

## CS 1110, LAB 02: FUNCTIONS AND (IN) MODULES — SOME “HI”-LIGHTS

<http://www.cs.cornell.edu/courses/cs1110/2017sp/labs/lab02.pdf>

First Name: \_\_\_\_\_ Last Name: \_\_\_\_\_ NetID: \_\_\_\_\_

Lab updated from the printed version on Feb 6 4:38pm: on pg 7, see change in orange.

Learning goals: Practice with: (1) the Command Shell; (2) using and editing functions defined in an external file; (3) Komodo Edit; (4) running Python scripts on the command line; (5) using one function as a helper for another.

Getting Credit: Deadline: **in the first 10 minutes of (your) lab next week (Tu Feb 14 or Wed Feb 15)**. The checking-off procedure is the same as before.<sup>1</sup>

### 1. BUILT-INS WE’LL USE IN THIS LAB

**1.1. Repeated string concatenation.** You know that `*` is the multiplication operator for ints and floats. It turns out that `*` can also be used to “repeat” a string: If `s` is a string and `n` is a positive int, then the expression `s*n` yields a string that is `n` copies of `s` concatenated together.

Some examples should make this clear. Open a command shell and start up Python interactive mode.<sup>2</sup> Type in the following expressions in order and write down the evaluation results.

Expression	Evaluation Result
<code>'abc'*2</code>	
<code>'abc.'*4</code>	
<code>s = 'a b c. '</code>	
<code>s*2</code>	
<code>n = 3</code>	
<code>s*n</code>	

---

Course authors: E. Andersen, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White

<sup>1</sup> In case you’ve forgotten, here’s a reminder: Show this handout and/or your code to a staff member either (a) during your lab 02 session, (b) in consulting hours listed at <http://www.cs.cornell.edu/courses/cs1110/2017sp/about/staff.php> up to the day **before** your next scheduled lab section, or (c) in the first 10 minutes of (your) next scheduled lab (Tu Feb 14 or Wed Feb 15). Beyond that time, the staff have been instructed not to give you credit. Labs are graded on effort, not correctness. We just want to see that you tried all the exercises, and to clarify any misunderstandings or questions you have.

<sup>2</sup>Remember, you do this by typing `python` at the command shell prompt. You’ll then see the Python interactive prompt, `>>>`.

**1.2. Random number generation.** As mentioned in lecture, the Python built-in module `random` contains functions for generating random numbers. One such function is `randint(a,b)`, which, if `a` and `b` are ints with `b ≥ a`, returns a random integer drawn from between `a` and `b` inclusive.

Let's try it out. First, in Python, make the name of the `random` module accessible to Python by entering `import random` at the interactive prompt.

Next, here are two ways you might now try to generate a random int between 1 and 3 inclusive. Circle the one that is correct.

`random.randint(1,3)`

`randint(1,3)`

Why would the other option would give you an error?






Check your answer by entering it four separate times in Python. (Hint: save time and typing by using the up-arrow key to bring up the previous line you entered.) Write down the four numbers you get here:

## 2. NAVIGATING FOLDER (DIRECTORY) STRUCTURE IN A COMMAND SHELL

**2.1. Setting up.** Create a new directory called `lab02` on your Desktop.<sup>3</sup>

Download the files you need for lab 02 from the Labs section of the course web page, <http://www.cs.cornell.edu/courses/cs1110/2017sp/labs> and put them all in the new `lab02` folder you just created.

**2.2. Getting to the right directory (folder) in the command shell.** Start the command shell for your computer:

- On Windows, this is called the *Command Prompt*. It is found in  **Accessories** in the Start Menu on Windows 7, or  **Apps** ▶ **Windows System** on Windows 10.
- On Mac OS X, this is called the *Terminal*. It is found in  **Applications** ▶ **Utilities**, or type  +  to call up Spotlight Search, and then type **Terminal** in the Spotlight Search window to find it.

You will type commands into this command shell, *not* in the Python interactive shell! Make sure you do **not** see the “>>>” prompt!<sup>4</sup>

<sup>3</sup>In future labs, you can put directories wherever you want, but this lab assumes you put `lab02` in your Desktop folder.

<sup>4</sup>If you do, enter `exit()` or hit  +  to exit Python interactive mode.

The command shell “looks into” a specific folder. We call this folder the *active* or *working directory*. When the command shell starts, the active directory is your *home directory*. If you are using your own computer, in both Windows and OS X, this is the folder with your name on it.

Next, open up a window for your home directory, and put it next to the command shell. Go back to the command shell. If you are using Windows, type

```
dir
```

and hit `[Return]`; if you are using OS X, type

```
ls -l
```

(that is a “ell ess space hyphen ell”, not a “one ess space hyphen one” — the name comes from the word list, and the `-l` means use the option that generates long output format) and hit `[Return]`. In a few words, describe what you see and how this correlates to what you see in the window next to the command shell.

