see above, I use 'check_file_exists()' and 'verify_space_available()' to determine whether the file already exists and whether there is sufficient space available.

Now I am ready to do the file transfer. First, I call a small function that uses Netmiko to enable SCP on the ASA. I then transfer the file. Finally, I use the same function to disable SCP on the ASA.

```
            print "Enabling SCP"
            output = asa_scp_handler(ssh_conn, mode='enable')
            print output

            print "\nTransferring file\n"
            scp_transfer.transfer_file()

            print "Disabling SCP"
            output = asa_scp_handler(ssh_conn, mode='disable')
            print output
```

At this point, I just need to verify that the file is correct. Consequently, I call the "verify_file()" method which performs an MD5 comparison on the files.

```
        print "\nVerifying file"
        if scp_transfer.verify_file():
            print "Source and destination MD5 matches"
        else:
            raise ValueError("MD5 failure between source and destination
files")
```

I now know the file is on the ASA and that the MD5 matches. Consequently, I can configure the 'boot system' command and then verify the boot variable is correct.

After, I do this verification, I then need to save the config and reload the ASA. Once again, I use Netmiko to accomplish this.

Note, when testing this program I manually verified the boot variable before sending the 'wr mem' and 'reload' commands. In other words, the program needs additional logic that verifies the boot variable. The program also should have additional sanity checks on the remote file (to prevent against cases where you specify the wrong source file).

```
    print "\nSending boot commands"
    full_file_name = "{}/{}".format(dest_file_system, alt_dest_file)
    boot_cmd = 'boot system {}'.format(full_file_name)
    output = ssh_conn.send_config_set([boot_cmd])
    print output

    print "\nVerifying state"
    output = ssh_conn.send_command('show boot')
    print output

    print "\nWrite mem and reload"
    output = ssh_conn.send_command_expect('write mem')
    output += ssh_conn.send_command('reload')
    output += ssh_conn.send_command('y')
    print output

    print "\n>>>> {}".format(datetime.now() - start_time)
    print
```

What does this script look like when it executes:

```
Enter ASA IP address: 10.10.10.26
Password:
>>>> 2016-03-23 10:21:55.579044

Logging in to ASA

Enabling SCP
config term
```

```
twb-py-lab(config)# ssh scopy enable
twb-py-lab(config)# end
twb-py-lab#

Transferring file

Disabling SCP
config term
twb-py-lab(config)# no ssh scopy enable
twb-py-lab(config)# end
twb-py-lab#

Verifying file
Source and destination MD5 matches

Sending boot commands
config term
twb-py-lab(config)# boot system disk0:/asa825-59-k8.bin
twb-py-lab(config)# end
twb-py-lab#

Verifying state

BOOT variable =
Current BOOT variable = disk0:/asa825-59-k8.bin
CONFIG_FILE variable =
Current CONFIG_FILE variable =

Write mem and reload
Building configuration...
Cryptochecksum: feefe731 960a78a8 1c1217e0 553b57fe

8983 bytes copied in 1.310 secs (8983 bytes/sec)
[OK]Proceed with reload? [confirm] twb-py-lab#
twb-py-lab#


***
*** --- START GRACEFUL SHUTDOWN ---
Shutting down isakmp
Shutting down File system



***
*** --- SHUTDOWN NOW ---


>>>> 0:00:23.222710
```

You can see above that the code took a bit over 23 seconds to execute. As I mentioned earlier this execution only transferred a small text file and didn't actually transfer the ASA image file (which was transferred earlier).

After the reload the ASA was running the new OS. I verified this and performed a before-after diff on the config using some other tools.

Now just for grins let's see how long it takes to secure copy the actual ASA image file. Here, I modify the following variables:

```
    #source_file = 'test1.txt'
    source_file = 'asa825-59-k8.bin'
    dest_file = 'asa-newimage.bin'
```

I also commented out the code pertaining to configuring 'boot system' and executing 'write mem' and 'reload' (as the ASA is already running the new image).

```
Enter ASA IP address: 50.76.53.26
Password:
>>>> 2016-03-23 15:34:30.179171

Logging in to ASA

Enabling SCP
config term
twb-py-lab(config)# ssh scopy enable
twb-py-lab(config)# end
twb-py-lab#

Transferring file

Disabling SCP
config term
twb-py-lab(config)# no ssh scopy enable
twb-py-lab(config)# end
twb-py-lab#

Verifying file
Source and destination MD5 matches

>>>> 0:08:59.881756
```

As you can see the transfer took about nine minutes and the file is now on the device:

```
twb-py-lab# dir disk0:/asa-newimage.bin

Directory of disk0:/asa-newimage.bin

135    -rwx  15482880    14:55:23 Mar 23 2016  asa-newimage.bin
```

The ASA is obviously very slow to transfer a 15MB file, but this is an ASA problem (i.e. it would also be slow if I transferred the file manually).

Because of this slowness, however, I would probably decouple the file transfer operation from the final verifications, boot system, and reload. In other words, I would split the program into two separate parts, one part that handled the initial image transfer, and the second part that completed the upgrade including the reload.

Regards,
Kirk Byers
https://pynet.twb-tech.com

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, April 12, 2016 12:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Ansible line-by-line network config changes

Ricky

***** This article is part of my periodic mailings on Python and Network Automation. In these articles, I try to provide information that is useful to network engineers who are automating tasks in their environment. *****

Ansible is adding a set of new networking modules into their core. These modules are currently in the Ansible 'devel' branch and will be released as part of Ansible 2.1.

These new networking modules support a set of platforms: Junos, Arista EOS, Cisco Nexus, Cisco IOS-XR, and Cisco IOS. For each platform, there are three distinct modules: _command, _config, and _template. Consequently, Junos has a junos_config, junos_command, and junos_template module. Likewise, Cisco IOS has an ios_config, ios_command, and ios_template module.

At a high-level the three module types allow you to accomplish the following:

- **command** - execute a command(s) and retrieve the output from a remote device. This information could be used to make decisions in the playbook.
- **config** - execute a config change on the remote device.
- **template** - push config changes to the remote device generally from a file or from a template.

In this article, I am going to detail capabilities provided by the new "config" modules. These modules provide a valuable Ansible-feature to network engineers namely line-by-line configuration editing with an understanding of config hierarchy.

The other two module types (command and template) are not all that significant in my opinion. In particular, the functionality provided by the template module is better served by NAPALM (possibly coupled with Ansible's standard template module). NAPALM provides a better set of abstractions for performing config file operations on networking devices (disclaimer, I have made meaningful contributions to the NAPALM project).

In my lab environment, I have a Cisco IOS router. I also have an AWS server with Ansible2 installed (this was from the Ansible 'devel' branch as of March 2016).

My Ansible inventory has the following:

```
[local]
localhost ansible_connection=local

[cisco]
pynet-rtr1 host=10.10.10.27 port=22 username=admin password=pwd

[cisco:vars]
ansible_python_interpreter=~/VENV/ansible2/bin/python
ansible_connection=local
```

Since I am using Cisco IOS, I will be using the ios_config module. Here is the initial playbook that I started with:

```
---
- name: Test Cisco IOS line-by-line editing
  gather_facts: no
  hosts: cisco

  vars:
    creds:
        host: "{{host}}"
        username: "{{username}}"
        password: "{{password}}"

  tasks:
    - ios_config:
        provider: "{{creds}}"
        lines:
           - logging buffered 19000
           - no logging console
        match: line
```

This playbook will operate on the 'cisco' group which as defined in the inventory file and is a single Cisco IOS router (pynet-rtr1). The 'vars' section of the playbook defines a dictionary named 'creds' which basically consolidates the ip address (host), username, and password in a single variable. This information is pulled from the Ansible inventory file.

In the next section, I use the ios_config module with the provider, lines, and match arguments. Provider allows the ip address, username, and password to be passed in as a single variable. The lines argument specifies the configuration elements that must exist on the device.

Match is the final argument. It allows three possible values--'line', 'strict', and 'exact'. Line is the most basic matching. Basically these lines must exist in the config and the order doesn't matter. Note, there is the possibility of qualifying this with a parent which I will discuss below.

What does the above playbook do when executed? Basically, Ansible will SSH to the network device, retrieve the current running configuration, and check whether the two configuration lines exist. If they aren't in the configuration, they will be added. If they are in the config, then no changes will be made. Note, there is a bit more subtlety depending on the parent argument which I will detail shortly.

Currently the router has the following config:

```
pynet-rtr1#show run | inc logging
logging buffered 20000
no logging console
pynet-rtr1#
```

Now, I execute the playbook:

```
$ ansible-playbook -i ./ansible-hosts ios_test_config.yml

PLAY [Test Cisco IOS line-by-line editing] **************************

TASK [ios_config] ***************************************************
changed: [pynet-rtr1]

PLAY RECAP **********************************************************
pynet-rtr1         : ok=1    changed=1    unreachable=0    failed=0
```

And back on the router I now have:

```
pynet-rtr1#show run | inc logging
logging buffered 19000
no logging console
pynet-rtr1#
```

If I run the playbook again, Ansible detects that no changes need to be made.

Now let's do something a bit more interesting and try to add a VLAN interface to the router. I update the playbook and append the following to the end of it (the initial task is still present).

```
    - name: Add interface vlan127 to the config
      ios_config:
        provider: "{{creds}}"
        lines:
          - ip address 192.168.127.1 255.255.255.0
          - no ip proxy-arp
        parents: ['interface Vlan127']
        match: line
```

Here I have added a new argument 'parents'. Parents specifies a configuration context that the lines must exist inside of. In other words, the two lines must exist inside of the interface Vlan127 section. If no parents are specified, then the configuration context is the global config.

At this point, interface Vlan127 doesn't exist on the router:

```
pynet-rtr1#show ip int brief
Interface          IP-Address      OK? Method Status       Protocol
```

```
FastEthernet0     unassigned      YES unset   down          down
FastEthernet1     unassigned      YES unset   down          down
FastEthernet2     unassigned      YES unset   down          down
FastEthernet3     unassigned      YES unset   down          down
FastEthernet4     10.220.88.20    YES NVRAM   up            up
Vlan1             unassigned      YES unset   down          down
```

Now I run the playbook and observe the following:

```
$ ansible-playbook -i ./ansible-hosts ios_test_config.yml

PLAY [Test Cisco IOS line-by-line editing] *************************

TASK [ios_config] **************************************************
ok: [pynet-rtr1]

TASK [Add interface vlan127 to the config] *************************
changed: [pynet-rtr1]

PLAY RECAP *********************************************************
pynet-rtr1        : ok=2    changed=1    unreachable=0    failed=0
```

And back on the router:

```
pynet-rtr1#show ip int brief | inc 127
Vlan127         192.168.127.1   YES manual down   down

pynet-rtr1#show run int vlan 127
Building configuration...
Current configuration : 82 bytes
!
interface Vlan127
 ip address 192.168.127.1 255.255.255.0
 no ip proxy-arp
end
```

Now what about an ACL example where both hierarchy and order matter. First I add the following task to my playbook:

```
    - name: Add TEST1 ACL
      ios_config:
        provider: "{{creds}}"
        lines:
          - permit ip host 1.1.1.1 any log
          - permit ip host 2.2.2.2 any log
          - permit ip host 3.3.3.3 any log
          - permit ip host 4.4.4.4 any log
          - permit ip host 5.5.5.5 any log
        parents: ["ip access-list extended TEST1"]
        before: ["no ip access-list extended TEST1"]
        replace: block
        match: line
```

Here I have the ACL lines in a specific order. Additionally, I have the parent that will create the access-list. Beneath the parents argument, is a new argument named 'before'. On any change operation the 'before' command will be executed. So in this example, if Ansible determines the ACL needs to be changed (or doesn't exist), then the before command will be executed. After the before command is executed, the ACL will be added.

Note, executing 'no ip access-list extended TEST1' does not generate an error (in the case where the ACL does not exist). If your 'before' command generated an error (when the element doesn't exist), then you would to add some additional logic to your playbook.

A couple of additional items to note about this playbook, 'replace: block' indicates to replace the entire configuration block (all the lines including the parents). In other words, do not try to determine individual lines that are missing from the ACL, always do all of the lines. Since the 'before' command removes the entire ACL, not doing 'replace: block' could result in a partial ACL being configured. Finally, I have once again specified 'match: line'. So for this task, Ansible will check that all of the ACL entries exist, but it will not care about their order.

```
$ ansible-playbook -i ./ansible-hosts ios_test_config.yml

PLAY [Test Cisco IOS line-by-line editing] **************************

TASK [ios_config] ***************************************************
ok: [pynet-rtr1]

TASK [Add interface vlan127 to the config] **************************
ok: [pynet-rtr1]

TASK [Add TEST1 ACL] ************************************************
changed: [pynet-rtr1]

PLAY RECAP **********************************************************
pynet-rtr1          :
ok=3    changed=1    unreachable=0    failed=0
```

And now I have the ACL on the router:

```
pynet-rtr1#show access-lists TEST1
Extended IP access list TEST1
    10 permit ip host 1.1.1.1 any log
    20 permit ip host 2.2.2.2 any log
    30 permit ip host 3.3.3.3 any log
    40 permit ip host 4.4.4.4 any log
    50 permit ip host 5.5.5.5 any log
```

Let's make some small changes and see how 'match: exact' works. First, I am going to keep the same five ACL lines but reorder them.

I have manually reconfigured the TEST1 ACL to be the following:

```
pynet-rtr1#show access-lists TEST1
Extended IP access list TEST1
    10 permit ip host 5.5.5.5 any log
    20 permit ip host 2.2.2.2 any log
    30 permit ip host 3.3.3.3 any log
    40 permit ip host 4.4.4.4 any log
    50 permit ip host 1.1.1.1 any log
```

Re-running the exact same playbook--Ansible thinks nothing needs to change. This is because Ansible checks that the parent exists. It also checks all the children, but with 'match: line' Ansible doesn't care about the ACL order.

If we want to enforce the ACL order, then we need to use either 'match: strict' or 'match: exact'. Match strict will enforce the order of the elements, but it will not change the ACL in subset/superset situations. In our example, 'match: strict' would indicate the ACL was correct if TEST1 had the five lines in the correct order even if the ACL on the router had an additional ten lines.

Match exact on the other hand enforces not only order, but also that the elements specified are the only elements in the given context.

