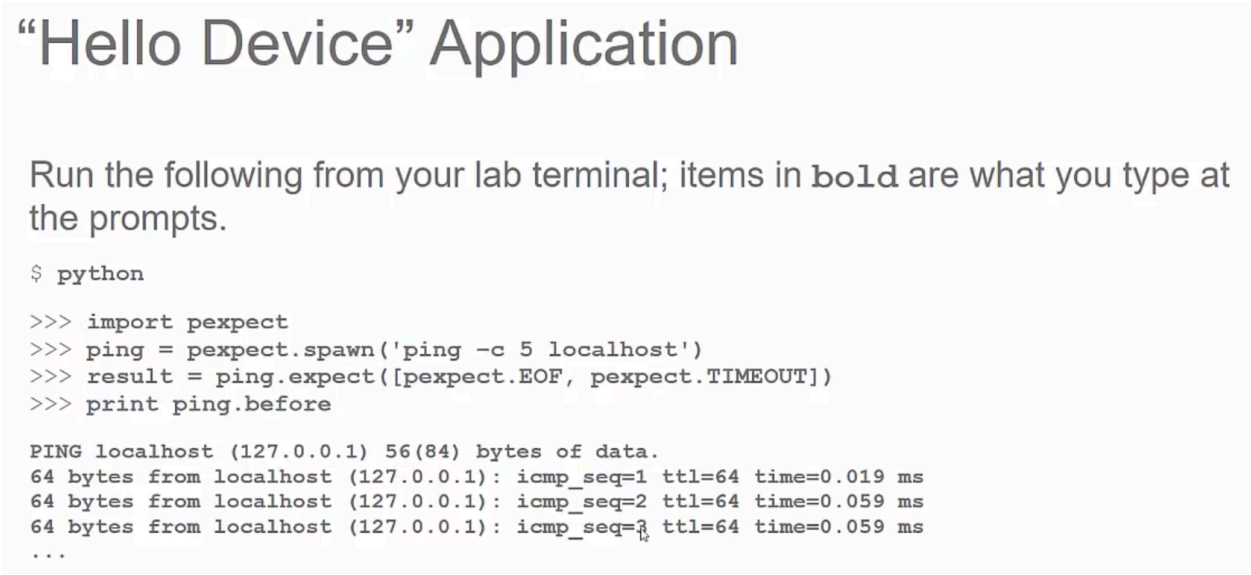
"Hello Device" Application

Here is your first Python network-based application. This small sample application uses the following Python runtime environment:

- Running Python from a 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 library.
- Prints the results of the ping.



Running Python: Command Line

As we have just done:

- Type '**`python`**'
- Get python prompt: '**`>>>`** '
- Type in Python code one line at a time
- Python executes each line (each Python 'statement') after you hit carriage return
- **`Ctrl-D`** or **`exit()`** to exit

Running Python: From a File

- Use an editor to enter the code for our 'hello device' Python program in a '.py' file, for example 'hello-device.py'.
- Invoke python with name of the file as parameter:

```
$ python hello-device.py
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.019 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.059 ms
```

Step 1

Start the Python Interpreter

```
$ python
>>>
```

You know that you are typing into Python if the prompt is those three right-angle-brackets. When you type in a Python statement and press return, the statement will be executed immediately.

Step 2

Tell Python to use the 'pexpect' library:

```
>>> import pexpect
```

The import statement tells Python to import all of the functions contained in the external library named pexpect.

Step 3

Set the ping options:

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

'pexpect.spawn is using the pexpect library to create, or spawn, a command process. The information inside the parenthesis is telling the spawned process to ping the localhost five times.

Step 4

Execute the ping:

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

This line is doing two things: telling pexpect what value to expect in response to the ping command, which is EOF or TIMEOUT; and setting the 'result' value based on the results of the ping.

Step 5

Print the ping results:

```
>>> print(ping.before)
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.054 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.054 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.087 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.114 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.122 ms
...
```

This line tells Python to print the value stored in the 'ping' object. The parameter 'before' is used to store what came 'before' the EOF or TIMEOUT. In other words, the actual output of the ping command, like you would see if you ran the ping at terminal yourself. That output is what is shown below the print statement.

In summary, to execute a Python application from the command line:

- Type in 'python' 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 mis-type part of a command and want to correct it.
- 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.