**Lab 1: The Basics**

***1.1 – Printing IPs***

**Objectives:**

* **Get accustomed to assigning variables**
* **Use print statements to output to terminal**

**Instructions:**

* **Create a Python script that has three variables:**
* **ip\_addr1**
* **ip\_addr2**
* **ip\_addr3**

**Which all represent corresponding IP addresses.**

* **Print all of them to standard output using a singular print statement**

***1.2 – IP Conversion***

**Objectives:**

* **Use the input() function to take in user Input**
* **Learn how to manipulate strings using string operators**
* **Use python's built in type conversion**

**Instructions:**

* **Prompt a user to enter an IP address from the command line.**
* **Break the IP down into individual octets and print each octet as a decimal, binary and hexadecimal.**

***1.3 – IP Comparison***

**Objectives:**

* **Compare strings**

**Instructions:**

* **Create 3 variables that represent different ipv6 addresses. Check to see if they equal each other.**

***1.4 – Serial Number Parsing***

**Objectives:**

* **Split strings to break them down and digest them more easily**
* **Get comfortable using string class methods**

**Instructions:**

* Create a show\_version variable that is equal to the following:

"\*0 CISCO881-SEC-K9 FTX0000038X "

* **Remove all the excess whitespace.**
* **Use the .split() method (of the string class) to extract both the model and serial number.**
* **Check to see if the model number contains the word 'cisco' (not case sensitive).**
* **Print the model number and serial number to the command line.**

***1.5 – Systematic Parsing***

**Objectives:**

* **Properly digest large strings**
* **Use string formatting to make your output easier to understand**

**Instructions:**

* **You have three variables from the arp table of a router:**

mac1 = "Internet 10.220.88.29 94 5254.abbe.5b7b ARPA FastEthernet4"

mac2 = "Internet 10.220.88.30 3 5254.ab71.e119 ARPA FastEthernet4"

mac3 = "Internet 10.220.88.32 231 5254.abc7.26aa ARPA FastEthernet4"

* **Isolate the IP address and MAC address of each listing and print them to console in an orderly manner (use the .format() method).**

**Lab 2: Lists and Files**

***2.1 – Reading Files***

**Objectives:**

* **Open and read files**

**Instructions:**

* Open the "show\_version.txt" file for reading. Use the .read() method to read in the entire file contents to a variable.
* Print out the file contents to the screen.
* Also print out the type of the variable (you should have a string at this point).
* Close the file.
* Open the file a second time using a Python context manager (with statement). Read in the file contents using the .readlines() method. Print out the file contents to the screen.
* Also print out the type of the variable (you should have a list at this point).

***2.2 – List Management***

**Objectives:**

* **Learn to navigate and access list data**
* **Mutate lists using list class methods**

**Instructions:**

* Create a list of IP addresses:

my\_ipaddress = ['192.168.1.1', '10.1.1.1', '172.16.31.254', '8.8.8.8', '8.8.4.4']

* Use the .append() method to add an IP address onto the end of the list.
* Use the .extend() method to add two more IP addresses to the end of the list.
* Use list concatenation to add two more IP addresses to the end of the list.
* Print out the entire list of IP addresses.
* Print out the first IP address in the list.
* Print out the last IP address in the list.
* Use the .pop() method to remove the first IP address in the list and the last IP address in the list.
* Update the new first IP address in the list to be '2.2.2.2'. Print out the new first IP address in the list.
* Verify the contents of your list by printing the entire list.

***2.3 – File Parsing***

**Objectives:**

* **Read a file**
* **Parse ARP information**
* **Write desired information to a new file**

**Instructions:**

* Read in the "show\_arp.txt" file using the .readlines() method.
* Remove the header line using a list slice.
* Use pretty print to print out the resulting list to the screen, syntax is:

from pprint import pprint

pprint(some\_var)

* Use the list .sort() method to sort the list based on IP addresses.
* Create a new list slice that is only the first three ARP entries.
* Use the .join() method to join these first three ARP entries back together as a single string using the newline character ('\n') as the separator.
* Write this string containing the three ARP entries out to a file named "arp\_entries.txt".