Note, there is a bug in the 'match: strict' code pertaining to the subset/superset behavior. Consequently, I am going to demonstrate 'match: exact' here. Additionally, I did not test the behavior of a multi-level hierarchy. In other words, how would children of children be handled.

In the playbook, I toggle the match condition to be match: exact and then re-execute the playbook.

Now Ansible makes a change and puts the correct ACL in place:

```
$ ansible-playbook -i ./ansible-hosts ios_test_config.yml

PLAY [Test Cisco IOS line-by-line editing] **************************

TASK [ios_config] ***************************************************
ok: [pynet-rtr1]

TASK [Add interface vlan127 to the config] **************************
ok: [pynet-rtr1]

TASK [Add TEST1 ACL] ************************************************
changed: [pynet-rtr1]

PLAY RECAP **********************************************************
pynet-rtr1          :
ok=3    changed=1    unreachable=0    failed=0
```

And the router now has:

```
pynet-rtr1#show access-lists TEST1
Extended IP access list TEST1
    10 permit ip host 1.1.1.1 any log
    20 permit ip host 2.2.2.2 any log
    30 permit ip host 3.3.3.3 any log
    40 permit ip host 4.4.4.4 any log
    50 permit ip host 5.5.5.5 any log
```

And for one last test, let me manually expand the TEST1 ACL on the router to have some additional lines:

```
pynet-rtr1#show access-lists TEST1
Extended IP access list TEST1
    10 permit ip host 1.1.1.1 any log
    20 permit ip host 2.2.2.2 any log
    30 permit ip host 3.3.3.3 any log
    40 permit ip host 4.4.4.4 any log
    50 permit ip host 5.5.5.5 any log
    60 permit ip host 6.6.6.6 any log
    70 permit ip host 7.7.7.7 any log
```

Now, let me re-run the 'match: exact' playbook (with the five ACL lines) and see what happens:

```
$ ansible-playbook -i ./ansible-hosts ios_test_config.yml

PLAY [Test Cisco IOS line-by-line editing] **************************

TASK [ios_config] ***************************************************
ok: [pynet-rtr1]

TASK [Add interface vlan127 to the config] **************************
ok: [pynet-rtr1]

TASK [Add TEST1 ACL] ************************************************
changed: [pynet-rtr1]

PLAY RECAP **********************************************************
pynet-rtr1          : ok=3    changed=1    unreachable=0    failed=0


pynet-rtr1#show access-lists TEST1

Extended IP access list TEST1
    10 permit ip host 1.1.1.1 any log
    20 permit ip host 2.2.2.2 any log
    30 permit ip host 3.3.3.3 any log
    40 permit ip host 4.4.4.4 any log
    50 permit ip host 5.5.5.5 any log
```

As expected the ACL is restored back to the five-line state.

Regards,

Kirk Byers
ktbyers@twb-tech.com

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, April 5, 2016 12:00 PM
To:	Laney, Ricky A
Subject:	[PyNet] - NAPALM, Ansible, and Cisco IOS

Ricky

****** Starting on April 7, I will be running a paid course on Network Automation using Python and Ansible. See https://pynet.twb-tech.com/class.html for more information.****

David Barroso and Elisa Jasinska recently created a library called NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support). The general idea behind this library is to create a standardized, multivendor interface for certain file and get operations. Last fall, Gabriele Gerbino added Cisco IOS support to NAPALM.

Independent of NAPALM, I have been thinking about and experimenting with programmatic file operations using Cisco IOS. I wrote a proof of concept related to this here. Consequently, I thought it made sense to add/improve the Cisco IOS file operations in NAPALM. Because of this, I re-wrote the file methods in the NAPALM Cisco IOS driver (sorry about that Gabriele). Basically, I thought using secure copy (SCP), 'configure replace', and a file-based configuration merge (copy file system:running-config) would be more consistent with NAPALM and ultimately provide a better solution (in the context of NAPALM file operations).

So what can you do with NAPALM in the context of Cisco IOS?

- **Configuration replace:** replace the entire running-config with a completely new configuration.
- **Configuration merge:** merge a set of changes from a file into the running-config.
- **Configuration compare:** compare your new proposed configuration file with the running-config. This only applies to configuration replace operations; it does not apply to merge operations.
- **Commit:** deploy the staged configuration. This can be either an entire new file (for replace operations) or a merge file.
- **Discard:** revert the candidate configuration file back to the current running-config; reset the merge configuration file back to an empty file.
- **Rollback:** revert the running configuration back to a file that was saved prior to the previous commit.

But what happens under the hood?

Under the hood, there are three files that can potentially be used: 'candidate_config.txt', 'merge_config.txt', and 'rollback_config.txt'. The default file system is 'flash:', but this default can be overridden.

For configure replace operations, the new config file will be secure copied to 'candidate_config.txt'. Upon commit, Cisco's 'configure replace' command will be executed (see here for details).

Similarly, a merge operation will SCP a file to 'flash:/merge_config.txt'. Upon commit, a 'copy flash:merge_config.txt system:running-config' command will be executed (once again a different file system can be used and 'flash:' is not required).

Compare configuration will perform a diff between the candidate_config.txt file and the running-config using 'show archive config differences'.

Finally, the discard and rollback commands will manipulate the three files ('candidate_config.txt', 'merge_config.txt', 'rollback_config.txt'). Discard config will cause the current running-config file to be copied into candidate_config.txt. Additionally, discard config will cause the merge_config.txt file to be zeroed out. The rollback command will cause the 'rollback_config.txt' file to become the running-config (once again using Cisco's 'configure replace' command). Note, the current running-config is saved to rollback_config.txt on any commit operation.

For additional details including examples, see:

https://pynet.twb-tech.com/blog/automation/napalm-ios.html

Regards,

Kirk Byers
ktbyers@twb-tech.com
1235 Connecticut St
San Francisco CA 94107
USA

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, April 5, 2016 10:06 AM
To:	Laney, Ricky A
Subject:	[PyNet] - Python + Ansible course starts this Thursday (4/7)

Ricky

One last reminder, I am running a paid network automation course starting this Thursday.

The sign-up page for the course will close on Thursday at 10AM.

You can find more details about the course here:

https://pynet.twb-tech.com/class.html

Regards,

Kirk Byers
ktbyers@twb-tech.com

1235 Connecticut St
San Francisco CA 94107
USA

Laney, Ricky A

From:	pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Thursday, March 31, 2016 9:15 AM
To:	Laney, Ricky A
Subject:	[PyNet] Learning Python: Week10 (Connecting to a router via SSH/telnet)

Ricky

In this class we are going to do just one exercise. This exercise will consist of connecting to a network device via SSH or telnet, executing a show command, parsing the output, and saving the information to a file. This exercise is specified in more detail below.

Beneath the exercise, there are also a series of videos on--1)using Python's SSH/telnet libraries to connect to a network device, and 2)how to save Python objects to a file using pickle.

Exercise

Reference code for these exercises is posted on GitHub at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class10

Create a script that can login to a network device (using either telnet or SSH) and retrieve 'show version' from the device.

Process the 'show version' output and store the below attributes in a NetworkDevice object:

```
NetworkObject
   hostname
   ip
   username
   password
   device_type          # router, switch, firewall, etc.
   vendor
   model
   os_version
   uptime               # seconds
   serial_number
```

This object should be stored to a file using pickle.

I am assuming that you have a test or lab network device that you can test your code against.

Videos

I. **Introduction**
video http://youtu.be/6ABYJst9XVM
Length is 2 minutes

II. **SSH to a router (Part1)**
video http://youtu.be/ZUWV0GdKueE
Length is 9 minutes

III. **SSH to a router (Part2)**
video http://youtu.be/jQhTNXIXkfQ
Length is 15 minutes

IV. **Telnet to a router**
video http://youtu.be/jb_NLzre9h8
Length is 10 minutes

V. **Pickle**
video http://youtu.be/ZJOJjyhhEvM
Length is 6 minutes

Make sure that you set your video resolution in YouTube to at least 360p. The resolution button is at the bottom of the video and looks like a little gear.

Additional Note About Paramiko:

Paramiko, by default, will try to use your SSH keys to connect to the network device. This can cause problems as the username and password that you specify will not be used (if SSH keys exist). Similarly, the existence of an SSH agent can cause Paramiko connection problems. Consequently, you should add the following two arguments to your Paramiko connect() statement:

```
+    remote_conn_pre.connect(ip, username=username, password=password,
+                            look_for_keys=False, allow_agent=False)
```

Installing Paramiko:

Note: my Paramiko SSH videos were created and tested using Paramiko 1.7.7.1 (which is old). I also tested the script (that was created during the video) using Paramiko 1.14.0.

If you have issues getting Paramiko installed, which is entirely possible, feel free to email me.

Linux:

http://paramiko-www.readthedocs.org/installing.html

Note: you might also have to install PIP:
https://pip.pypa.io/en/latest/installing.html#install-pip

MacOS:

For MacOS, I used Homebrew and installed a separate version of Python onto my system. See the following document for details:
http://docs.python-guide.org/en/latest/starting/install/osx/

Note, I did the following before installing the new Python:

```
brew doctor
brew update
brew install python
```

After Homebrew and the new Python are installed, I did the following:

```
pip install --upgrade setuptools
pip install --upgrade pip
pip install pycrypto
pip install paramiko
```

This will separate the Python you use for development from the system Python (and also install Paramiko).

Note, there is a lot of context here regarding your environment that I don't know (which packages you already have installed, whether you want to use the system Python or a separate Python, which version of MacOS you are using). The above worked well for me on MacOS Yosemite.

Windows:

Download and install the right PyCrypto executable for your machine:
http://www.voidspace.org.uk/python/modules.shtml#pycrypto

Download and install pip:
https://pip.pypa.io/en/latest/installing.html#install-pip

Then from the command prompt:

```
> pip install paramiko
> pip install ecdsa
```

Note, if you are using Windows and 64-bit Python, you might run into an import issue with winrandom.

Additional content that you may be interested in:

My blog post on "Python, Paramiko SSH, and Network Devices":
https://pynet.twb-tech.com/blog/python/paramiko-ssh-part1.html

Pexpect is a Python library that creates an interaction somewhat similar to using Expect. I am not a big fan of either Expect or Pexpect so you will have to decide if this is valuable to you.
http://pexpect.readthedocs.org/en/latest/overview.html

Scapy is an interesting library for creating packets of various types. "Introduction to Scapy" by Jeremy Stretch:
http://packetlife.net/blog/2011/may/23/introduction-scapy/

TABLE OF CONTENTS

    F. Disable paging on the router

IV. Pickle
    A. What is pickle?
    B. Using pickle to write objects to a file
    C. Using pickle to read objects from a file

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops
Passing Arguments into a Script

Week4
Introduction
While Loops
Dictionaries
Exceptions

Week5
Class Review (weeks 1 - 4)

Week6
Introduction
Functions, Part1
Namespaces
Functions, Part2

Week7
Files
Regular Expressions

Week8

Modules - Part1
Modules - Part2
Packages

Week9
Introduction
Classes and Objects - Part1
Classes and Objects - Part2

Week10
Introduction
SSH to a router - Part1
SSH to a router - Part2
Telnet to a router
Pickle

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, March 29, 2016 12:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Using Python's ipaddress Library

Ricky

**** Starting on April 7, I will be running a paid course on Network Automation using Python and Ansible. See https://pynet.twb-tech.com/class.html for more information.****

Python has numerous libraries pertaining to IP addresses. PEP3144 (Python Enhancement Proposals) creates a standardized IP address library. This PEP has been implemented in the ipaddress library which is integrated into Python as of Python 3.3 and has been back-ported to Python 2.6, 2.7, and 3.2)

What follows are some useful things you can do with the ipaddress Library.

First, you can create both IPv4Address and IPv6Address objects.

```
>>> import ipaddress
>>> alt_ip = ipaddress.ip_address(u'10.220.7.193')
>>> alt_ip
IPv4Address(u'10.220.7.193')
>>> str(alt_ip)
'10.220.7.193'

>>> myipv6 = ipaddress.ip_address(u'ff05::1:3')
>>> str(myipv6)
'ff05::1:3'
>>> myipv6.exploded
u'ff05:0000:0000:0000:0000:0000:0001:0003'
>>> myipv6
IPv6Address(u'ff05::1:3')
```

The ip_address() call is a factory function that determines the proper class to use based on the input you provide (you can also directly create an object using ipaddress.IPv4Address).

Note, when creating IPv4Address/IPv6Address objects, a unicode string is required.

There really isn't all that much you can do with just IPv4Address/IPv6Address objects. Of more value are the IPv4Network and IPv6Network classes:

```
>>> my_net = ipaddress.ip_network(u'10.220.192.192/29')
>>> my_net
```

```
IPv4Network(u'10.220.192.192/29')

>>> ipv6_net = ipaddress.ip_network(u'fe80::0202:b3ff:fe1e:8329/64', strict=False)
>>> ipv6_net
IPv6Network(u'fe80::/64')
```

By default both the IPv4Network class and the IPv6Network class require the host portion of the address to be all zeroes. You can change this behavior by setting the 'strict=False' flag as I did with above IPv6 address. The 'strict=False' flag will cause the class to accept the network, but it will still zero out the host component. If you want to support both a network component and a host component, see the IPv4Interface/IPv6Interface classes (described below).