Now let us move the view of the command shell to the Desktop. This is actually a folder stored inside your home directory. To get there from the home directory, enter (on either OS)

```
cd Desktop
```

into the command shell. (“`cd`” stands for change directory.)

Once again enter either `dir` or `ls -l` (depending on your operating system) into the command shell. Is the command shell still listing the files that are in your home directory, or is it now displaying the files that are on your Desktop, or something else?

We want you to change the active directory to the folder `lab02`. If you followed the instructions above, this folder should be in your Desktop folder. So, enter

```
cd lab02
```

If all the above instructions have been executed correctly, then you should now be able to run a verification python script we've written for you; so, enter

```
python verify_shell_location.py
```

(To get the two *underscore* characters “\_” in the name `verify_shell_location.py`, on most keyboards, do `shift`+`-`, where the latter is the key you hit to get a minus-sign.)

What output do you get? **If you get anything that doesn't contain “Hurrah!”, ask for help immediately.**

**IF YOU GOT ANY ERROR MESSAGES OR DIDN'T UNDERSTAND SOMETHING ON PAGE 3 OR THIS PART OF PAGE 4, ASK FOR HELP BEFORE PROCEEDING!**

Later, you can also refer to our webpage for more on working with the command shell here: <http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/command.php>

### 3. USING FUNCTIONS FROM A MODULE IN INTERACTIVE MODE



The previous section set up the command shell so that its working directory contains the lab files. Let's now experiment with using the functions stored in the file `greetings.py`, at first in Python interactive mode.

Start up Python in the command shell.<sup>5</sup>

Make the name of the module `greetings` accessible to Python by entering `import greetings`.

See human-readable documentation of what is in `greetings` by entering `help(greetings)`, and pay special attention to the description for `multi_hi`. When you're done, hit `Q` to quit seeing the documentation.

Do you still remember the `multi_hi` description? Probably not. So instead, since you're starting to become a programmer:

- (1) Start up Komodo Edit.
- (2) In Komodo Edit, open<sup>6</sup> file `file`  `Desktop` `lab02` `greetings.py`, where the  symbol stands for your home directory.
- (3) In `greetings.py`, look for the line `def multi_hi(name, num):` (line 9). Underneath it is the *docstring*, enclosed in triple double-quotes, that explains what the function `multi_hi` does; when you were in Python interactive mode, when you typed `help(greetings)`, Python extracted that docstring and displayed it to you.<sup>7</sup>

<sup>5</sup>Reminder: enter `python` at the command shell prompt; you should then get the `>>>` Python prompt.

<sup>6</sup>See the course webpage labeled “Python/Komodo” for instructions on how to open files: <http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/python.php>.

<sup>7</sup>Which is kind of cool, but you might not appreciate that until later. It's OK if you don't find that particularly interesting now.

- (4) Also, below the docstring for `multi_hi` is the actual code body for the function. Notice that it uses the string repetition `*` we learned about in Section 1.1.

The docstring and code itself should give you enough information to guess what each expression below evaluates to. So, fill in your guesses below.<sup>8</sup> Then verify if Python agrees with you!

Expression	Expected Value	If Python disagrees, why?
<code>greetings.multi_hi("Wei", 2)</code>		
<code>greetings.multi_hi("Jinx", 3)</code>		
<code>greetings.multi_hi("Potenuse", 1)</code>		
<code>greetings.multi_hi(Wei, 4)</code>		

#### 4. USING FUNCTIONS FROM A MODULE IN A SCRIPT

Now, let's see how function `multi_hi` can be used in a Python script that is run by the command shell.

First, exit Python<sup>9</sup>.

You'll be running the Python script `run_multi_hi.py`, so open it up in Komodo Edit to see what it's supposed to do.

Hey, look, except for using `print` to display the result of evaluating expressions, this script does just what you just did in Section 3. Prove it to yourself as follows: at your command-shell prompt<sup>10</sup>, enter `python run_multi_hi.py` in order to have Python run the script.

Notice that you get an error message stating that there's a problem with a particular line. Read the error message to answer the following questions: Which file has the problem? Which line number has the problem? And what does that line say? Is it similar to what you saw in Python interactive mode?

#### 5. WRITING YOUR OWN FUNCTIONS

5.1. **Complete function `rand_hi()`.** Let's now make `rand_hi()` in `greetings.py` do something interesting: generate a random number of "hi"s, using the function `randint` from Section 1.2.

<sup>8</sup>You'll probably find using the up-arrow key useful here, again.

<sup>9</sup>Enter `exit()` or hit `ctrl+D`.

<sup>10</sup>Which should *not* be `>>>`.