***2.4 – Locate Peer IP***

**Objectives:**

* **Break files down into smaller, more manageable parts**
* **Isolate data**

**Instructions:**

* Read the 'show\_ip\_bgp\_summ.txt' file into your program. From this BGP output obtain the first and last lines of the output.
* From the first line use the string .split() method to obtain the local AS number.
* From the last line use the string .split() method to obtain the BGP peer IP address.
* Print both local AS number and the BGP peer IP address to the screen.

**Lab 3: Loops**

***3.1 – Nested Sequences***

**Objectives:**

* **Utilize for loops to perform actions on every item of a list**
* **Understand nested data**

**Instructions:**

* Read the "show\_vlan.txt" file into your program.
* Loop through the lines in this file and extract all of the VLAN\_ID, VLAN\_NAME combinations.
* From these VLAN\_ID and VLAN\_NAME construct a new list where each element in the list is a tuple consisting of (VLAN\_ID, VLAN\_NAME).
* Print this data structure to the screen. Your output should look as follows:

[('1', 'default'),

('400', 'blue400'),

('401', 'blue401'),

('402', 'blue402'),

('403', 'blue403')]

***3.2 – Find the default IP***

**Objectives:**

* **More file parsing using for loops**
* **Manipulate loops with flow statements**

**Instructions:**

* Read the contents of the "show\_arp.txt" file. Using a for loop, iterate over the lines of this file.
* Process the lines of the file and separate out the ip\_addr and mac\_addr for each entry into a separate variable.
* Add a conditional statement that searches for '10.220.88.1'. If 10.220.88.1 is found, print out the string "Default gateway IP/Mac" and the corresponding IP address and MAC Address.
* Using a conditional statement, also search for '10.220.88.30'. If this IP address is found, then print out "Arista3 IP/Mac is" and the corresponding ip\_addr and mac\_addr.
* Keep track of whether you have found both the Default Gateway and the Arista3 switch.
* Once you have found both of these devices, 'break' out of the for loop.

***3.3 – Remote Information***

**Objectives:**

* **Isolate remote variables and utilize them**
* **control loop flow**

**Instructions:**

* Read the 'show\_lldp\_neighbors\_detail.txt' file.
* Loop over the lines of this file. Keep reading the lines until you have encountered the remote "System Name" and remote "Port id".
* Save these two items into variables and print them to the screen. You should extract only the system name and port id from the lines (i.e. your variables should only have 'twb-sf-hpsw1' and '15').
* Break out of your loop once you have retrieved these two items.

***3.4 – MAC address formatting***

**Objectives:**

* **Manipulate nested data**

**Instructions:**

* You have the following data structure:

arp\_table = [('10.220.88.1', '0062.ec29.70fe'),

('10.220.88.20', 'c89c.1dea.0eb6'),

('10.220.88.21', '1c6a.7aaf.576c'),

('10.220.88.28', '5254.aba8.9aea'),

('10.220.88.29', '5254.abbe.5b7b'),

('10.220.88.30', '5254.ab71.e119'),

('10.220.88.32', '5254.abc7.26aa'),

('10.220.88.33', '5254.ab3a.8d26'),

('10.220.88.35', '5254.abfb.af12'),

('10.220.88.37', '0001.00ff.0001'),

('10.220.88.38', '0002.00ff.0001'),

('10.220.88.39', '6464.9be8.08c8'),

('10.220.88.40', '001c.c4bf.826a'),

('10.220.88.41', '001b.7873.5634')]

* Loop over this data structure and extract all of the MAC addresses.
* Process all of the MAC addresses to get them into a standard format.
* Print all of the new standardized MAC address to the screen. The standardized format should be as follows:  
    
  00:62:EC:29:70:FE
* The hex digits should be capitalized. Additionally, there should be a colon between each octet in the MAC address.

**Lab 4: Regular Expressions**

***4.1 – Organizing your data***

**Objectives:**

* **Use dictionaries to organize data**
* **Learn to iterate over dictionaries**

**Instructions:**

* Create a dictionary representing a network device. The dictionary should have key-value pairs representing the 'ip\_addr', 'vendor', 'username', and 'password' fields.
* Print out the 'ip\_addr' key from the dictionary.
* If the 'vendor' key is 'cisco', then set the 'platform' to 'ios'. If the 'vendor' key is 'juniper', then set the 'platform' to 'junos'.
* Create a second dictionary named 'bgp\_fields'. The 'bgp\_fields' dictionary should have a keys for 'bgp\_as', 'peer\_as', and 'peer\_ip'.
* Using the .update() method add all of the 'bgp\_fields' dictionary key-value pairs to the network device dictionary.
* Using a for-loop, iterate over the dictionary and print out all of the dictionary keys.
* Using a single for-loop, iterate over the dictionary and print out all of the dictionary keys and values.