There are several alternate forms that you can use to create an IPv4Network object:

```
>>> my_net = ipaddress.ip_network(u'10.220.192.192/255.255.255.248')
>>> my_net
IPv4Network(u'10.220.192.192/29')
>>> my_net = ipaddress.ip_network(u'10.220.192.192/0.0.0.7')
>>> my_net
IPv4Network(u'10.220.192.192/29')
```

There are also several useful attributes on IPv4Network objects:

```
>>> my_net.network_address
IPv4Address(u'10.220.192.192')
>>> my_net.broadcast_address
IPv4Address(u'10.220.192.199')

>>> my_net.hostmask
IPv4Address(u'0.0.0.7')
>>> my_net.netmask
IPv4Address(u'255.255.255.248')

>>> my_net.with_netmask
u'10.220.192.192/255.255.255.248'
>>> my_net.with_hostmask
u'10.220.192.192/0.0.0.7'

>>> my_net.with_prefixlen
u'10.220.192.192/29'
>>> my_net.prefixlen
29

>>> my_net.num_addresses
8
```

You can iterate over the hosts of a network (this excludes the network number and the broadcast address):

```
>>> for test_ip in my_net.hosts():
...     print test_ip
...
10.220.192.193
```

```
10.220.192.194
10.220.192.195
10.220.192.196
10.220.192.197
10.220.192.198
```

You can find the subnets for a given network (fixed length subnets):

```
>>> my_net = ipaddress.ip_network(u'10.220.2.0/24')
>>> my_net
IPv4Network(u'10.220.2.0/24')
>>> for subnet in my_net.subnets(new_prefix=27):
...     print subnet
...
10.220.2.0/27
10.220.2.32/27
10.220.2.64/27
10.220.2.96/27
10.220.2.128/27
10.220.2.160/27
10.220.2.192/27
10.220.2.224/27
```

You can also find a supernet for a given network:

```
>>> ipaddress.ip_network(u'10.220.167.0/24').supernet(new_prefix=20)
IPv4Network(u'10.220.160.0/20')
```

Finally, the ipaddress library includes both IPv4Interface and IPv6Interface classes. These classes allow both a host component and a network component in one object:

```
>>> my_ip = ipaddress.ip_interface(u'10.220.192.194/29')
>>> my_ip
IPv4Interface(u'10.220.192.194/29')

>>> my_ipv6 = ipaddress.ip_interface(u'fe80::0202:b3ff:fe1e:8329/64')
>>> my_ipv6
IPv6Interface(u'fe80::202:b3ff:fe1e:8329/64')
```

From this you can obtain both the IP address and the IP network:

```
>>> my_ip.ip
IPv4Address(u'10.220.192.194')
>>> my_ip.network
IPv4Network(u'10.220.192.192/29')
```

The ipaddress library (while by no means earth shattering) has some useful features that could be used for ACL construction, ping sweeping, minor input validation, network-subnet determination, etc. Why re-invent the wheel when an okay one already exists.

Regards,

Kirk

Laney, Ricky A

From: pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Saturday, March 26, 2016 4:17 AM
To: Laney, Ricky A
Subject: [PyNet] Learning Python: Week9 (Classes and Objects):

Ricky

In this email of Learning Python we are going to cover the following:

I. **Introduction**
   video http://youtu.be/69w7HO5cpvE
   Length is 2 minutes

II. **Classes and Objects (Part1)**
   video http://youtu.be/TAACEw69jBw
   Length is 14 minutes

III. **Classes and Objects (Part2)**
   video http://youtu.be/h8jlJ1kb_mY
   Length is 23 minutes

Make sure that you set your video resolution in YouTube to at least 360p. The resolution button is at the bottom of the video and looks like a little gear.

Additional content that you may be interested in:

Understanding object-oriented programming can be hard (probably harder than it should be). The terminology is new and can be difficult to understand. Also, a lot of the Internet articles on Python object-oriented programming are not very good for beginners. They can presuppose too much background knowledge, not be precise enough, not understand the context of a beginner.

Because of the above, I had to search around to find good articles on Python and object-oriented programming.

If you are interested in understanding object-oriented programming at least at a beginner level, I recommend that you read both of the below two links and experiment with creating classes, objects, and methods.

Learn Python the Hard Way, Exercise 40: Modules, Classes, and Objects:

http://learnpythonthehardway.org/book/ex40.html

Introduction to OOP with Python (this is very good):

http://www.voidspace.org.uk/python/articles/OOP.shtml

Exercises

Reference code for these exercises is posted on GitHub at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class9

I. Create a Python class representing an IPAddress. The class should have only one initialization variable--an IP address in dotted decimal formation. You should be able to do the following with your class:

```
>>> test_ip = IPAddress('192.168.1.1')
>>> test_ip.ip_addr
'192.168.1.1'
```

A. Write a method for this class that returns the IP address in dotted binary format (each octet should be eight binary digits in length).

```
>>> test_ip.display_in_binary()
'11000000.10101000.00000001.00000001'
```

B. Write a method for this class that returns the IP address in dotted hex format (each octet should be two hex digits in length).

```
>>> test_ip.display_in_hex()
'c0.a8.01.01'
```

C. Re-using the IP address validation code from class8, exercise1--create an is_valid() method that returns either True or False depending on whether the IP address is valid.

```
>>> test_ip.is_valid()
True
```

II. Create an Uptime class that takes in a Cisco IOS uptime string and parses the string into years, weeks, days, hours, minutes (assigning these as attributes of the object). For example:

```
>>> test_obj = Uptime('twb-sf-881 uptime is 6 weeks, 4 days, 2 hours, 25 minutes')
>>> test_obj.years
0
>>> test_obj.weeks
6
>>> test_obj.days
4
>>> test_obj.hours
2
>>> test_obj.minutes
25
```

You class should be able to process the following four uptime strings:

```
'twb-sf-881 uptime is 6 weeks, 4 days, 2 hours, 25 minutes'
'3750RJ uptime is 1 hour, 29 minutes'
'CATS3560 uptime is 8 weeks, 4 days, 18 hours, 16 minutes'
'rtr1 uptime is 5 years, 18 weeks, 8 hours, 23 minutes'
```

A. Create a method for this class that returns the uptime in seconds.

```
>>> test_obj.uptime_seconds()
3983100
```

III. Write a new class, IPAddressWithNetmask, that is based upon the IPAddress class created in exercise1 (i.e. use inheritance). The netmask should be stored in slash notation (i.e. /24). With this class you should be able to do the following:

```
>>> test_ip2 = IPAddressWithNetmask('172.31.255.1/24')
>>> test_ip2.ip_addr
'172.31.255.1'
>>> test_ip2.netmask
'/24'
```

If possible, the ip_addr attribute should be assigned via the __init__ method in the IPAddress Class (in other words, call the __init__ method of the base IPAddress class with '172.31.255.1' as an argument).

All of the other methods from the IPAddress class should be left unchanged.

A. Create a new method in the IPAddressWithNetmask class that converts the slash notation netmask to dotted decimal.

B. You should be able to do the following when you are done:

```
>>> test_ip2 = IPAddressWithNetmask('172.31.255.1/24')
>>> test_ip2.ip_addr
'172.31.255.1'
>>> test_ip2.netmask
'/24'

>>> test_ip2.display_in_binary()
'10101100.00011111.11111111.00000001'
>>> test_ip2.display_in_hex()
'ac.1f.ff.01'
>>> test_ip2.is_valid()
True

>>> test_ip2.netmask_in_dotdecimal()
'255.255.255.0'
```

TABLE OF CONTENTS

    2. First parameter is always the object itself (by convention named 'self')
    3. You call methods by using object_name.method_name()
  B. Inheritance
    1. Simple inheritance
    2. What happens with __init__()
    3. Over-writing the __init__() method in the subclass
    4. Calling the parent's __init__() method
    5. Over-writing a method in the subclass.

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops
Passing Arguments into a Script

Week4
Introduction
While Loops
Dictionaries
Exceptions

Week5
Class Review (weeks 1 - 4)

Week6
Introduction
Functions, Part1
Namespaces
Functions, Part2

Week7

Files
Regular Expressions

Week8
Modules - Part1
Modules - Part2
Packages

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, March 22, 2016 12:00 PM
To:	Laney, Ricky A
Subject:	[PyNet] -- Automating the Console using pySerial

Ricky

**** Starting on April 7, I will be running a paid course on Network Automation using Python and Ansible. See https://pynet.twb-tech.com/class.html for more information.****

So you have a brand new router and you want to fully automate the configuration process?

Now once the device is on the network, then you can use SSH or an API to programmatically configure it.

But you still potentially need to minimally configure the device to safely add it onto the network. For example, disable the DHCP server, configure SSH, add a user, etc. The specifics will depend on the device.

There are obviously "zero touch" ways to accomplish this like Cisco's POAP or Arista's ZTP.

But for a quick and easy solution--why can't you just programmatically configure the device via the console connection?

You can using pySerial.

Let's do some experimentation with this.

First, I am working on an old Windows machine that has Python 2.7.6 installed. I also installed pySerial on this machine.

The first thing I need to do is establish a serial connection. After some experimentation I was able to do the following:

```
>>> import serial
>>>
>>> console = serial.Serial(
...         port='COM1',
...         baudrate=9600,
...         parity="N",
...         stopbits=1,
...         bytesize=8,
...         timeout=8
...
)
```

I can then verify that the serial port is open using the isOpen() method.

```
>>> console.isOpen()
True
```

At this point, I need to write and read from the serial port and this is where it got a bit tricky. First, let's just send a newline down the channel (note, it is a Windows-style newline).

```
>>> console.write("\r\n")
2L
```

Now I use the inWaiting() method to see if there are any data bytes waiting to be read. inWaiting() will return the number of bytes that need to be read.

```
>>> console.inWaiting()
225L
```

Now there are 225 bytes ready to be read. Let's read them and display what we receive:

```
>>> input_data = console.read(225)
>>> print input_data


User Access Verification

Username:
% Username:  timeout expired!
```

There was more on the screen, but you get the picture.

As you can see we are able to send data to the serial port and read data from it. Now let's create a Python script and see if we can handle the login process.

In order to do this, I will need to send a newline, wait for a second, and then read the data. If 'Username' is present in the input data, then I can proceed.

Here is a crude script to accomplish this:

```
import serial
import sys
import time

import credentials

READ_TIMEOUT = 8


def main():

    print "\nInitializing serial connection"
    console = serial.Serial(
        port='COM1',
        baudrate=9600,
        parity="N",
        stopbits=1,
        bytesize=8,
        timeout=READ_TIMEOUT
    )

    if not console.isOpen():
        sys.exit()

    console.write("\r\n\r\n")
    time.sleep(1)
    input_data = console.read(console.inWaiting())
    print input_data
    if 'Username' in input_data:
        console.write(credentials.username + '\r\n')
    time.sleep(1)
    input_data = console.read(console.inWaiting())

if __name__ == "__main__":
    main()
```

Note, I have stored the username and password in an external file called credentials.py and am importing them.

At this point if I run the script I get:

```
$ python serial1.py

Initializing serial connection


User Access Verification

Username:
```

Note, I do not see 'Username:' consistently when I run this script. This is due to the behavior of the Cisco router (i.e. it will show 'Username:' three times and then it will delay for about 3 seconds and then it will show me a message that 'router_name con0 is now available').

Even with that problem let's proceed and try to send the password down the channel.

```
import serial
import sys
import time

import credentials

READ_TIMEOUT = 8
```

```
def main():

    print "\nInitializing serial connection"

    console = serial.Serial(
        port='COM1',
        baudrate=9600,
        parity="N",
        stopbits=1,
        bytesize=8,
        timeout=READ_TIMEOUT
    )

    if not console.isOpen():
        sys.exit()

    console.write("\r\n\r\n")
    time.sleep(1)
    input_data = console.read(console.inWaiting())
    if 'Username' in input_data:
        console.write(credentials.username + '\r\n')
    time.sleep(1)
    input_data = console.read(console.inWaiting())
    if 'Password' in input_data:
        console.write(credentials.password + '\r\n')
    time.sleep(1)
    input_data = console.read(console.inWaiting())
    print input_data


if __name__ == "__main__":
    main()
```

As you can see I now handle sending the password. If I run this script, I get the following:

```
$ python serial2.py

Initializing serial connection


pynet-rtr1>
pynet-rtr1>
```

We are now receiving the router prompt and are able to login successfully.

Now the script has some significant reliability issues, but you can see from the above that we are able to successfully interact with the router via its console port using Python.

Here is an expanded program that I created:

https://github.com/ktbyers/pynet/blob/master/serial/cisco_serial.py

This script still is pretty rough, but it better handles some of the issues. I also had the script execute the command "show ip int brief" and return the output.

Here you can see the program being run:

```
$ python cisco_serial.py

Initializing serial connection
Logging into router
We are logged in

show ip int brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0              unassigned      YES unset  down                  down
FastEthernet1              unassigned      YES unset  down                  down
FastEthernet2              unassigned      YES unset  down                  down
FastEthernet3              unassigned      YES unset  down                  down
FastEthernet4              10.220.88.20    YES NVRAM  up                    up
Vlan1                      unassigned      YES unset  down                  down
pynet-rtr1>
pynet-rtr1>
Logging out from router
Successfully logged out from router
```

Obviously, in order to send configuration changes--we would have to go into enable mode, then config mode, and then execute our changes. But these series of actions should not be too tough to accomplish.

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From: pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Monday, March 21, 2016 3:33 PM
To: Laney, Ricky A
Subject: [PyNet] - Network Automation Course Starts on April 7

Ricky

Having fun banging-out the same CLI-commands over and over and over again? Well...

My next network automation course starts on April 7. This is a paid course and covers topics like Paramiko(SSH), Netmiko, CiscoConfParse, Python and SNMP, writing reusable Python code, Ansible (in various network engineering use cases), Git, and using Arista's eAPI.

Here are some quotes from previous students:

"Thanks for helping me get started with all this, your course made coding so much more personally relevant, because I could solve problems that mattered to me - networking ones."

"I am now coding usable scripts. Nothing I had done to date had connected the dots for me like this class did."

For more information see:

https://pynet.twb-tech.com/class.html

Early sign-up discount ends on Tuesday 3/22 at 9PM Pacific Time.

Regards,

Kirk Byers
ktbyers@twb-tech.com

Laney, Ricky A

From:	pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Thursday, March 17, 2016 9:18 AM
To:	Laney, Ricky A
Subject:	[PyNet] Learning Python: Week8 (Modules and Packages)

Ricky

In this email of Learning Python we are going to cover the following:

I. Modules (Part 1)
video http://youtu.be/FanwaY25Kkc
Length is 13 minutes

II. Modules (Part 2)
video http://youtu.be/qG-MxgakbE8
Length is 6 minutes

III. Packages
video http://youtu.be/3QO1mQy-LRE
Length is 13 minutes

Make sure that you set your video resolution in YouTube to at least 360p. The resolution button is at the bottom of the video and looks like a little gear.

