

---

**From:** Kirk Byers

**Sent:** Tuesday, January 10, 2023 9:00 PM

**To:** sydasif78@hotmail.com

**Subject:** [PyNet Learning Python] - Lesson5 / Functions and the Python Debugger

*Note: There is a table of contents for each video at the bottom of this email including timestamps to where various content is located. This should be helpful in navigating the videos.*

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

1. **Functions (Part1)**  
Video link <https://vimeo.com/247570174>  
Length is 8 minute
2. **Functions (Part2)**  
Video link <https://vimeo.com/247581011>  
Length is 11 minutes
3. **Misc Topics (Part1)**  
Video link <https://vimeo.com/247582360>  
Length is 10 minutes
4. **Misc Topics (Part2)**  
Video link <https://vimeo.com/247655574>  
Length is 8 minutes
5. **Python Debugger (pdb)**  
Video link <https://vimeo.com/247724017>  
Length is 10 minutes

**Additional Content:**

[A Byte of Python, Functions](#)

[How to use the Python Debugger](#)

**Exercises**

Reference code for these exercises is posted on GitHub at:

[https://github.com/kbyers/pynet/tree/master/learning\\_python/lesson5](https://github.com/kbyers/pynet/tree/master/learning_python/lesson5)

1a. 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.

1b. 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.

2. 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.

3. Similar to lesson3, exercise4 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.

4. Copy your solution from exercise3 to exercise4. 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 = [])

## **CLASS OUTLINE**

### **1. Functions (Part1) (VIDEO1)**

- A. How to define [0:04]
- B. How to call [0:35]
- C. Return values [1:41]
- D. Adding positional arguments [3:21]
- E. Adding named arguments [4:54]
- F. Default values [7:13]

### **2. Functions (Part2) (VIDEO2)**

- A. Calling with \*args [1:57]
- B. Calling with \*\*kwargs [3:45]
- C. Passing mutables as function arguments [4:49]
- D. Variables are local to the function [7:59]

### **3. Misc Topics (Part1) (VIDEO3)**

- A. Classes [0:23]
  - 1. Why you might want to use classes [0:43]
  - 2. Syntax for creating a class [2:16]
- B. Modules [5:15]
  - 1. What is a module [5:38]
  - 2. Importing a module you have created [5:54]
  - 3. Calling your imported function [6:05]

- 4. How Python finds something using `sys.path` [6:40]
- 5. Two ways to perform imports [7:10]

#### 4. Misc Topics (Part2) (VIDEO4)

- A. Packages [0:41]
- B. List comprehensions [1:55]
- C. Lambda functions [4:47]

#### 5. Python's Debugger (PDB) (VIDEO5)

- A. The (Pdb) shell [0:08]
- B. Listing lines using 'list' [0:52]
- C. Using 'step (s)' to step through a program [1:27]
- D. 'args' to look at arguments of a function [1:56]
- E. 'p' to print variables [2:01]
- F. 'pp' to pretty print [2:10]
- G. 'up' to move up the stack [2:27]
- H. 'down' to move down the stack [2:56]
- I. execute until the return statement [3:15]
- J. Executing using 'python -m pdb' [4:10]
- K. Using 'next (n)' [5:11]
- L. Difference between step and next [5:56]
- N. Setting a breakpoint [6:13]
- O. 'continue' to execute until the breakpoint [6:24]
- P. Use '!<command>' to execute python code [6:48]
- Q. Debugging principles [7:54]
  - 1. Get information out of your program [8:08]
  - 2. Quick feedback loop [8:15]

*Kirk Byers*

<https://pynet.twb-tech.com>

---

To make sure you keep getting these emails, please add [support@twb-tech.com](mailto:support@twb-tech.com) to your address book or whitelist us. Want out of the loop? [Unsubscribe](#).

Our postal address: Twin Bridges Technology, 88 King Street #1217, San Francisco, CA 94107.