LAB – HELLO DEVICE

OBJECTIVE

In this lab you will use a simple Python network-based application, which uses the following Python runtime environment:

- Running Python from a Linux terminal window.
- Entering Python statements one-by-one at the Python prompt **>>>**

The application:

- Uses an external library **pexpect**.
- Launches a network-based function **ping** that is imported from the pexpect external module.
- Prints the results of the ping.

Then create a reusable application to save entering the commands every time.

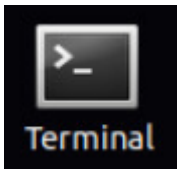
Optionally, create a GitHub repository to save your code files.

PART 1

Open a terminal and switch to the lab directory

STEP 1: OPEN A TERMINAL WINDOW

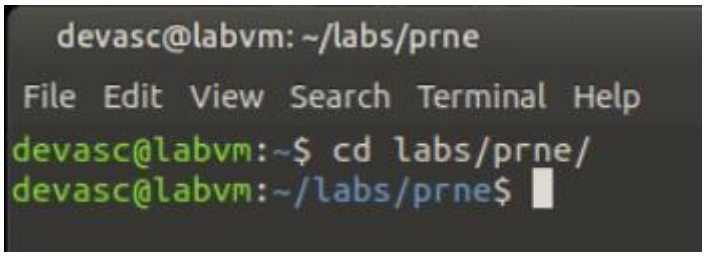
Double-click the Terminal icon on the desktop to open the terminal window for use in this lab.



STEP 2: CHANGE DIRECTORY

Change to the directory **labs/prne/** in the user home directory, which holds the files for the course labs.

```
~$ cd labs/prne/
```



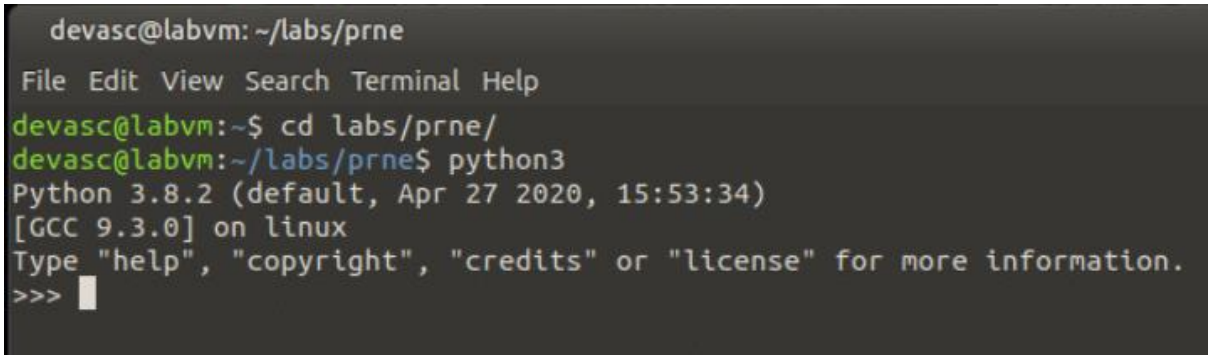
PART 2

Create and run the simple application

STEP 1: START THE PYTHON INTERPRETER

To start the Python Interpreter, within the terminal window type **python3**.
Once the prompt changes to three right-angle-brackets (**>>>**), you are now in the Python interactive interpreter and can start typing.
Whilst in the Python interactive interpreter, when you type in a Python statement and press return, the statement will be executed immediately.

```
~/labs/prne$ python3
>>>
```



STEP 2: IMPORT THE REQUIRED MODULES/LIBRARIES

The import statement tells Python to import all of the functions contained module **pexpect** from the Python library.

```
>>> import pexpect
```

STEP 3: SET THE PING OPTIONS

Using the pexpect module, create (spawn), a command process that sends a ping five times to the localhost.
The information inside the parenthesis is telling the spawned process what command to run and any options.

```
>>> ping = pexpect.spawn('ping -c 5 localhost')
```

STEP 4: EXECUTE THE PING

Tell pexpect what value to expect in response to the ping command, which is EOF or TIMEOUT and set the **result** value based on the results of the ping.

```
>>> result = ping.expect([pexpect.EOF, pexpect.TIMEOUT])
```

STEP 5: PRINT THE PING RESULTS

Tell Python to print the value stored in the **ping** object.

```
>>> print(ping.before)
```