Python Humor

```
# try it :-)
from __future__ import braces
```

Additional content that you may be interested in:

One thing that I forgot to mention in the videos--Python modules (files) and packages (directories that contain __init__.py) must follow Python naming rules or else you will not be able to import them. For example:

```
# You can execute the file test-zzz.py, but you cannot import it because test-zzz is not
# a valid name in Python (hyphens are not allowed in a Python name).
$ python test-zzz.py
hello world

$ python
>>> import test-zzz
  File "<stdin>", line 1
    import test-zzz
               ^
SyntaxError: invalid syntax

# Just rename this file to be test_zzz.py (which is a valid Python name)
$ mv test-zzz.py test_zzz.py
$ python
>>> import test_zzz
hello world
```

Good article on the differences between Python2 and Python3:
Key differences between Python 2.7.x and Python 3.x

David Gee (@LSP42) has a good blog post on network automation and learning Python:
http://ipengineer.net/2014/05/from-cli-to-python-beginner/

Note, I had trouble finding a good document on modifying the PYTHONPATH in Windows. Here is how you can modify the PYTHONPATH environment variable using the 'set' command (tested on Windows XP, but this should also work on Windows7). This is from the 'command prompt' and is only usable during that command prompt session.

```
# Shows my current PYTHONPATH
C:\Python27>set PYTHONPATH
PYTHONPATH=C:\Python27\Lib;C:\Python27\DLLs;C:\Python27\Scripts

# Sets a new PYTHONPATH (appending to the current %pythonpath%)
C:\Python27>set PYTHONPATH=%pythonpath%;C:\Python27\ktb;

# Checks that the PYTHONPATH is right
C:\Python27>set PYTHONPATH
PYTHONPATH=C:\Python27\Lib;C:\Python27\DLLs;C:\Python27\Scripts;C:\Python27\ktb;

# Test that it works correctly (C:\Python27\ktb contains z_test.py)
C:\Python27>python
>>> import z_test
Hello world
```

You can also permanently modify this PYTHONPATH variable by editing your Windows Environment Variables (the below link shows you how to edit

Environment Variables in Windows7). On my old Windows XP system, PYTHONPATH is a 'System Variable' (NOT a User Variable) and is set to:

```
    PYTHONPATH = C:\Python27\Lib;C:\Python27\DLLs;C:\Python27\Scripts
```

Be careful when editing your system's Environment Variables--you can cause significant problems with your computer (if done incorrectly).

Exercises

Reference code for these exercises is posted on GitHub at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class8

I. Re-using the IP address validation function created in Class #6, exercise3; create a Python module that contains only this one ip validation function.

A. Modify this Python module such that you add a set of tests into the module. Use the __name__ == '__main__' technique to separate the test code from the function definition. In your test code, verify your IP address validation function against each of the following IP addresses (False means it is an invalid IP address; True means it is a valid IP address).

```
test_ip_addresses = {
    '192.168.1' : False,
    '10.1.1.' : False,
    '10.1.1.x' : False,
    '0.77.22.19' : False,
    '-1.88.99.17' : False,
    '241.17.17.9' : False,
    '127.0.0.1' : False,
    '169.254.1.9' : False,
    '192.256.7.7' : False,
    '192.168.-1.7' : False,
    '10.1.1.256' : False,
    '1.1.1.1' : True,
    '223.255.255.255': True,
    '223.0.0.0' : True,
    '10.200.255.1' : True,
    '192.168.17.1' : True,
}
```

B. Execute this module--make sure all of the tests pass.

C. Import this module into the Python interpreter shell. Make sure the test code does not execute. Also test that you can still correctly call the ip validation function.

II. On GitHub, there is the following show version output:

https://github.com/ktbyers/pynet/blob/master/learnpy_ecourse/class8/show_version.txt

A. Create three functions in three separate modules and put them in a show_version directory (in practice you wouldn't do this--you would just have them all in one file, but this will let you experiment with packages).

1. Function1 = obtain_os_version -- process the show version output and return the OS version (Version 15.0(1)M4) else return None.

2. Function2 = obtain_uptime -- process the show version output and return the network device's uptime string (uptime is 12 weeks, 5 days, 1 hour, 4 minutes) else return None.

3. Function3 = obtain_model -- process the show version output and return the model (881) else return None.

B. Make a package out of this 'show_version' directory using a blank __init__.py file.

1. Now that this package has been created, test importing each of the modules individually from the parent directory. In other words, you have a parent directory that contains the following:

```
.
./show_version
./show_version/__init__.py
./show_version/os_version.py
./show_version/model.py
./show_version/uptime.py
```

From the parent directory go into the Python interpreter shell and make sure you can do the following:

```
>>> import show_version.os_version
>>> import show_version.model
>>> import show_version.uptime
```

2. Now edit the __init__.py file such that it automatically loads each of the modules. In other words, you should be able to type:

```
>>> import show_version
```

and have all of your functions available as follows:

```
>>> show_version.obtain_model(show_ver_data)
>>> show_version.obtain_os_version(show_ver_data)
>>> show_version.obtain_uptime(show_ver_data)
```

C. Write a script that processes the show_version output using this package. It should return something similar to the following:

```
model:          881
os_version:    Version 15.0(1)M4
uptime:         uptime is 12 weeks, 5 days, 1 hour, 4 minutes
```

III. Experiment with modifying your PYTHONPATH. Modify your PYTHONPATH such that the Python module you created in exerciseI is importable (from outside your current working directory).

CLASS OUTLINE

I. Modules (Part1)
- A. What is a module?
- B. Why do we need modules?
  - 1. Re-using other people's code
  - 2. Re-using your own code
- C. Two ways of importing a module (import and from)
- D. What happens when a module is imported?

II. Modules (Part2)
- A. A module has its own namespace
- B. Using __name__ == '__main__' to separate executable code from importable code

III. How Python finds libraries
- A. Modifying the PYTHONPATH

IV. Packages
- A. What is a package?
- B. The __init__.py file
- C. A package allows modules in the package directory to be imported
- D. Processing of __init__.py
- E. Adding other import statements into __init__.py

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops
Passing Arguments into a Script

Week4
Introduction
While Loops
Dictionaries
Exceptions

Week5
Class Review (weeks 1 - 4)

Week6
Introduction
Functions, Part1
Namespaces
Functions, Part2

Week7
Files
Regular Expressions

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, March 15, 2016 1:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - ciscoconfparse for Parsing Hierarchical Configurations

Ricky

There is a Python library named ciscoconfparse that helps you parse Cisco hierarchical configurations. This would include other vendors that are Cisco-like. Now what I mean by Cisco-like is that the configuration is text-based and that it uses indentation to indicate configuration hierarchy.

For example, consider the following configuration:

```
interface FastEthernet4
 description *** LAN connection (don't change) ***
 ip address 10.220.88.20 255.255.255.0
 duplex auto
 speed auto
```

'description', 'ip address', 'duplex', and 'speed' are all configuration items pertaining to interface FastEthernet4. They are inside FastEthernet4 hierarchically.

Now programmatic string processing of Cisco configurations can get complex. The ciscoconfparse library greatly helps with this. It can be used to identify the parent/child relationships of the Cisco hierarchy and it adds convenient searching capabilities.

Let's look at some examples of this.

Now one minor note on terminology. There is a ciscoconfparse library which I am going to identify in all lower case. There is also a CiscoConfParse class which I will identify in this Python PEP8 style.

First, I have installed ciscoconfparse using:

```
pip install ciscoconfparse
```

Then I have a Cisco configuration that contains the following:

```
interface FastEthernet0
 no ip address
!
interface FastEthernet1
 no ip address
!
interface FastEthernet2
```

```
 no ip address
!
interface FastEthernet3
 no ip address
!
interface FastEthernet4
 description *** LAN connection (don't change) ***
 ip address 10.220.88.20 255.255.255.0
 duplex auto
 speed auto
!
interface Vlan1
 no ip address
```

Now let's go into the Python interpreter shell and see what we can do with CiscoConfParse.

First we have to load the CiscoConfParse class.

```
>>> from ciscoconfparse import CiscoConfParse
```

Then I use CiscoConfParse to parse my Cisco configuration (which resides in an external file):

```
>>> cisco_cfg = CiscoConfParse("cisco.txt")
>>> cisco_cfg
<CiscoConfParse: 167 lines / syntax: ios / comment delimiter: '!' / factory: False>
```

Now that I have a CiscoConfParse object, let's search for all of the config lines that start with the word 'interface'.

```
>>> rtr_interfaces = cisco_cfg.find_objects(r"^interface")
>>> rtr_interfaces
[<IOSCfgLine # 111 'interface FastEthernet0'>, <IOSCfgLine # 114 'interface FastEthernet1'>,
<IOSCfgLine # 117 'interface FastEthernet2'>, <IOSCfgLine # 120 'interface FastEthernet3'>,
<IOSCfgLine # 123 'interface FastEthernet4'>, <IOSCfgLine # 129 'interface Vlan1'>]
```

Notice, I now have a list of six elements matching the six interfaces in the configuration.

Now that I have all these interfaces--what if I want to look at the children of one of them:

```
>>> intf_fa4 = rtr_interfaces[4]
>>> intf_fa4
<IOSCfgLine # 123 'interface FastEthernet4'>
>>>
>>> intf_fa4.children
[<IOSCfgLine # 124 ' description *** LAN connection (don't change) ***' (parent is # 123)>,
<IOSCfgLine # 125 ' ip address 10.220.88.20 255.255.255.0' (parent is # 123)>, <IOSCfgLine #
126 ' duplex auto' (parent is # 123)>, <IOSCfgLine # 127 ' speed auto' (parent is # 123)>]
```

Since '.children' returns a list, I can also iterate over it. This will allow me to view the children in a cleaner way.

```
>>> for child in intf_fa4.children:
...   print child.text
```

```
...
 description *** LAN connection (don't change) ***
 ip address 10.220.88.20 255.255.255.0
 duplex auto
 speed auto
```

So that is fairly nice...I could pretty easily search for lines beginning with 'interface' and then from this retrieve all of the child configuration elements.

You can also directly search for a combination of conditions. For example, what if I want to find the interfaces that have 'no ip address'.

```
>>> no_ip_address = cisco_cfg.find_objects_w_child(parentspec=r"^interface", childspec=r"no
ip address")
>>> no_ip_address
[<IOSCfgLine # 111 'interface FastEthernet0'>, <IOSCfgLine # 114 'interface FastEthernet1'>,
<IOSCfgLine # 117 'interface FastEthernet2'>, <IOSCfgLine # 120 'interface FastEthernet3'>,
<IOSCfgLine # 129 'interface Vlan1'>]
```

I can then print out both the parents and children fairly easily:

```
>>> for my_int in no_ip_address:
...   print my_int.text
...   for child in my_int.children:
...     print child.text
...
interface FastEthernet0
 no ip address
interface FastEthernet1
 no ip address
interface FastEthernet2
 no ip address
interface FastEthernet3
 no ip address
interface Vlan1
 no ip address
```

CiscoConfParse also has a search for find_objects_wo_child (i.e. without child). So what if I want to find all of the interfaces that do NOT have 'no ip address'.

```
>>> ip_configured = cisco_cfg.find_objects_wo_child(parentspec=r"^interface", childspec=r"no
ip address")
>>> ip_configured

[<IOSCfgLine # 123 'interface FastEthernet4'>]
```

You can also directly look at child and parent relationships. For example, the 'cisco.txt' file that we loaded contains the following configuration (Note, I slightly modified this to hide the key-hash and email address):

```
ip ssh pubkey-chain
  username testuser
   key-hash ssh-rsa C0B699C6EAAAE18E9EC099B30F5D01DA invalid@domain.com
```

Now I can use the find_objects() method to find this configuration element:

```
>>> ssh_pubkey_chain = cisco_cfg.find_objects(r"^ip ssh pubkey")[0]
>>> ssh_pubkey_chain
<IOSCfgLine # 97 'ip ssh pubkey-chain'>
```

I can then check if it is a parent and if it is a child:

```
>>> ssh_pubkey_chain.is_parent
True
>>> ssh_pubkey_chain.is_child
False
```

I can find its direct children:

```
>>> ssh_pubkey_chain.children
[<IOSCfgLine # 98 '  username testuser' (parent is # 97)>]
```

I can find all of its children (note, from this that .children implies a direct child whereas .all_children implies both direct and indirect children).

```
>>> ssh_pubkey_chain.all_children
[<IOSCfgLine # 98 '  username testuser' (parent is # 97)>, <IOSCfgLine # 99 '   key-hash ssh-rsa C0B699C6EAAAE18E9EC099B30F5D01DA invalid@domain.com' (parent is # 98)>]
```

There are similar attributes that go the other way i.e. from children to parents (namely 'parent' and 'all_parents').

As you can see (hopefully), CiscoConfParse can help you parse Cisco and search through Cisco and Cisco-like configurations. It is also pretty simple to use.

For more information see:
ciscoconfparse on GitHub

CiscoConfParse Tutorial

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From:	pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Thursday, March 10, 2016 9:01 AM
To:	Laney, Ricky A
Subject:	[PyNet] Learning Python: Week7 (Files and Regular Expressions)

Ricky

In this email of Learning Python we are going to cover the following:

I. Files
video http://youtu.be/Hjr3ev1dJWo
Length is 15 minutes

II. Regular Expressions
video http://youtu.be/EjpPivmkbWI
Length is 14 minutes

Make sure that you set your video resolution in YouTube to at least 360p. The resolution button is at the bottom of the video and looks like a little gear.

Regular Expression Humor:

http://xkcd.com/208/

Additional content that you may be interested in:

Google Developers - "Python Regular Expressions". Good content on regular expressions in Python; I recommend that you read through this.
https://developers.google.com/edu/python/regular-expressions

"Dive Into Python" - Regular Expressions. I recommend that you work through sections 7.1 and 7.2.
http://www.diveintopython.net/regular_expressions/
http://www.diveintopython.net/regular_expressions/street_addresses.html

A brief article on using regular expressions on network devices:
http://packetpushers.net/regular-expressions-for-network-engineers/

The following web page has details on the different Python file open modes:
http://www.tutorialspoint.com/python/python_files_io.htm

See the table (about 1/3 of the way down the page) with the column headers "Modes" and "Description" in "the open Function" section ("r", "rb", "r+", "rb+", etc.).

