# CS 1133, LAB 2: MODULES AND FUNCTIONS

http://www.cs.cornell.edu/courses/cs1133/2017fa/labs/lab2/

**First Name**: _____ **Last Name**: _____ **NetID**: _____

The purpose of this lab is to get you comfortable with making your own modules and scripts. As part of this, you will write your first function. Unlike the last lab, there are no files to download this time.

**Getting Credit for the Lab.** This lab is unlike the previous one in that it will involve a combination of both code and answering questions on this paper. In particular, you are expected to complete both the module `area.py` and the script `dice.py`.

When you are done, show both the handout and these two files to your instructor, who will rercord your success. You do not need to submit the paper with your answers, and you do not need to submit the computer files anywhere.

As with the previous lab, if you do not finish during the section, you have **until Tuesday of next week to finish it**. Simply show your lab to a consultant during consulting hours. You should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

## 1. Die Roller Script

Before we have define your first function, we want you to be comfortable with modules and scripts. We are going to have you make a single fint that will work as either, depending upon how you use it.

A lot of board games require you to roll two (six-sided) dice and add the results together. In this lab you will create a script that essentially does this. Of course, Python does not have any physical dice to roll. But it can generate random numbers, which is the same thing.

More specifically, you should create a file called `dice.py`. In that file, you should do three things:

(1) Generate two random numbers in the range 1 to 6 (inclusive)

(2) Add the numbers together and assign them to the variable `roll`

(3) Print the variable `roll` at the end.

The first step is the only one that we did not show you in class. Fortunately, there is a module that solves this problem for you. Look at the specification for the function `randint(a,b)` in the module `random`.

http://docs.python.org/3/library/random.html#random.randint

You should use this function to generate your two random numbers.

Course authors: D. Gries, L. Lee, S. Marschner, W. White

The first thing we want you to do is to put a document comment at the start of the file. A document comment starts and ends with three quotes (`"""`). It also contains the following four lines:

(1) The first line is a one-line description of the module.
(2) The second line is blank.
(3) The third line your name and net-id
(4) The last line is today's date.

As we shall see in class, sometimes there are more lines between line two and three. But four is enough for now, and is typically the format we will use for this class. For the file `dice.py` we want the first line to be

```
A simple die roller
```

Complete the file `dice.py` before going on to the next questions.

## 2. Questions

While we can test your script, we want you to test it on your own. Create a folder called `lab02` and store `dice.py` in this folder. Like you did last week, open up a command shell and navigate to this folder. Start Python in interactive mode and type

```
>>> import dice
```

What do you see?

Now type the command

```
>>> dice.roll
```

Again, what do you see? How does it compare to what you saw before?

The document comment that you wrote at the top is important, as it provides the user with information on how to use the script. Type the command.

```
>>> help(dice)
```

What do you see just below the word Name (which is in bold)?

Finally, quit Python so that you are back in the general command shell from last week's lab. Now we want you to run `dice.py` as a script. In your command shell, type

```
python dice.py
```

One last time, what do you see?

## 3. Writing a Function

Now you are ready to define your first function. Create a new module, this time called `area.py`. You should put it in the same folder as `dice.py`. Once again, you should put the following comments at the top of the file:

(1) The name of the file (`area.py`)
(2) Your name **and net-id**
(3) Today's date.

You are going to define a function that that computes an area of a rectangle. The catch is that we will not have the length and width of the rectangle. Instead, the function parameters will be the width and *perimeter*. To get you started, copy the following into your module.

```
def area(perim,width):
    """
    Returns: The area of the rectangle defined by the given paramaters.

    Example: If perim is 10.0 and width is 2.0, this function returns 6.0

    Parameter perim: The rectangle perimeter (as a float > 0)
    Parameter width: The rectangle width (as a float > 0)
    pass
```

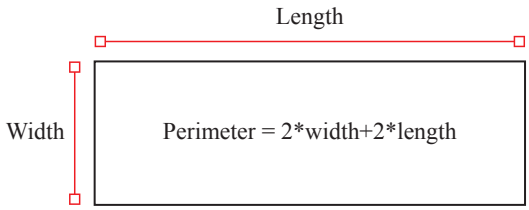
This is called a *function stub*. A stub is a function definition that has the header but no body. This allows you to call the function for testing, even when you have no finished it.

Before you do anything else, let us test that it is working. Navigate to that folder containing `area.py` and start the Python interactive shell. Type the following:

```
>>> import area
>>> area.area(10.0,2.0)
```

What happens?

Now implement the function. If you need a clue on how to define the function, remember that you can recover the rectangle length from the perimeter as shown below.




## 4. Testing Your Function

This last part of the lab is a head start preparing you for a future lecture. To make sure that the function is working properly, you need to test it. The way you test a function is as follows:

(1) You choose a sample input for the function.
(2) You read the specification (the part in """) to determine what the output should be.
(3) You call the function with your input as argument and look at the output.

If the output you get in (3) is the same as what you guessed in (2) then the test worked correctly. Otherwise, there is something wrong with your function definition and you need to fix it.

Let us start with the example test that we tried before you wrote function body. The input was perimeter 10.0, width 2.0. Reading the specification, we see that the function should return 6.0. To try out this test, make sure that you are in the same folder as `area.py` and start the Python interactive shell. Type

```
>>> import area
>>> area.area(10.0,2.0)
6.0
```

If you see the `6.0` after the function call, then your function is working correctly (so far). If not, call over a staff member to help you fix your function.

One test is never enough. Try to come up with three different tests for this function. All you need for a test is an input (for both parameters) and an expected output. We will not grade you on how good your tests are, as that is the subject of the first assignment. Right now, we just want you to make some tests. List them below.

| **Input** | **Expected Output** |
|---|---|
| | |
| | |
| | |

Now test your function using each of these three tests. Did you get the expected output each time? If so, you are done. Show this handout and the files `dice.py`, `area.py` to recieve credit.