The parameter **before** is used to store what came **before** the EOF or TIMEOUT. In other words, the actual output of the ping command. In other words, the actual output of the ping command, like you would see if you ran the ping at terminal yourself.
However, that may not always be the case, as can be seen in the following output from the print statement. This has happened due to an encoding error, where Python doesn't know definitively which encoding standard to use whilst attempting to interpret the data in the buffer.

```
>>> print(ping.before)
b'PING localhost(localhost (::1)) 56 data bytes\r\n64 bytes from localhost (::1)
: icmp_seq=1 ttl=64 time=0.077 ms\r\n64 bytes from localhost (::1): icmp_seq=2 t
tl=64 time=0.038 ms\r\n64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.0
46 ms\r\n64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.046 ms\r\n64 by
tes from localhost (::1): icmp_seq=5 ttl=64 time=0.046 ms\r\n\r\n--- localhost p
ing statistics ---\r\n5 packets transmitted, 5 received, 0% packet loss, time 40
93ms\r\nrtt min/avg/max/mdev = 0.038/0.050/0.077/0.013 ms\r\n'
```

To resolve this, the current print line needs to be replaced to convert the data from **bytes** to **str**. In this case, the command **decode(…)** needs to be added to the print line. In this instance, setting the data to be decoded as **UTF-8** and to **ignore** any values that it cannot decode.

Replace the previous ping statement with the following, to get the expected output.

```
>>> print(ping.before.decode('utf-8', 'ignore'))
```

The correctly formatted output is shown below.

```
>>> print(ping.before.decode('utf-8', 'ignore'))
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.077 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.046 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.046 ms

--- localhost ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4093ms
rtt min/avg/max/mdev = 0.038/0.050/0.077/0.013 ms
```

STEP 6: OTHER ENCODING OPTIONS (OPTIONAL)

Although in the above step the encoding was provided through the **print** command, another option exists in that the encoding could have been set within the spawn class of **pexpect**.

To see the difference in this option, you need to modify the ping child process to add the **UTF-8** encoding:

```
>>> ping = pexpect.spawn('ping -c 5 localhost', encoding='utf-8')
```

Then you need to re-execute the ping again:

```
>>> result = ping.expect([pexpect.EOF, pexpect.TIMEOUT])
```

Finally, print the new value stored in the **ping** object:

```
>>> print(ping.before)
```

STEP 7: EXIT PYTHON CONTEXT

In order to exit the Python context, you can press Control-D or type in **exit()**. The latter option actually runs a Python statement, calling the standard **exit()** function in Python.

SUMMARY

To execute a Python application from the command line:

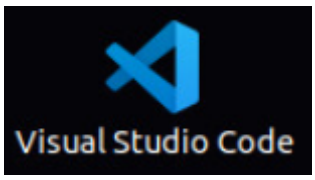
- Type in **python3** to enter the Python context, which is indicated by the **>>>** prompt.
- Type in your Python code, one line at a time. Python will execute each Python statement as you type it in.
  Note that:
  - For statements that span lines, such as a function call with many parameters, Python will execute the statement when it is complete.
  - If a statement spans more than one line, the Python prompt changes to **...**
  - You can use the up or down arrow keys to scroll through previous commands. You can edit previous commands and press enter again to execute the edited command. This feature is extremely useful, especially if you mistype part of a command and want to correct it.

PART 3

Entering the above commands every time that you wish to run a ping is not very productive, so let's create a reusable application that can be ran when required.

STEP 1: CREATE REUSABLE APPLICATION

Open **Visual Studio Code**, create a new file and save it with a filename of **hello-device.py**. Ensuring to save the file in the **~/labs/prne/** directory.



STEP 2: COPY CODE FROM PREVIOUS PART

Copy the same statements used in the previous part, to create the application that can be reused.

```
import pexpect
ping = pexpect.spawn('ping -c 5 localhost')
result = ping.expect([pexpect.EOF, pexpect.TIMEOUT])
print(ping.before.decode('utf-8', 'ignore'))
```

STEP 3: SAVE, RUN AND VERIFY APPLICATION

Save you application and then run it from the terminal rather than from within visual studio code.

```
~/labs/prne$ python3 hello-device.py
```

After a few moments, the output from your application will be displayed in your terminal window, verify that it is comparable to below.

```
devasc@labvm:~/labs/prne$ python3 hello-device.py
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.043 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.045 ms

--- localhost ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4100ms
rtt min/avg/max/mdev = 0.043/0.048/0.060/0.006 ms
```

PART 4 (OPTIONAL BUT HIGHLY RECOMMENDED)

As this lab is completed in NETLAB+ and your code files will be erased when the reservation ends, it is advisable to save your files in GitHub under a new private repository for this course.

This will allow you to download the code files again in NETLAB to complete the lab, or re-use it elsewhere.

Recommended repository settings:

Repository name: **prne-labs**
Description: **Code files for the Programming for Network Engineers module**
Public/Private: **Private**