Exercises:

Reference code for these exercises is posted on GitHub at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class7

The 'show ip ospf interface' output is courtesy of Darren O'Connor see:
http://mellowd.co.uk/ccie/?p=5097

1. In the following directory there are six CDP files:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class7/CDP_DATA

These CDP files correspond to the following network (same network as exercise1 from class #5):
https://pynet.twb-tech.com/static/img/simple_diagram1.png

a. Create a program that opens the 'r1_cdp.txt' file and using regular expressions extracts the remote hostname, remote IP address, model, vendor, and device_type.

b. Create a program that opens the 'sw1_cdp.txt' file and finds all of the remote hostnames, remote IP addresses, and remote platforms. Your output should look similar to the following:

```
remote_hosts: ['R1', 'R2', 'R3', 'R4', 'R5']
         IPs: ['10.1.1.1', '10.1.1.2', '10.1.1.3', '10.1.1.4', '10.1.1.5']
    platform: ['Cisco 881', 'Cisco 881', 'Cisco 881', 'Cisco 881', 'Cisco 881']
```

2. In the following directory there is a file 'ospf_single_interface.txt':

https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class7/OSPF_DATA

Open this file and extract the interface, IP address, area, type, cost, hello timer, and dead timer. Use regular expressions to accomplish your extraction.

Your output should look similar to the following:

```
Int:     GigabitEthernet0/1
IP:      172.16.13.150/29
Area:    30395
Type:    BROADCAST
Cost:    1
Hello:   10
Dead:    40
```

3. In the following directory there is a file 'ospf_data.txt':

https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class7/OSPF_DATA

This file contains the output from 'show ip ospf interface'. Using functions and regular expressions parse this output to display the following (note, I ended up using re.split() as part of the solution to this problem):

```
Int:    Loopback0
IP:     10.90.3.38/32
Area: 30395
Type: LOOPBACK
Cost: 1

Int:    GigabitEthernet0/1
IP:     172.16.13.150/29
Area:  30395
Type:  BROADCAST
Cost:  1
Hello: 10
Dead: 40

Int:    GigabitEthernet0/0.2561
IP:     10.22.0.117/30
Area:  30395
Type:  POINT_TO_POINT
Cost:  1
Hello: 10
Dead: 40
```

4. Experiment with 'glob' (see below)

Using the glob library you can more easily open a set of files. Notice how I use the shell '*' character to match *_cdp.txt. I could then open all of these files and process the data inside of them.

```
>>>> CODE <<<<

# This code assumes that all of the CDP files are in a subdirectory called
# CDP_DATA beneath the current working directory

>>> from glob import glob
>>> cdp_files = glob('CDP_DATA/*_cdp.txt')

>>> cdp_files
['CDP_DATA/r1_cdp.txt', 'CDP_DATA/r2_cdp.txt',
'CDP_DATA/r3_cdp.txt',  'CDP_DATA/sw1_cdp.txt', 'CDP_DATA/r4_cdp.txt',
'CDP_DATA/r5_cdp.txt']
```

>>>> END CODE <<<<

CLASS OUTLINE

I. Files

A. Reading a file
1. f = open("a_file")
2. readlines() method
3. seek() method
4. readline() method
5. read() method
6. directly iterating over a file
7. close() method

B. Writing a file
1. Caution when writing binary files on Windows
2. Caution not to overwrite existing files
3. f = open("newfile", "w")
4. write() method
5. flush() method

C. Appending to a file
1. f = open("appendfile", "a")

D. Using with
1. with open("a_file") as f:

II. Regular expressions

A. Using re.search()
1. Always use raw strings for the pattern
2. Using .group() for retrieval
3. Saving parts of pattern using ()
4. + and * are by default greedy
5. Making + and * non-greedy

B. Using re.findall()
1. findall() with parenthesis
2. findall() with multiple () returns a list of tuples

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops
Passing Arguments into a Script

Week4
Introduction
While Loops
Dictionaries
Exceptions

Week5
Class Review (weeks 1 - 4)

Week6
Introduction
Functions, Part1
Namespaces
Functions, Part2

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, March 8, 2016 12:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Netmiko Library

Ricky

I have recently been working on an open-source Python library that simplifies SSH management to network devices. The library is based on the Paramiko SSH library.

The library is located here:

https://github.com/ktbyers/netmiko

and the latest release/download can be found at:

https://github.com/ktbyers/netmiko/releases

The purposes of the library are the following:
1. Successfully establish an SSH connection to a device
2. Make it easy to execute show commands and to retrieve the output data
3. Make it easy to execute configuration commands
4. Do the above across a broad set of networking vendors and platforms

I have observed across time that you can encounter quite a few problems in managing Python SSH sessions. As an example, an HP ProCurve switch presents a "Press any key to continue" message after login. The ProCurve also has ANSI escape codes in the output. Note, this is not to pick on HP, you will run into issues with other vendors as well.

These type of issues can easily add hours of development time to your script and what is worse we have no way of leveraging the learning from other people (i.e. each person goes and reinvents the wheel).

So netmiko intends to simplify this lower-level SSH management across a wide set of networking vendors and platforms.

As of July 2015, Netmiko has been tested on the following:
Cisco IOS
Cisco IOS-XE
Cisco ASA
Cisco NX-OS

Cisco IOS-XR
Cisco WLC (limited testing)
Arista EOS
HP ProCurve
HP Comware (limited testing)
Juniper Junos
Brocade VDX (limited testing)
F5 LTM (experimental)
Huawei (limited testing)

Let me show you a couple of examples.

Example 1, a simple SSH session to a Cisco router that executes the 'show ip int brief' command.

First, I import the netmiko library and then define the network device as a dictionary:

```
>>> import netmiko
>>> cisco_881 = {
...     'device_type': 'cisco_ios',
...     'ip':   '10.10.10.10',            # real IP address hidden as it was a public IP
...     'username': 'test1',
...     'password': 'password',
...     'secret': 'secret',
... }
```

I then call a dispatcher using the device_type. The dispatcher just makes sure the right class is used.

```
>>> SSHClass = netmiko.ssh_dispatcher(cisco_881['device_type'])
```

I then connect to the device using the dictionary I created earlier:

```
>>> net_connect = SSHClass(**cisco_881)
SSH connection established to 10.10.10.10:22
Interactive SSH session established
```

At this point I have an SSH connection to the device and can execute commands on the channel. I use the send_command() method for this.

```
>>> output = net_connect.send_command("show ip int brief")
>>> print output
Interface                  IP-Address      OK? Method Status                Protocol
```

```
FastEthernet0          unassigned     YES unset  down             down
FastEthernet1          unassigned     YES unset  down             down
FastEthernet2          unassigned     YES unset  down             down
FastEthernet3          unassigned     YES unset  down             down
FastEthernet4          10.220.88.20    YES NVRAM  up                up
Vlan1              unassigned     YES unset  down             down
```

Let's also try to make a configuration change to this router. First, I go into enable mode (which may or may not be necessary depending on your AAA setup):

```
>>> net_connect.enable()
>>> output = net_connect.send_command("show run | inc logging")
>>> print output
logging buffered 20010
no logging console
```

Now in order to make configuration changes, I specify a list of config commands that I want to execute. This could be a single command or multiple commands.

```
>>> config_commands = ['logging buffered 19999']
```

I then execute the send_config_set() method. This method takes care of entering configuration mode, enters the command(s), and then exits configuration mode.

```
>>> output = net_connect.send_config_set(config_commands)
>>> print output

pynet-rtr1#config term
Enter configuration commands, one per line.  End with CNTL/Z.
pynet-rtr1(config)#logging buffered 19999
pynet-rtr1(config)#end
pynet-rtr1#
```

I can then verify my change:

```
>>> output = net_connect.send_command("show run | inc logging")
>>> print output
logging buffered 19999
no logging console
```

Example 2, executing 'show arp' on a set of networking devices consisting of different vendors and platforms.

First, I define the networking devices:

```
>>> import netmiko
>>>
```

```
>>> cisco_881 = {
...     'device_type': 'cisco_ios',
...     'ip':   '10.10.10.10',
...     'username': 'admin',
...     'password': 'password',
...     'secret': 'secret',
... }
>>>
>>> cisco_asa = {
...     'device_type': 'cisco_asa',
...     'ip':   '10.10.10.11',
...     'username': 'admin',
...     'password': 'password',
...     'secret': 'secret',
... }
>>>
>>> arista_veos_sw = {
...     'device_type': 'arista_eos',
...     'ip':   '10.10.10.12',
...     'username': 'admin',
...     'password': 'password',
...     'port': 8522,                 # Firewall in front of this device, PAT from 8522 to 22
... }
>>>
>>> hp_procurve = {
...     'device_type': 'hp_procurve',
...     'ip':   '10.10.10.12',
...     'username': 'admin',
...     'password': 'password',
...     'port': 9922,                 # Firewall in front of this device, PAT from 9922 to 22
... }
>>>
>>> juniper_srx = {
...     'device_type': 'juniper',
...     'ip':   '10.10.10.12',
...     'username': 'root',
...     'password': 'password',
...     'port': 9822,                 # Firewall in front of this device, PAT from 9822 to 22
... }
```

Then create a list that includes all of these devices:

```
>>> all_devices = [cisco_881, cisco_asa, arista_veos_sw, hp_procurve, juniper_srx]
```

Finally, iterate over each of these devices and execute the 'show arp' command:

```
>>> for a_device in all_devices:
...   SSHClass = netmiko.ssh_dispatcher(a_device['device_type'])
...   net_connect = SSHClass(**a_device)
...   output = net_connect.send_command("show arp")
...   print "\n\n>>>>>>>>> Device {0} <<<<<<<<<".format(a_device['device_type'])
...   print output
...   print ">>>>>>>>> End <<<<<<<<<"
...
```

```
>>>>>>>>> Device cisco_ios <<<<<<<<<
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  10.220.88.1            42   001f.9e92.16fb  ARPA   FastEthernet4
Internet  10.220.88.20            -   c89c.1dea.0eb6  ARPA   FastEthernet4
Internet  10.220.88.100         207   f0ad.4e01.d933  ARPA   FastEthernet4
>>>>>>>>> End <<<<<<<<<


>>>>>>>>> Device cisco_asa <<<<<<<<<
    inside 10.220.88.10 0018.fe1e.b020 179
    inside 10.220.88.39 6464.9be8.08c8 316
    inside 10.220.88.31 5254.0001.3737 1068
    inside 10.220.88.100 f0ad.4e01.d933 2568
    inside 10.220.88.30 5254.0092.13bb 4049
    inside 10.220.88.29 5254.0098.69b6 5408
    inside 10.220.88.21 1c6a.7aaf.576c 6318
    inside 10.220.88.28 5254.00ee.446c 6796
    inside 10.220.88.40 001c.c4bf.826a 8358
    inside 10.220.88.20 c89c.1dea.0eb6 11838
>>>>>>>>> End <<<<<<<<<


>>>>>>>>> Device arista_eos <<<<<<<<<
Address         Age (min)  Hardware Addr   Interface
10.220.88.1             0  001f.9e92.16fb  Vlan1, Ethernet1
10.220.88.28            0  5254.00ee.446c  Vlan1, not learned
10.220.88.29            0  5254.0098.69b6  Vlan1, not learned
10.220.88.30            0  5254.0092.13bb  Vlan1, not learned
>>>>>>>>> End <<<<<<<<<


>>>>>>>>> Device hp_procurve <<<<<<<<<

 IP ARP table

  IP Address       MAC Address       Type    Port
  --------------- ----------------- ------- ----
  10.220.88.1      001f9e-9216fb     dynamic 19


>>>>>>>>> End <<<<<<<<<


>>>>>>>>> Device juniper <<<<<<<<<

MAC Address       Address         Name                      Interface           Flags
00:1f:9e:92:16:fb 10.220.88.1     10.220.88.1               vlan.0              none
f0:ad:4e:01:d9:33 10.220.88.100   10.220.88.100             vlan.0              none
Total entries: 2

>>>>>>>>> End <<<<<<<<<
```

Note, it took 51 seconds for the above for-loop code to execute. This could be improved by executing the SSH sessions in parallel, for an example of this see:

https://github.com/ktbyers/netmiko/blob/master/examples/multiprocess_example.py

For a list of supported device_types see:

https://github.com/ktbyers/netmiko/blob/master/netmiko/ssh_dispatcher.py

If you have additional vendors or platforms that you want supported, email me (or even better send me a pull-request).

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From:	pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Thursday, March 3, 2016 9:02 AM
To:	Laney, Ricky A
Subject:	[PyNet] Learning Python: Week6 (Functions)

Ricky

We are now entering the second half of the Learning Python Course.

The second half of this course includes some important content for improving your Python programming. In particular, Class6 on Functions and Namespaces, Class8 on Modules, and Class9 on Classes and Objects.

In this email of Learning Python we are going to cover the following:

I. Introduction Week6
video http://youtu.be/sjYBE6EWwuc
Length is 3 minutes

II. Functions, Part1
video http://youtu.be/i1MzASLYOZY
Length is 12 minutes

III. Namespaces
video http://youtu.be/r8U3_KkvKtw
Length is 10 minutes

IV. Functions, Part2
video http://youtu.be/pRaVagkh9I8
Length is 12 minutes

Additional content that you may be interested in

There is a good chapter on functions in "Learn Python the Hard Way" (I would stop after you finish, "What You Should See").
http://learnpythonthehardway.org/book/ex18.html

Darren O'Connor has a blog on "Defined Functions - Python".
http://mellowd.co.uk/ccie/?p=5118

Exercises

Reference code for these exercises is posted on GitHub at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class6

1. Create a function that returns the multiplication product of three parameters--x, y, and z. z should have a default value of 1.
   a. Call the function with all positional arguments.
   b. Call the function with all named arguments.
   c. Call the function with a mix of positional and named arguments.
   d. Call the function with only two arguments and use the default value for z.

2. Write a function that converts a list to a dictionary where the index of the list is used as the key to the new dictionary (the function should return the new dictionary).

3a.Convert the IP address validation code (Class4, exercise1) into a function, take one variable 'ip_address' and return either True or False (depending on whether 'ip_address' is a valid IP). Only include IP address checking in the function--no prompting for input, no printing to standard output.

3b. Import this IP address validation function into the Python interpreter shell and test it (use both 'import x' and 'from x import y').

4. Create a function using your dotted decimal to binary conversion code from Class3, exercise1. In the function--do not prompt for input and do not print to standard output. The function should take one variable 'ip_address' and should return the IP address in dotted binary format always padded to eight binary digits (for example, 00001010.01011000.00010001.00010111). You might want to create other functions as well (for example, the zero-padding to eight binary digits).