***4.2 – Checking IP overlap***

**Objectives:**

* **Understand what sets can do for you**
* **Use set operations to your advantage**
* **Check to see what IP addresses are being used in different locations**

**Instructions:**

* Create three separate lists of IP addresses. The first list should be the IP addresses of the Houston data center routers, and it should have over ten RFC1918 IP addresses in it (including some duplicate IP addresses).
* The second list should be the IP addresses of the Atlanta data center routers, and it should have at least eight RFC1918 IP addresses (including some addresses that overlap with the Houston data center).
* The third list should be the IP addresses of the Chicago data center routers, and it should have at least eight RFC1918 IP addresses. The Chicago IP addresses should have some overlap with both the IP addresses in Houston and Atlanta.
* Convert each of these three lists to a set.
* Using a set operation, find the IP addresses that are duplicated between Houston and Atlanta.
* Using set operations, find the IP addresses that are duplicated in all three sites.
* Using set operations, find the IP addresses that are entirely unique in Chicago.

***4.3 – Acute Data Extraction***

**Objectives:**

* **Use regular expressions to recognize certain patterns**
* **Extract information using regular expressions**

**Instructions:**

* Read in the 'show\_version.txt' file. From this file, use regular expressions to extract the OS version, serial number, and configuration register values.
* Your output should look as follows:

OS Version: 15.4(2)T1

Serial Number: FTX0000038X

Config Register: 0x2102

***4.4 – Named Regular Expressions***

**Objectives:**

* **Use named regular expressions to parse information with precision**

**Instructions:**

* Using a named regular expression (?P<name>), extract the model from the below string:

show\_version = '''

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

Processor board ID FTX0000038X

5 FastEthernet interfaces

1 Virtual Private Network (VPN) Module

256K bytes of non-volatile configuration memory.

126000K bytes of ATA CompactFlash (Read/Write)

'''

* Note that, in this example, '881' is the relevant model. Your regular expression should not, however, include '881' in its search pattern since this number changes across devices.
* Using a named regular expression, also extract the '236544K/25600K' memory string.
* Once again, none of the actual digits of the memory on this device should be used in the regular expression search pattern.
* Print both the model number and the memory string to the screen.

***4.5 – More Regular Expression Practice***

**Objectives:**

* **Use multiple named regular expressions**

**Instructions:**

* Read the 'show\_ipv6\_intf.txt' file.
* From this file, use Python regular expressions to extract the two IPv6 addresses.
* The two relevant IPv6 addresses you need to extract are:

    2001:11:2233::a1/24

    2001:cc11:22bb:0:2ec2:60ff:fe4f:feb2/64

* Try to use re.DOTALL flag as part of your search. Your search pattern should not include any of the literal characters in the IPv6 address.
* From this, create a list of IPv6 addresses that includes only the above two addresses.
* Print this list to the screen.

**Lab 5: Writing Functions**  
  
***5.1 – SSH connection***

**Objectives:**

* **Define a function**
* **Get accustomed to calling function by different means**

**Instructions:**

a. Create an ssh\_conn function. This function should have three parameters: ip\_addr, username, and password. The function should print out each of these three variables and clearly indicate which variable it is printing out.

* Call this ssh\_conn function using entirely positional arguments.
* Call this ssh\_conn function using entirely named arguments.
* Call this ssh\_conn function using a mix of positional and named arguments.

b. Expand on the ssh\_conn function from exercise1 except add a fourth parameter 'device\_type' with a default value of 'cisco\_ios'. Print all four of the function variables out as part of the function's execution.

* Call the 'ssh\_conn2' function both with and without specifying the device\_type
* Create a dictionary that maps to the function's parameters. Call this ssh\_conn2 function using the \*\*kwargs technique.