A skeleton of the new function `rand_hi` is in `greetings.py`. Additionally, we've written a test script named `test_rand_hi.py` that will help you check your function as you go. It's useful to be able to see both files in Komodo Edit at the same time, so here's how you can do it. Go to (or re-open) `greetings.py` in Komodo Edit; then activate “*Split View*”: in the Komodo Edit menu, `View` `Split View`. You'll get two file-viewing panes. Click on the pane you want to view `test_rand_hi.py` in, and then open `test_rand_hi.py`.<sup>11</sup>

You can see that `test_rand_hi.py` just prints the result of evaluating the function `rand_hi` several times. Now, scroll down in `greetings.py` so you can see the current definition of `rand_hi`, which starts at line 19. Right now, the function just sets variable `reps` to 0, and doesn't have a return statement. This means that it implicitly returns the special value `None`. What do you therefore think will happen if you run Python on `test_rand_hi.py`?



Test it out (enter `python test_rand_hi.py` in the command shell). What do you get, and why?



Let's get to work on `rand_hi`!

The first step is to take responsibility for the changes you'll be making<sup>12</sup>, as follows: go back to `greetings.py`, and in line 2, replace the text “PUT YOUR NAME AND NETID HERE” with, well, your name and net ID. Make sure you leave the comment character `#` at the beginning of the line.

Now, *save and test and ask*: save<sup>13</sup> the file (*This is important! Forgetting to save leads to confusion!*) and check that you didn't insert any syntax errors by running Python on `test_rand_hi.py` again; you should get the same boring result you got before, but no errors. *If you do get an error or get different output than before, ask for help.*

Since `randint()` is in module `random`, make that module's name available in `greetings.py` by inserting the line

```
import random
```

underneath the module docstring, i.e., around line 7 or 8. Make sure that you did *not* indent your new line. Insert blank lines so that the import statement is visually separated from both the module docstring above it and the beginning of the function definitions below it.<sup>14</sup>

Save-and-test-and-ask again; you should get the same output as before.

---

<sup>11</sup>We think you'll probably agree that *Split View* is cool. If you'd like the two files to be side-by-side instead of one-above-the-other, use the Komodo Edit menu option `View` `Rotate Split View`. You can toggle back and forth between horizontal and vertical layout by repeated application of `View` `Rotate Split View`.

<sup>12</sup>You will need to do this for every assignment, so you might as well get into the habit now.

<sup>13</sup>See the course webpage labeled “Python/Komodo” for instructions on how to save files and *check whether files have been saved* in Komodo Edit: <http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/python.php>.

<sup>14</sup>You can look at line 7 of file `run_multi_hi.py` to see an example of an import statement in a Python file.

Now, we want `reps` to store a random number, not just be 0 all the time. So, replace the placeholder line that starts with `reps = 0` with the following: (next line updated, Feb 6 4:38pm)  
`reps = random.randint(1,6)`

Make sure that your new line is still indented the way the previous version was.

Save-and-test-and-ask again; you should still get the same output as before, but we need to make sure there aren't any syntax errors introduced.

Finally, add the line `return multi_hi(name,reps)`, indented the same amount as your `reps = ...` line.

Save-and-test-and-ask: now what output do you get?

Why do you have to say `return multi_hi(name,reps)`, and not `return greetings.multi_hi(name,reps)`?

## 5.2. Use `rand_hi()` as a helper for `natural_hi()`.

The greetings produced by `test_rand_hi.py` are informal, but rather impersonal. Function `natural_hi`, later in file `greetings.py` is meant to be an improvement.

Scroll down in `greetings.py` so you can see the code for `natural_hi`. Right now, it just uses a Python built-in called `raw_input` to ask the user a question, and then stores the user's answer in variable `input_name`.

Let's have `natural_hi()` feed the value in `input_name` as an input into `rand_hi()`, and use the value that the latter returns as a result as its own output. To do this, add the following line, indented exactly as the line `input_name = raw_input ...` is:

```
return rand_hi(input_name)
```

To see whether you have `natural_hi()` implemented correctly, save the file first! and then in the command shell, run Python on `finally.py`. If you get some personalized output plus an encouraging statement, write them here; otherwise, ask for help.

**Indicate all the changes you made to `greetings.py` on the copy of the code given on the last page of this handout** so that you can show the lab staff what you did and they can check off your work.

*We hope you got a "natural high" from completing this lab!!*

6. INDICATE CHANGES YOU MADE TO GREETINGS.PY HERE

```
1 # greetings.py
2 # PUT YOUR NAME AND NETID HERE
3 # Feb, 2017
4 # Skeleton by Profs Erik Andersen and Lillian Lee
5
6 """Library of functions producing greetings"""
7
8
9 def multi_hi(name,num):
10     """Returns a string of the form: "hi " repeated <num> times, followed by
11     <name>, followed by "!".
12     Preconditions (i.e., assumptions this function makes about its input):
13         name is a string
14         num is a positive int
15     """
16     return "hi "*num + name + "!"
17
18
19 def rand_hi(name):
20     """SEE LAB HANDOUT.
21     Precondition: name is a string"""
22     reps = 0 # ***placeholder: replace as instructed***
23
24
25
26 def natural_hi():
27     """SEE LAB HANDOUT"""
28     input_name = raw_input('Please enter your name: ')
29
```