5. Write a program that prompts a user for an IP address, then checks if the IP address is valid, and then converts the IP address to binary (dotted decimal format). Re-use the functions created in exercises 3 and 4 ('import' the functions into your new program).

Class Outline

I. Introduction
   A. Why write functions?

II. Functions Part1
   A. Function with no parameters
      1. Syntax and structure

        2. Calling the function
        3. Return value
        4. Using the return value
        5. Docstrings

    B. Function with parameters
        1. Syntax
        2. Default values

    C. Various ways of passing arguments to functions
        1. Positional arguments
        2. Named arguments
        3. Mixing positional and named arguments

III. Python Namespaces
    A. Functions create their own namespace
    B. Name resolution order

IV. Functions Part2
    A. Using lists and dicts as function arguments
    B. Importing a function

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops
Passing Arguments into a Script

Week4
Introduction
While Loops
Dictionaries
Exceptions

Week5
Class Review (weeks 1 - 4)

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, March 1, 2016 12:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Retrieving SNMPv3 information using Python

Ricky

In this email, I provide a brief demonstration of using SNMPv3 and Python to retrieve information from a Cisco router.

As a quick overview, SNMPv3 was standardized in 2002. It added both encryption and a reasonable authentication scheme to SNMP. Prior to this, both SNMPv1 and SNMPv2c relied on a simple community string that passed clear-text on the wire. Similarly, all of the SNMP data itself was not encrypted in SNMPv1 and SNMPv2c.

In order to use SNMPv3, I configured the following on a Cisco 881 router:

```
>>>>
snmp-server view VIEWSTD iso included
snmp-server group READONLY v3 priv read VIEWSTD access 98
snmp-server user pysnmp READONLY v3 auth sha <auth_key> priv aes 128 <crypt_key>
>>>>
```

The above commands accomplish three items. First, a view is created that allows access to the entire SNMP tree. Second, a read-only group is created that requires both authentication and encryption ('priv' option) and that uses the previously created view. Third, an SNMP user is created named 'pysnmp' that is bound to the READONLY group and that uses SHA authentication with AES128 encryption.

Note, ACL98 is also bound to the READONLY group, but this ACL is currently configured to allow all IP.

After the router is configured, we should be able to connect to it using SNMPv3:

```
$ snmpget -v 3 -u pysnmp -l authpriv -a SHA -A "<auth_key>" -x AES -X "<crypt_key>"
<ip_address>:7961 sysUpTime.0

DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (370751064) 42 days, 21:51:50.64
```

Here I use the snmpget command to retrieve the system uptime. This command uses SNMPv3 (-v 3), a username of pysnmp (-u pysnmp), both encryption and authentication (-l authpriv), SHA authentication (-a SHA), and AES encryption (-x AES). The command above also uses a non-standard SNMP port (7961) due to a static PAT (i.e. there is a firewall performing PAT between the test server and the router).

Now that we can communicate with the router using SNMPv3, let's try to connect using Python's PySNMP library. In order to do this, I created an SNMPv3 helper function named snmp_get_oid_v3. You can find this function in the following library:

https://github.com/ktbyers/pynet/blob/master/snmp/snmp_helper.py

The function definition is as follows:

```
def snmp_get_oid_v3(snmp_device, snmp_user, oid='.1.3.6.1.2.1.1.1.0',
        auth_proto='sha', encrypt_proto='aes128', display_errors=True):
```

snmp_device is a tuple consisting of (IP_or_hostname, snmp_port); snmp_user is also a tuple consisting of (username, auth_key, encrypt_key). The default OID is the MIB-2 sysDescr field. I also intentionally set the default authentication and the default encryption to match the settings of my router (i.e. SHA and AES128).

Here is an example of using this function to retrieve data via SNMPv3:

```
>>> import snmp_helper
>>>
>>> IP = '10.10.10.10'
>>>
>>> a_user = 'pysnmp'
>>> auth_key = '********'
>>> encrypt_key = '********'
>>>
>>> snmp_user = (a_user, auth_key, encrypt_key)
>>>
>>> pynet_rtr1 = (IP, 7961)
>>>
>>> snmp_data = snmp_helper.snmp_get_oid_v3(pynet_rtr1, snmp_user)
>>> output = snmp_helper.snmp_extract(snmp_data)
>>>
>>> print output
Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Version 15.4(2)T1,
RELEASE SOFTWARE (fc3)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2014 by Cisco Systems, Inc.
Compiled Thu 26-Jun-14 14:15 by prod_rel_team
```

* The IP, auth_key, and encrypt_key values have been masked or otherwise modified.

Note, the call to snmp_helper.snmp_extract uses a second function in the snmp_helper library. This snmp_extract function converts the output data into a more readable form.

Now let's repeat this process using a different OID. For example, sysUpTime:

```
>>> sys_uptime = '1.3.6.1.2.1.1.3.0'
>>> snmp_data = snmp_helper.snmp_get_oid_v3(pynet_rtr1, snmp_user, oid=sys_uptime)
>>> output = snmp_helper.snmp_extract(snmp_data)
>>> print output
370829991
```

The sysUpTime value is in hundredths of seconds. Consequently, the above value is an uptime of slightly less than 43 days.

Thus, we are able to communicate with a Cisco router using an encrypted and authenticated SNMP channel via Python.

Regards,

Kirk Byers
CCIE #6243 emeritus
ktbyers@twb-tech.com

1235 Connecticut St San Francisco CA 94107 USA

Laney, Ricky A

From:	pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Thursday, February 25, 2016 9:09 AM
To:	Laney, Ricky A
Subject:	[PyNet] Learning Python: Week5 (Review and Exercises)

Ricky

This week's class is a review class. There is a single video where I review the content from classes 1 through 4. Additionally, there are a couple of extended exercises.

Finally, since we are midway through the course, I have a couple of course feedback questions.

Course Feedback Questions:

1. What one things needs most improved in the course?

2. Has the course been valuable to you?

Videos

I. Class Review (weeks 1 - 4)
video https://vimeo.com/121951502
Length is 16 minutes

Exercises

In these two exercises, I am going to use the following diagram and CDP data:
Diagram
CDP data

Disclaimer: the CDP data is from a test switch and a test router. I have manually modified this data to be consistent with the above diagram.

Reference code for these exercises is posted at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class5

1. Parse the CDP data (see link above) to obtain the following information: hostname, ip, model, vendor, and device_type (device_type will be either 'router', 'switch', or 'unknown').

From this data create a dictionary of all the network devices.

The network_devices dictionary should have the following format:

```
network_devices = {
    'SW1': { 'ip': '10.1.1.22', 'model': 'WS-C2950-24', 'vendor': 'cisco', 'device_type': 'switch' },
    'R1': { 'ip': '10.1.1.1', 'model': '881', 'vendor': 'Cisco', 'device_type': 'router' },
     ...
    'R5': { 'ip': '10.1.1.1', 'model': '881', 'vendor': 'Cisco', 'device_type': 'router' },
 }
```

Note, this data structure is a dictionary that contains additional dictionaries. The key to the outer dictionary is a hostname and the data corresponding to this key is another dictionary. For example, for 'R1':

```
>>> network_devices['R1']
{'ip': '10.1.1.1', 'model': '881', 'vendor': 'Cisco', 'device_type': 'router'}
```

You can access a given attribute in the inner dictionary as follows:

```
>>> a_dict['R1']['ip']
'10.1.1.1'
```

If this is confusing, you might want to experiment with it in the Python shell:

```
##### Python Shell - experimenting with dictionary of dictionaries #####

# Initialize network_devices to be a blank dictionary
>>> network_devices = {}

# Assign the key 'R1' to network_devices using a value of a blank dictionary (in other words, I
am adding a key:value pair to network_devices where the key is 'R1' and the value is {} )
>>> network_devices['R1'] = {}

# Look at network_devices at this point
>>> network_devices
{'R1': {}}

# Add the 'ip' and 'vendor' fields to the inner dictionary
>>> network_devices['R1']['ip'] = '10.1.1.1'
>>> network_devices['R1']['vendor'] = 'Cisco'
>>> network_devices
{'R1': {'ip': '10.1.1.1', 'vendor': 'Cisco'}}

##### Python Shell - experimenting end #####
```

For the output to this exercise, print network_devices to standard output. Your output should look similar to the following (six network devices with their associated attributes).

```
{'R1': {'device_type': 'Router',
        'ip': '10.1.1.1',
        'model': '881',
        'vendor': 'Cisco'},
 'R2': {'device_type': 'Router',
        'ip': '10.1.1.2',
        'model': '881',
        'vendor': 'Cisco'},
 'R3': {'device_type': 'Router',
        'ip': '10.1.1.3',
        'model': '881',
        'vendor': 'Cisco'},
 'R4': {'device_type': 'Router',
        'ip': '10.1.1.4',
        'model': '881',
        'vendor': 'Cisco'},
 'R5': {'device_type': 'Router',
        'ip': '10.1.1.5',
        'model': '881',
        'vendor': 'Cisco'},
 'SW1': {'device_type': 'Switch',
         'ip': '10.1.1.22',
         'model': 'WS-C2950-24',
         'vendor': 'cisco'}}
```

2. Create a second program that expands upon the program from exercise 1.

This program will keep track of which network devices are physically adjacent to each other (physically connected to each other).

You will accomplish this by adding a new field (adjacent_devices) to your network_devices inner dictionary. adjacent_devices will be a list of hostnames that a given network device is physically connected to.

For example, the dictionary entries for 'R1' and 'SW1' should look as follows:

```
'R1': {'IP': '10.1.1.1',
        'adjacent_devices': ['SW1'],
        'device_type': 'Router',
        'model': '881',
        'vendor': 'Cisco'},

 'SW1': {'IP': '10.1.1.22',
         'adjacent_devices': ['R1', 'R2', 'R3', 'R4', 'R5'],
         'device_type': 'Switch',
         'model': 'WS-C2950-24',
         'vendor': 'cisco'},
```

For the output to this exercise, print network_devices to standard output. Your output should look similar to the following (six network devices with their associated attributes including 'adjacent_devices').

```
{'R1': {'IP': '10.1.1.1',
        'adjacent_devices': ['SW1'],
        'device_type': 'Router',
        'model': '881',
        'vendor': 'Cisco'},
 'R2': {'IP': '10.1.1.2',
        'adjacent_devices': ['SW1'],
        'device_type': 'Router',
        'model': '881',
        'vendor': 'Cisco'},
 'R3': {'IP': '10.1.1.3',
        'adjacent_devices': ['SW1'],
        'device_type': 'Router',
        'model': '881',
        'vendor': 'Cisco'},
 'R4': {'IP': '10.1.1.4',
        'adjacent_devices': ['SW1'],
        'device_type': 'Router',
        'model': '881',
        'vendor': 'Cisco'},
 'R5': {'IP': '10.1.1.5',
        'adjacent_devices': ['SW1'],
        'device_type': 'Router',
        'model': '881',
        'vendor': 'Cisco'},
 'SW1': {'IP': '10.1.1.22',
         'adjacent_devices': ['R1', 'R2', 'R3', 'R4', 'R5'],
         'device_type': 'Switch',
         'model': 'WS-C2950-24',
         'vendor': 'cisco'}}
```

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers

Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops
Passing Arguments into a Script

Week4
Introduction
While Loops
Dictionaries
Exceptions

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, February 23, 2016 12:01 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Python and SNMP Introduction

Ricky

This is my periodic newsletter on network automation. This content is not a part of my Learning Python email course.

In this article, I briefly introduce Python and SNMP using the pysnmp library.

I assume that you already have some knowledge on SNMP including MIBs and OIDs. If not, you should be able to find this information fairly easily on the Internet.

One resource that I found particularly helpful for working out OIDs was the Cisco SNMP Object Navigator

In order to get started, you need to install the pysnmp library. For context, I am testing on an AWS AMI server (RedHat based i.e. yum instead of apt).

For easy installation just use 'pip' (assuming that you have it installed):

```
# As root or sudo
$ pip install pysnmp
```

Alternatively, you can manually install the library:

```
$ wget http://downloads.sourceforge.net/project/pysnmp/pysnmp/4.2.5/pysnmp-4.2.5.tar.gz
$ gunzip pysnmp-4.2.5.tar.gz
$ tar -xvpf pysnmp-4.2.5.tar
$ cd pysnmp-4.2.5
$ python setup.py install
```

I also installed net-snmp to simplify testing and to add an easy way to perform an SNMP walk. The installation method for net-snmp will vary depending on your system.

```
# As root or sudo
$ yum install net-snmp
$ yum install net-snmp-utils
```

To keep things simple I am only going to use SNMPv1/2c (i.e. this article does not cover SNMPv3). This is obviously not secure.

Now that I have the pysnmp library installed, the next step is to verify that I can communicate with my test router using SNMP. First, let's test this directly from the Linux command line:

```
$ snmpget -v 2c -c <COMMUNITY> <IP_ADDR> .1.3.6.1.2.1.1.1.0
SNMPv2-MIB::sysDescr.0 = STRING: Cisco IOS Software, C880 Software (C880DATA-
UNIVERSALK9-M), Version 15.0(1)M4, RELEASE SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2010 by Cisco Systems, Inc.
Compiled Fri 29-Oct-10 00:02 by prod_rel_team
```

The OID .1.3.6.1.2.1.1.1.0 is the MIB-2 sysDecr object. During testing I had multiple problems getting SNMP to work on the router including that I had Cisco's Control Plane Policing enabled (ooops) and that I needed to allow access through both an edge ACL and through a separate SNMP ACL.

At this point, I am able to communicate using SNMP from my AWS server to my test router.

Now let's try the same thing except using the pysnmp library. In order to simplify this I have created a couple of SNMP helper functions see:

https://github.com/ktbyers/pynet/tree/master/snmp/snmp_helper.py

First we need to do some initialization:

```
>>> from snmp_helper import snmp_get_oid,snmp_extract
>>>
>>> COMMUNITY_STRING = '<COMMUNITY>'
>>> SNMP_PORT = 161
>>> a_device = ('1.1.1.1', COMMUNITY_STRING, SNMP_PORT)
```

This code loads my two functions (snmp_get_oid and snmp_extract); it also creates a tuple named 'a_device' consisting of an IP, community string, and port 161.