***5.2 – IP Generation***

**Objectives:**

* **Write a function that randomly creates IP addresses**

**Instructions:**

* Create a function that randomly generates an IP address for a network. The default base network should be '10.10.10.'. For simplicity the network will always be a /24.
* You should be able to pass a different base network into your function as an argument.
* Randomly pick a number between 1 and 254 for the last octet and return the full IP address.
* You can use the following to randomly generate the last octet:

import random

random.randint(1, 254)

* Call your function using no arguments.
* Call your function using a positional argument.
* Call your function using a named argument.
* For each function call print the returned IP address to the screen.

***5.3 – Mac Address Formatting, part 2***

**Objectives:**

* **Write a function to convert Mac addresses to one format**
* **Test your function**

**Instructions:**

* Similar to 3.4, write a function that normalizes a MAC address to the following format:  
  01:23:45:67:89:AB
* This function should handle the lower-case to upper-case conversion.
* It should also handle converting from '0000.aaaa.bbbb' and from '00-00-aa-aa-bb-bb' formats.
* The function should have one parameter, the mac\_address. It should return the normalized MAC address
* Single digit bytes should be zero-padded to two digits. In other words, this:

a:b:c:d:e:f  
  
should be converted to:

0A:0B:0C:0D:0E:0F

* Write several test cases for your function and verify it is working properly.

***5.4 – Debugging***

**Objectives:**

* **Get familiar with using the Python debugger and what you can do with it**

**Instructions:**

Copy your solution from exercise 5.3 to exercise 5.4. Add an 'import pdb' and pdb.set\_trace() statement outside of your function (i.e. where you have your function calls).  
  
Inside of pdb, experiment with:

* Listing your code.
* Using 'next' and 'step' to walk through your code. Make sure you understand the difference between next and step.
* Experiment with 'up' and 'down' to move up and down the code stack.
* Use p <variable> to inspect a variable.
* Set a breakpoint and run your code to the breakpoint.
* Use '!command' to change a variable (for example !new\_mac = [])

**Use Case Labs:**

***1 – The basics:***

Start off by asking the user a series of questions to simulate scanning an RFID.

This is obviously not how an RFID is scanned, but it is useful to practice the basics.

On top of that, if they design a script that takes user input (like for UI for example),

This will help them understand how to do that

INSTRUCTIONS:

1. Ask user for a product name
2. Ask user for a location
3. Ask user for a manufacturer ID
4. Output all of this in an orderly manner

GOALS:

* Understand variable assignment
* Know how to take user input
* Displaying information to the terminal in a way that makes sense

***2 – Lists:***

Given a list of Products that a company has on hand, see if a newly added RFID would require a new entry

INSTRUCTIONS:

1. Create a list with 5 entries. All should be various product names
2. Check to see if the added RFID product name is in the list
   1. If it is, confirm with a statement and display it's index
   2. If it's not, add it to the list
3. Print your list to verify its contents

OBJECTIVES:

* Know how to create a list
* Know how to add to that list
* Check the contents of a list

***3 - Dictionaries:***

Modify your "scan" code so that instead of writing the user inputs to three separate variables,

you instead input it into a dictionary. Create a list of RFID objects and then add your new RFID

to the list. Print the list to verify its contents

Instructions:

1. Create a list of RFID dictionaries. Each should have keys for the product name, location and manufacturer ID with corresponding values.
2. Reformat your code for "scanning" the RFID so that it assigns the input values to a new RFID dictionary. Append this RFID dictionary into the list of RFID dictionaries
3. Iterate through the list to output the dictionaries in an organized manner. Use the string .format() method for this.

Objectives:

* Use dictionaries to group and organize data
* Define dictionaries and get accustomed to accessing their data

***4- Loops:***

Modify your "scan" code so that instead of writing the user inputs to three separate variables,

you instead input it into a dictionary. Create a list of RFID objects and then add your new RFID

to the list. Print the list to verify its contents

Instructions:

1. Create a list of RFID dictionaries. Each should have keys for the product name, location and manufacturer ID with corresponding values.
2. Reformat your code for "scanning" the RFID so that it assigns the input values to a new RFID dictionary. Append this RFID dictionary into the list of RFID dictionaries
3. Iterate through the list to output the dictionaries in an organized manner. Use the string .format() method for this.

Objectives:

* Use dictionaries to group and organize data
* Define dictionaries and get accustomed to accessing their data