Program Walkthrough

Before looking at the first line of code, it is helpful to describe the actual operation and flow of control of the application – basically its design. That design is as follows:

- Read a list of device IP addresses from a file and store it in a device list
- For every device in the device list:
  - Connect to the device using Telnet
  - Run the show version command and extract the software version string
  - Write the device IP address and its software version to a file

Read Device Info from File

The application will read a list of IP addresses from a file, and place them into a Python list. The first step is to create an empty list of devices as follows:

```
devices_list = []
```

Notice that empty list is created using empty square brackets ('[]'). Python is reasonably consistent in its use of brackets, and when used to show data, square brackets mean 'list'.

Next, the application opens up the file of device IP addresses.

```
file = open('devices', 'r')
```

The application is using the built-in function open in order to open the file named 'devices'. Be aware of a couple of things:

- In Python, variable names use underscores '_' by convention, which you should follow.
- However, file names sometimes have hyphens, for example 'device-list.txt'.

As you can see from the code, the file is being opened for reading ('r').

After the file is open, and the list is created, the application can set about looping through the reading of the file, line by line, taking each line (which should contain an IP address) and putting it into the list of devices *devices_list*. The use of *rstrip()* method on the *line* variable will strip any white space, line feeds or carriage returns from the line. The actual code looks as follows:

```
for line in file:
   devices_list.append( line.rstrip() )
```

Again there are a few things to notice here:

- A *for* loop is used to sequence through the file, stopping when all lines have been read.

- The end of the *for* statement and the beginning of the code inside the loop is identified by a colon (':') at the end of the for statement, and indentation of the code block that follows. By convention, Python code blocks are indented **four** spaces. **You must use spaces to indent—you cannot use the tab key.**

- For every iteration of the *for* loop, the variable *line* will be set to the value of the line in the device file.

- Elements are added to the end of the list using the *append* method.

You may be asking about error checking the file to make sure that it has IP addresses in each line, and about other potential issues. You are correct, in a completed production application you will be doing more error checking. But for this simple example, and for much of this course, the examples will be simplified in order to highlight the teaching points.

For Every Device

In this section of code the application once again uses the basic looping control structure in Python, the *for* loop. The example shows how easy it is to iterate through a list:

```
for ip_address in devices_list:
    ...
    # code to get version information
    ...
```

The *for* loop will start with the first device IP address in the list, and for every address, it will perform some operation (shown later). *for* loops make up a large part of much Python code, and this example for iterating a list, and the previous one for reading lines from a file are classic examples.

Things to note about for loops:

for is the simplest way to iterate through lists.

- The end of the *for* statement, and transition to the beginning of the code block, is identified by the colon ":" at the end of the *for* statement. There are no curly braces ('{}') or 'begin' and 'end' in Python like you may be familiar with from other programming languages.

- The example includes a comment, denoted by the '#'.

Connect Using Telnet

Some of the detailed code has been organized in function called *connect*. This function uses the external *pexpect* library to connect to the device. For now, focus on the actual mechanism used for calling those functions, and for receiving the return values.

session = connect(ip_address, username, password)

In this function call, the device IP address, username, and password parameters are passed to the *connect()* function. The function will receive those parameters and use them in connecting to the device using Telnet, which will return the *session*.

Note that the *session* will need to be closed after the version information has been retrieved.

Telnet Session

The actual code to make the connection via Telnet to the device is located in the *connect* function. Not all the code is shown here, to keep things simple. However, certain aspects are worth making note of, even in this introductory module.

import pexpect

This *import* statement is the reference to the external Python library that is used to connect to the device. The library, Pexpect, was created for the specific purpose of enabling communication with networking devices – an example of the fact that there is a lot of helpful code out there to use to accomplish common tasks. The name Pexpect is indicative of the functionality of the library—it helps the programmer to tell Python what output to expect from the device, and simplifies the code that needs to be developed to handle the different cases.

There are a couple things to note here:

- The function name *connect* should be unique within the module.
- The parameters *ip_address*, *username*, and *password* can in general be any names that you choose to use and are identified by their order.

Note
There are things such as named parameters in Python, but in this example the parameters are specified by their order.

- There is a colon (*:*) at the end of the function, signifying the beginning of a code block, as is done consistently in Python.

There is a fair amount of code that is involved in connecting the Telnet session, providing the *username* and *password*, and returning *session*, which is the reference to the *session*. This code can be examined in the complete program listing.

Get Version Information

Gathering version information is the simple task of connecting to the device, running the ***show version*** command, and extracting the appropriate version string.

The first step is to call the *get_version_info()* function, providing a reference to the session that was created previously:

device_version = get_version_info(session)

Calling this function will return the actual version string. The function needs to do the following:

- Send the show version command to the device.
- Take the output from the command, and search for the pattern that represents the version string.

The code to accomplish these tasks is shown below. First, the application sends the ***show version*** command to the device, via the Pexpect telnet session.

session.sendline('show version | include Version')

This command in Python uses the *session* reference, which was assigned when the telnet session was opened using Pexpect. The application is invoking the method *sendline*, which does what it appears – it sends a line of input to the device. In this case, the line of input being sent is the *show version | include Version* CLI command.

After sending the ***show version*** command, the application should be expecting some output back from the device. That output is received as a complete block of text representing all output. The application splits the output into lines and then uses the first line of the output splitting that on ',' to extract the version string from the output. Then the third part of the split version string is extracted and striped to remove any white space assigning that to the *version* variable.

```
# Extract the 'version' part of the output
version_output_lines = session.before.splitlines()
version_output_parts = version_output_lines[1].split(',')
version = version_output_parts[2].strip()
```

Once the version line has been found, and the *version* saved, the function can return the found *version* to the caller.

return version

At this point, the application has the version for this specific device. The next step is to print that version information to a file.

Write Version to File

Writing text files is relatively easy, as with reading them. The first step is to *open* the output text file for writing, getting a reference to the output 'version-info-out' file:

version_file_out = open('version-info-out', 'w')

The application is now able to use 'version-info-out' to write output lines.

version_file_out.write('IP: '+ip_address+' Version: '+device_version+'\n')

The *version_file_out* object invokes the *write* method, passing the information to be printed to that line of the file. The output will be the string *'IP: '* then the IP address of the device, followed by the string *' Version: '* and the version number that has been extracted from the ***show version*** command.

The last step is to behave well by cleaning up—which means to close the output file:

version_file_out.close()

Review

Now is a good time to review some of the fundamental components of Python that have been demonstrated in the example program and that you will be exploring in the course:

Data structures: Strings, Lists, and Tuples have been used to store and manipulate networking and network device information.

Input / output: The application read text files to retrieve network device information, and to write out information about each of those network devices.

Control structures: for loops were used to control the operation of the program, reading through a list of network device IP addresses, iterating through the list of devices for communicating with each, and iterating through lines of output to find the string that contained the version of code running on the network device.

Functions: Functions were used to consolidate and isolate certain tasks.

Code blocks: Code blocks were used and you learned something about Python's somewhat unique method of identifying them (':' and indentation).

Libraries: An external library (Pexpect) was used to help accomplish the task – something all Python developers tend to do, leveraging pre-existing code to simplify and expedite development of their own applications.

Regular expressions: A regular expression was used to match the string containing the software version of the device.