I then call my snmp_get_oid function using the OID of MIB-2 sysDescr:

```
>>> snmp_data = snmp_get_oid(a_device, oid='.1.3.6.1.2.1.1.1.0', display_errors=True)
>>> snmp_data
[(MibVariable(ObjectName(1.3.6.1.2.1.1.1.0)),
DisplayString(hexValue='436973636f20494f5320536f6674776172652c204338383020536f6674776172652
02843383830444154412d554e4956455253414c4b392d4d292c2056657273696f6e2031352e302831294d
342c2052454c4541534520534f4654574152452028666331290d0a546563686e6963616c20537570706f72
743a20687474703a2f2f7777772e636973636f2e636f6d2f74656368737570706f72740d0a436f7079726967
68742028632920313938362d3230313020627920436973636f2053797374656d732c20496e632e0d0a436
f6d70696c6564204672692032392d4f63742d31302030303a30322062792070726f645f72656c5f74656
16d'))]
```

I can see that I received SNMP data back albeit in an ugly format. I can now use the snmp_extract function to display the output in a more friendly way.

```
>>> output = snmp_extract(snmp_data)
>>> print output
Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Version 15.0(1)M4,
RELEASE SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2010 by Cisco Systems, Inc.
Compiled Fri 29-Oct-10 00:02 by prod_rel_team
```

Now, let's repeat this process but using a different OID. Using snmpwalk on the 'interfaces' MIB and the Cisco SNMP Object Navigator, I was able to determine that the OID = .1.3.6.1.2.1.2.2.1.16.5 corresponded to the output octets on FastEthernet4.

Here I query that OID a couple of times in fairly quick succession (less than a minute between queries):

```
>>> snmp_data = snmp_get_oid(a_device, oid='.1.3.6.1.2.1.2.2.1.16.5', display_errors=True)
>>> output = snmp_extract(snmp_data)
>>> print output
293848947

>>> snmp_data = snmp_get_oid(a_device, oid='.1.3.6.1.2.1.2.2.1.16.5', display_errors=True)
>>> output = snmp_extract(snmp_data)
>>> print output
293849796
```

You can see that the count incremented.

Hopefully, this article helps you get started with Python and SNMP.

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From: pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Thursday, February 18, 2016 9:22 AM
To: Laney, Ricky A
Subject: [PyNet] Learning Python: Week4 (While Loops, Dictionaries, and Exceptions)

Ricky

In this email of Learning Python we are going to cover the following:

I. **Introduction Week4**
video http://youtu.be/UJmDynoQxps
Length is 1 minute

II. **While Loops**
video http://youtu.be/xnZzrnAQdG0
Length is 7 minutes

III. **Dictionaries**
video http://youtu.be/iGVJmUXcLtI
Length is 14 minutes

IV. **Exceptions**
video http://youtu.be/fCkjmtIq7wU
Length is 18 minutes

Make sure that you set your video resolution in YouTube to at least 360p (480p or 720p are better assuming your Internet connection has sufficient bandwidth). The settings button is at the bottom of the video and looks like a little gear.

Additional content that you may be interested in:

Content on dictionaries from Google's Python class (skip the 'Files' section we haven't talked about this yet):
https://developers.google.com/edu/python/dict-files

Additional content on dictionaries from the, 'Dive Into Python' book:
http://www.diveintopython.net/native_data_types/

'Learn Python the Hard Way' has a section on dictionaries as well:
http://learnpythonthehardway.org/book/ex39.html

Exercises

Reference code for these exercises is posted at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class4

I. Prompt a user to input an IP address. Re-using some of the code from class3, exercise4--determine if the IP address is valid. Continue prompting the user to re-input an IP address until a valid IP address is input.

II. Parse the below 'show version' data and obtain the following items (vendor, model, os_version, uptime, and serial_number). Try to make your string parsing generic i.e. it would work for other Cisco IOS devices.

The following are reasonable strings to look for:

'Cisco IOS Software' for vendor and os_version
'bytes of memory' for model
'Processor board ID' for serial_number
' uptime is ' for uptime

Store these variables (vendor, model, os_version, uptime, and serial_number) in a dictionary. Print the dictionary to standard output when done.

Note, "Cisco IOS Software...Version 15.0(1)M4...(fc1)" is one line.

```
>>>>> show version data <<<<<
Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Version 15.0(1)M4, RELEASE
SOFTWARE (fc1)
Technical Support:
Copyright (c) 1986-2010 by Cisco Systems, Inc.
Compiled Fri 29-Oct-10 00:02 by prod_rel_team
ROM: System Bootstrap, Version 12.4(22r)YB5, RELEASE SOFTWARE (fc1)

twb-sf-881 uptime is 7 weeks, 5 days, 19 hours, 23 minutes
System returned to ROM by reload at 15:33:36 PST Fri Feb 28 2014
System restarted at 15:34:09 PST Fri Feb 28 2014
System image file is "flash:c880data-universalk9-mz.150-1.M4.bin"
Last reload type: Normal Reload
Last reload reason: Reload Command

Cisco 881 (MPC8300) processor (revision 1.0) with 236544K/25600K bytes of memory.
Processor board ID FTX1000038X

5 FastEthernet interfaces
1 Virtual Private Network (VPN) Module
256K bytes of non-volatile configuration memory.
```

```
126000K bytes of ATA CompactFlash (Read/Write)

License Info:
License UDI:
-------------------------------------------------
Device#   PID                 SN
-------------------------------------------------
*0       CISCO881-SEC-K9      FTX1000038X

License Information for 'c880-data'
    License Level: advipservices   Type: Permanent
    Next reboot license Level: advipservices

Configuration register is 0x2102
>>>>> end <<<<<
```

III. Create a program that converts the following uptime strings to a time in seconds.

```
uptime1 = 'twb-sf-881 uptime is 6 weeks, 4 days, 2 hours, 25 minutes'
uptime2 = '3750RJ uptime is 1 hour, 29 minutes'
uptime3 = 'CATS3560 uptime is 8 weeks, 4 days, 18 hours, 16 minutes'
uptime4 = 'rtr1 uptime is 5 years, 18 weeks, 8 hours, 23 minutes'
```

For each of these strings store the uptime in a dictionary using the device name as the key.

During this conversion process, you will have to convert strings to integers. For these string to integer conversions use try/except to catch any string to integer conversion exceptions.

For example:
int('5') works fine
int('5 years') generates a ValueError exception.

Print the dictionary to standard output.

Class Outline

    B. How to create dictionaries.
    C. Characteristics of dictionaries
        1. Order is deterministic, but not predictable by you
        2. Keys must be unique
        3. Keys are usually strings
    D. Dictionary operations
        1. Assignment
        2. del
        3. Trying to access a non-existent key
        4. .get() method
        5. Checking if a key is in a dictionary
    E. Iterating over dictionaries
        1. .keys()
        2. .values()
        3. .items()

IV. Exceptions
    A. What happens when your program encounters an error?
    B. How to more gracefully handle errors - try / except
        1. Syntax and structure - example
        2. Catching multiple exception types
        3. except SomeException as e syntax
    C. Things to be careful of

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Week3
Introduction
If Conditionals
For Loops

Passing Arguments into a Script

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, February 16, 2016 3:00 PM
To:	Laney, Ricky A
Subject:	[PyNet] - Python, Paramiko SSH, and Network Devices

Ricky

You have been learning Python, but as a network engineer what can you do with it?

In this article, I will show you how to use Paramiko SSH (a Python SSH library) to connect to and gather information from a router. In a later article (potentially multiple articles), I will expand upon this--showing you how to gather information from multiple devices, and how to make configuration changes.

Now in this article I will be using Python to connect to an interface that is inherently expecting a human being (i.e. using an 'Expect-like' method). For various reasons, this Expect-like method is problematic--for example, there can be timing issues; there can be unexpected interaction problems; and there can be variations in device output (between OS versions, between device models, between device types, etc.).

While this Expect-like method is problematic, in many cases, it is a reasonable option for automation of existing equipment (given currently available alternatives). This will change across time as newer APIs (and other management/control mechanisms) become available and as devices in the field get refreshed and upgraded. In my opinion, the difficulties of using Expect-like mechanisms are also overstated (don't get me wrong there are a lot of difficulties, but APIs are not all a bed of roses either).

As with many things, a Python SSH 'Expect-like' method can be made to work in a reasonable way provided that you know its limitations; you test it appropriately; and you take appropriate precautions when using it.

One final warning--and this is a warning about network automation in general. Programmatically controlling a set of network devices is a powerful capability and because it is powerful the stakes are significantly increased. You potentially might be changing fifty devices instead of one; you could be changing devices in multiple geographies. The consequences of mistakes are greatly increased. Consequently, start small. Start with show commands instead of configuration changes. When making changes practice good operational procedures--know what you are doing; test it out appropriately in a test environment; deploy the change to a single device; expand it to a small set of devices; consider the consequences if things go wrong and how you will correct them; determine whether this change is something that makes sense to do programmatically.

So on that warm and fuzzy note, let's dive-in.

Read the rest of the article at:

https://pynet.twb-tech.com/blog/python/paramiko-ssh-part1.html

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From: pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Thursday, February 11, 2016 9:18 AM
To: Laney, Ricky A
Subject: [PyNet] Learning Python: Week3 (Conditionals and For Loops)

Ricky

In this email of Learning Python we are going to cover the following:

I. **Introduction Week3**
video https://vimeo.com/120754390
Length is 2 minutes

II. **If Conditionals**
video http://youtu.be/5lHWv9hkSn8
Length is 17 minutes

III. **For Loops**
video http://youtu.be/VR0ggQuCIOM
Length is 15 minutes

IV. **Passing Arguments into a Script**
video http://youtu.be/mUt0uJmD9y4
Length is 4 minutes

Make sure that you set your video resolution in YouTube to at least 360p (480p or 720p are better assuming your Internet connection has sufficient bandwidth). The settings button is at the bottom of the video and looks like a little gear.

Additional content that you may be interested in

Note, I made a comment at the end of the for loop video that could be misconstrued. You cannot use 'break' or 'continue' with if/else conditionals; you can use 'break' or 'continue' with both for and while loops.

Python2 vs Python3 (relevant differences for beginners):
http://www.cs.carleton.edu/faculty/jgoldfea/cs201/spring11/Python2vs3.pdf

Darren O'Connor has a recent blog post about passing arguments into a Python script (Darren is a dual-CCIE, JNCIE who has some blog posts about Python):
http://mellowd.co.uk/ccie/?p=5126

For more complicated argument parsing check out Argparse (requires Python 2.7) or getopt:
https://docs.python.org/2/howto/argparse.html

Exercises

Reference code for these exercises is posted on GitHub at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class3

Note, some of the exercises use lengthy strings that get modified when embedded in HTML. You can also grab the exercise strings from the above GitHub site. The strings should be embedded in the relevant exercise.

I. Create an IP address converter (dotted decimal to binary). This will be similar to what we did in class2 except:

A. Make the IP address a command-line argument instead of prompting the user for it.

```
./binary_converter.py 10.88.17.23
```

B. Simplify the script logic by using the flow-control statements that we learned in this class.

C. Zero-pad the digits such that the binary output is always 8-binary digits long. Strip off the leading '0b' characters. For example,

```
OLD:    0b1010
NEW:    00001010
```

D. Print to standard output using a dotted binary format. For example,

```
 IP address        Binary
10.88.17.23       00001010.01011000.00010001.00010111
```

Note, you might need to use a 'while' loop and a 'break' statement for part C.

```
while True:
    ...
    break       # on some condition (exit the while loop)
```

Python will execute this loop again and again until the 'break' is encountered.

II. Modify the 'show ip bgp' exercise from class2. Simplify the program using the flow-control statements from this class.

III. You have the following 'show ip int brief' output.

```
show_ip_int_brief = '''
Interface          IP-Address      OK?    Method      Status     Protocol
FastEthernet0   unassigned     YES    unset        up          up
FastEthernet1   unassigned     YES    unset        up          up
FastEthernet2   unassigned     YES    unset        down       down
FastEthernet3   unassigned     YES    unset        up          up
FastEthernet4    6.9.4.10        YES    NVRAM      up          up
NVI0                6.9.4.10        YES    unset        up          up
Tunnel1           16.25.253.2   YES    NVRAM      up          down
Tunnel2           16.25.253.6   YES    NVRAM      up          down
Vlan1              unassigned     YES   NVRAM      down       down
Vlan10            10.220.88.1   YES    NVRAM      up          up
Vlan20            192.168.0.1   YES    NVRAM      down       down
Vlan100          10.220.84.1   YES    NVRAM      up          up
'''
```

From this output, create a list where each element in the list is a tuple consisting of (interface_name, ip_address, status, protocol). Only include interfaces that are in the up/up state.

Print this list to standard output.

IV. Create a script that checks the validity of an IP address. The IP address should be supplied on the command line.

A. Check that the IP address contains 4 octets.
B. The first octet must be between 1 - 223.
C. The first octet cannot be 127.
D. The IP address cannot be in the 169.254.X.X address space.
E. The last three octets must range between 0 - 255.

For output, print the IP and whether it is valid or not.

CLASS OUTLINE

I. Introduction

II. Conditionals (if / else)
- A. Expanded meaning of true and false
- B. Comparison operators (==, !=, >, >=, <, <=)
- C. If statements - syntax and structure
- D. Nested if/else statements

III. For loops
- A. Iterating over a list
- B. Tracking indexes
    1. Method1 - manually tracking index
    2. Method2 - using range(len(a_list))
    3. Method3 - enumerate
- C. Continue, break, and pass

IV. Passing arguments to a script

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Week2
Introduction
Print and raw_input
Numbers
Lists and Tuples
Booleans

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Tuesday, February 9, 2016 9:04 AM
To:	Laney, Ricky A
Subject:	[PyNet] - Network Config Templating using Ansible

Ricky

This is my periodic newsletter on network automation. This content is not a part of my Learning Python email course.

Network Config Templating using Ansible

Earlier this year, I wrote a three-part article on using Ansible for network configuration templating.

The general problem is this--you want a systematic way of creating network device configurations based on templates and variables.

As background--Ansible is an open-source automation application that can be used to automate many tasks in your environment (predominantly compute and cloud tasks). Ansible can also generate files based on Jinja2 templates and variables. Jinja2 is a widely-used Python templating system, see http://jinja.pocoo.org/ for more information about Jinja2.

Thus, Ansible has all the components for network configuration templating built into it. This is a very good initial use case for Ansible for network engineers.

Part1 - https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template.html

Part2 - https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template-p2.html

Part3 - https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template-p3.html

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From:	pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Thursday, February 4, 2016 9:15 AM
To:	Laney, Ricky A
Subject:	[PyNet] Learning Python: Week2 (Printing, Numbers, and Lists)

Ricky

In this email of Learning Python we are going to cover the following:

I. **Introduction Week2**
video http://youtu.be/uqGZXPfX00E
Length is 2 minutes

II. **Print and raw_input**
video http://youtu.be/pEXUxySxygg
Length is 14 minutes

III. **Numbers**
video http://youtu.be/n5ZO8rRcWbA
Length is 12 minutes

IV. **Lists and Tuples**
video http://youtu.be/nUOMXXhQgZQ
Length is 14 minutes

V. **Booleans**
video http://youtu.be/gCpRBt7pw-0
Length is 5 minutes

Make sure that you set your video resolution in YouTube to at least 360p (480p or 720p are better assuming your Internet connection has sufficient bandwidth). The settings button is at the bottom of the video and looks like a little gear.

Additional content you may be interested in:

Packet Pushers podcast on 'Intro to Python & Automation for Network Engineers':

http://packetpushers.net/show-176-intro-to-python-automation-for-network-engineers/

Google's Python Class:

https://developers.google.com/edu/python/?csw=1

Humor:

https://www.python.org/doc/humor/#python-vs-perl-according-to-yoda

http://stackoverflow.com/questions/234075/what-is-your-best-programmer-joke

Q: How do you tell an introverted computer scientist from an extroverted computer scientist?

A: An extroverted computer scientist looks at your shoes when he talks to you.

Exercises:

Reference code for these questions is posted at:
https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class2

I. Create a script that does the following

A. Prompts the user to input an IP network.

Notes:
1. For simplicity the network is always assumed to be a /24 network

2. The network can be entered in using one of the following three formats 10.88.17.0, 10.88.17., or 10.88.17

B. Regardless of which of the three formats is used, store this IP network as a list in the following format ['10', '88', '17', '0'] i.e. a list with four octets (all strings), the last octet is always zero (a string).

Hint: There is a way you can accomplish this using a list slice.

Hint2: If you can't solve this question with a list slice, then try using the below if statement (note, we haven't discussed if/else conditionals yet; we will talk about them in the next class).

```
>>>> CODE <<<<

if len(octets) == 3:
   octets.append('0')
elif len(octets) == 4:
   octets[3] = '0'

>>>> END <<<<
```

C. Print the IP network out to the screen.

D. Print a table that looks like the following (columns 20 characters in width):

```
NETWORK_NUMBER   FIRST_OCTET_BINARY     FIRST_OCTET_HEX
88.19.107.0      0b1011000              0x58
```

II. Create an IP address converter (dotted decimal to binary):

A. Prompt a user for an IP address in dotted decimal format.

B. Convert this IP address to binary and display the binary result on the screen (a binary string for each octet).

```
Example output:
first_octet    second_octet    third_octet    fourth_octet
0b1010         0b1011000       0b1010         0b10011
```

III. You have the following four lines from 'show ip bgp':

```
entry1 = "*  1.0.192.0/18   157.130.10.233        0 701 38040 9737 i"
entry2 = "*  1.1.1.0/24     157.130.10.233        0 701 1299 15169 i"
entry3 = "*  1.1.42.0/24    157.130.10.233        0 701 9505 17408 2.1465 i"
entry4 = "*  1.0.192.0/19   157.130.10.233        0 701 6762 6762 6762 6762 38040 9737 i"
```

Note, in each case the AS_PATH starts with '701'.

Using split() and a list slice, how could you process each of these such that--for each entry, you return an ip_prefix and the AS_PATH (the ip_prefix should be a string; the AS_PATH should be a list):

Your output should look like this:

```
ip_prefix         as_path
1.0.192.0/18      ['701', '38040', '9737']
1.1.1.0/24        ['701', '1299', '15169']
1.1.42.0/24       ['701', '9505', '17408', '2.1465']
1.0.192.0/19      ['701', '6762', '6762', '6762', '6762', '38040', '9737']
```

Ideally, your logic should be the same for each entry (I say this because once I teach you for loops, then I want to be able to process all of these in one four loop).

If you can't figure this out using a list slice, you could also solve this using pop().

IV. You have the following string from "show version" on a Cisco router

```
cisco_ios = "Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Version
15.0(1)M4, RELEASE SOFTWARE (fc1)"
```

Note, the string is a single line; there is no newline in the string.

How would you process this string to retrieve only the IOS version:

```
ios_version = "15.0(1)M4"
```

Try to make it generic (i.e. assume that the IOS version can change).

You can assume that the commas divide this string into four sections and that the string will always have 'Cisco IOS Software', 'Version', and 'RELEASE SOFTWARE' in it.

<u>CLASS OUTLINE</u>

        3. remove()
        4. del
        5. range()
    C. Slices
        1. The issue with copying lists
        2. Slice notation
    D. Tuples

V. Booleans
    A. Boolean operators

Video Archive

Week1
Introduction and Some Questions
What is the Nature of Python
Interpreter Shell, Variables, and Assignment
Strings

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From: pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Friday, January 29, 2016 5:45 AM
To: Laney, Ricky A
Subject: [PyNet] Learning Python: Week1 (Preliminaries and Strings)

Ricky

In this email of Learning Python we are going to cover the following:

I. **Introduction** and **Some Questions**
video **https://vimeo.com/119478712**
Length is 11 minutes
The 'Some Questions' section continues into the first 1:12 of the next video

II. **Characteristics of Python**
video **https://vimeo.com/119480935**
Length is 7 minutes

III. **Interpreter Shell, Variables, and Assignment**
video **http://youtu.be/6ja4KlejT-g**
Length is 8 minutes

IV. **Strings**
video **http://youtu.be/ItIE8hItji8**
Length is 19 minutes

Note, you might need to install Flash Player to get the videos to work. I had to do this on FireFox on MacOs.

Make sure that you set your video resolution in YouTube to at least 360p (480p or 720p are better assuming your Internet connection has sufficient bandwidth). The settings button is at the bottom of the video and looks like a little gear.

Additional content that might be helpful:

Jeremy Stretch had a good article on 'Learn Python' on packetlife.net:
http://packetlife.net/blog/2014/mar/27/learn-python/

Chris Young has a page listing various Python resources for network engineers:
http://kontrolissues.net/python-for-network-engineers-resources/

The book, 'Learn Python the Hard Way' is a good resource for individuals that are

completely new to programming:
    http://learnpythonthehardway.org/book/

The book 'Treading on Python, Vol 1: Foundations" by Matt Harrison is a very good resource for individuals learning Python:
    http://www.amazon.com/Treading-Python-1-Foundations/dp/1475266413/

Additional content related to the videos:

1. Python naming conventions:

a. For variable names, function names, object names, and module names use lower case separated by underscore, for example:

```
my_router
find_set_of_devices
convert_id_string_to_list
```

b. For class names, capitalize the first letter of each word. Do not use any underscores. For example:

```
ManyToManyField
ClientHistory
UserProfile
```

c. For constants, use all upper case; use underscores for word separation.

```
PI = 3.14
EMAIL_MODE
EMAIL_FROM_ADDRESS
```

2. Indentation:

I stated on the video that indentation matters in Python, but I did not provide any examples. Here is some example code showing you indentation. We haven't talked about 'if conditionals' so this code is just to show you an example of indentation.

```
>>>>> CODE <<<<<

a = 10
b = 20

if a == 10:
    print 'First level of indentation'

    if b == 20:
        print 'Second level of indentation'
        print 'Be consistent in your indentation'
        print 'Use four spaces for each level'

    print 'I am now out of the second if statement'

print 'I am now out of the first if statement'

>>>>> END <<<<<
```

Note, the first if statement is terminated by a colon; after the colon you see some indented Python statements. Everything at this first indentation level is part of the first block (i.e. it will be executed if the first 'if' statement is True).

The second if statement starts a new block of code (and a new indentation level). Once again the if statement is terminated by a colon.

You indicate the end of each section (block of code) by ending the indentation.

3. Comments

I also didn't mention in the videos that you can indicate a comment by using the # character (all the characters after the pound sign are ignored).

```
# This is a comment

a = 10        # and so is this
```

4. How to pipe <stdin> into a Python script (Linux and MacOS)

Python supports a way that you can pipe data into it (on Linux and MacOS). You can then process the data you sent into it line-by-line (see below).

Note, some of these statements we haven't talked about yet (import and for loops). Consequently, if this doesn't make any sense, just skip this section.

```
>>>>> CODE (file test3.py) <<<<<

#!/usr/bin/env python

import fileinput

for line in fileinput.input():
    print line.split(".")

>>>>> END <<<<<
```

Here is an example using this script where I echo an IP address into it and then the IP address is split into its octets.

```
$ echo '192.168.1.1'  | ./test3.py
['192', '168', '1', '1\n']
```

Some Exercises

Reference code for these exercises is posted on GitHub at:

https://github.com/ktbyers/pynet/tree/master/learnpy_ecourse/class1

1. Use the split method to divide the following IPv6 address into groups of 4 hex digits (i.e. split on the ":")

FE80:0000:0000:0000:0101:A3EF:EE1E:1719

2. Use the join method to reunite your split IPv6 address back to its original value.

3. In the test3.py script above, how would you modify this to remove the trailing newline on the end of 192.168.1.1?

CLASS OUTLINE

I. Introduction
  A. About Me
  B. Purpose of the course
  C. Logistics
  D. Feedback
  E. Balance for various levels of existing skills

II. Some Questions
  A. Why learn programming at all?
  B. Why Python?
  C. Python2 vs Python3

III. What is the nature of Python?
  A. The Zen of Python (try "import this" at the Python interpreter shell)

IV. Interpreter Shell, Variables, and Assignment
  A. Enough with all of the preliminaries, let's get to some code
  B. Lauching the Interpreter Shell

        1. History
        2. Cntl-a, Cntl-e (home/end on Windows)
        3. Cntl-d or exit()
    C. What is the Interpreter Shell / what is it useful for?
    D. Variables and assignment
        1. Variable names can include alphanumeric and underscore, but can't begin with a number
        2. Variable names are case-sensitive
        3. Variables beginning with _something, __something, or __something__ are special; avoid creating variables beginning with an underscore (for now).

V. Strings.
    A. Dynamically Typed Language
    B. String assignment - single quotes, double quotes, triple quotes
    C. Characteristics of strings
        1. Accessing individual letters
        2. Strings are immutable
    D. Raw strings
    E. Unicode strings
    F. String methods
        1. strip()
        2. split()
        3. join()
    G. Using help() and dir()
    H. Substring in String

Kirk Byers
https://pynet.twb-tech.com
Twitter: @kirkbyers

Laney, Ricky A

From: pynet-beg-cl1@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Friday, January 29, 2016 5:41 AM
To: Laney, Ricky A
Subject: [PyNet] Learning Python Course

Ricky

The next Learning Python course starts on January 28th. You should receive the first class on that day.

A lot of people taking this course will fall into one of the three following categories. Here are some recommendations if this is you.

Case1 - You have never programmed before

I strongly recommend that you check out Learn Python the Hard Way. It is a good resource for individuals that are new to programming and it is free online. Note, the learning/teaching style of Learn Python the Hard Way is not for everyone.

You can then either participate in my course after you have completed some of LPTHW or in parallel with my course.

Case2 - Network engineer with some programming familiarity

You should be good to go for this course.

Case3 -- You are a programmer with extensive experience

This course is probably not for you. It will be too slow and cover too many basics.

Python2 vs Python3

This course uses Python2.

The differences between Python2 and Python3 (in the context of this course) are relatively minor, however. Consequently, if you are already somewhat familiar with Python3, then you can probably use it for this course without too much difficulty.

Note, in the first class I discuss why I recommend Python2 for network engineers.

Getting Python Installed

You will need a system that has Python2.7 on it.

MacOS X
If you are running a recent version of MacOS X, then you will already have Python 2.7 (launch Terminal and type 'python' at the prompt).

Linux
Just type 'python' from the shell and verify it has Python2.7.

Windows
If you are running on Windows, then you should be able to download and install it from here:

https://www.python.org/downloads/release/python-2711/

Either the 'Windows x86-64 MSI Installer' or the 'Windows x86 MSI Installer' is probably the right choice.

You might also want to update your Windows system Path so that your computer knows how to find the Python interpreter. You should be able to find this process online (for your version of Windows).

Web

You can also use an online Python Interpreter (at least for some of the course). You can find one online at http://repl.it/languages/Python.

Regards,

Kirk Byers
Twitter: @kirkbyers
https://pynet.twb-tech.com

Laney, Ricky A

From:	pynet@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent:	Friday, January 29, 2016 5:41 AM
To:	Laney, Ricky A
Subject:	PyNet email-list confirmed

Ricky

Thank you for joining my email-list.

I hope you enjoy the content.

Regards,

Kirk Byers

Laney, Ricky A

From: verifications@aweber.com on behalf of Kirk Byers <ktbyers@twb-tech.com>
Sent: Thursday, January 28, 2016 9:35 PM
To: Laney, Ricky A
Subject: CONFIRMATION NEEDED: PyNet - Learning Python Email Course

Please confirm that you want to receive the Python for Network Engineers, Learning Python Email Course.

CONFIRM BY VISITING THE LINK BELOW:

https://urldefense.proofpoint.com/v2/url?u=http-3A__www.aweber.com_z_c_-3Fad69473a-2Ddf6d-2D4051-2Dbf0a-2D74e9c0f58597&d=CwICaQ&c=B73tqXN8Ec0ocRmZHMCntw&r=N6hGSvAm0EA1cRt4oKpoqf5By74qiUfzqonLtR5k358&m=SIOyU8rcZcciMjUlwCq7LK00s7RveS5LlfZdrKim7eE&s=ejKCjMEAWbLjC0iI7zCze6BN7Fs_TUUaYcFMFdN5iX4&e=

Click the link above to give us permission to send you information. It's fast and easy! If you cannot click the full URL above, please copy and paste it into your web browser.

If you do not want to confirm, simply ignore this message.

Regards,

Kirk Byers

1235 Connecticut St
San Francisco CA 94107
USA

Request generated by:
IP: 127.0.0.1
Date: January 28, 2016 21:33 -0500
URL: