Computer Science 1 — CSci 1100
Lab 2 — Functions
Fall Semester 2014

Lab Overview

The main goal of this lab is to learn to define and use functions in our programs and to learn good program structure.

A good program is easy to read and debug. One of the first things we want is for function definitions to come before any other code. We will work on this throughout this lab.

Second, you will see that code we have given you contains simple comments to explain what the code does. These comments are preceded with a pound sign. We want you to get in the habit of commenting your code. This is also a good exercise to explain to yourself what you are doing.

Checkpoint 1: Using existing functions

First, we will experiment with using functions. Create a new directory in your dropbox for Lab 2, and copy from Piazza the program called `lab2_check1.py`. Run this code and see that it produces no output.

Let's look closely what the code does: it creates a function that takes four values: `x1,y1,x2,y2`, and returns a floating point value that represents the Euclidean distance between two points with Cartesian coordinates `(x1,y1)` and `(x2,y2)`.

Then, the code executes this function, but does not do anything with the output of the function. We must take the value returned by the function, assign it to a variable and then print it. Once you print the output, you should see that function returns a value of 5.

Now, let us use this function to do something more interesting. Suppose, a car started at location `(100,100)`, then moved from `(100,100)` to `(100, 160)`, then moved from `(100, 160)` to `(80, 160)`, and finally moved from `(80,160)` to `(90,120)`.

Write code to call the function `line_length` repeatedly to find the total distance the car travelled in this trip. Print out the final result as an integer.

To complete Checkpoint 1: Show the TA your program when complete. Your program must follow the required structure: function definition first, followed by function calls.

Checkpoint 2: Restructuring code

In this section, we will restructure code with the help of a function. First start by downloading the program called `lab2_check2.py` and save it in your lab2 folder. Create a copy of it `lab2_check2v2.py` by creating a new program with the same

content and saving it as a second time. You will be modifying this copy and comparing its output to the original.

Run the original program and make sure you understand what it does. We can see that this program contains repetitive code. If you found a bug in one computation, you have to fix it in 4 places. Instead, you can put all this computation in a function.

Write a function that takes as input all the necessary data for the repeated code, does the computation and prints the results. Your function definition must come before any other code in the program. What should be the arguments for this function?

Now, rewrite the rest of the code by simply calling this function four times. Check and make sure it provides the same output as the original program.

To complete Checkpoint 2: Show the TA your program and make sure that your program follows the required structure: function first, followed by function calls.

Checkpoint 3: Prey and Predator Population

Now that we have experimented with functions, you will write a full program that defines functions and then uses them.

Suppose you are trying to understand how the population of two types of animals will evolve over time. You have bunnies and foxes. Unfortunately, foxes like to eat bunnies. So, even though the bunny population grows every year when baby bunnies are born, it also goes down because some of them are eaten. Would the fox population keep increasing in size until there are no bunnies left? Not so. The fox population is directly linked to the bunny population, their main food source. What will happen over time if you start with a given population of bunnies and foxes? Who will win out? Will they balance each other? This is what you will investigate in this part.

Suppose `bpop, fpop` are the population of bunnies and foxes currently. Then, next year's population of bunnies (`bpop_next`) and foxes (`fpop_next` are given by:

```
bpop_next = (10*bpop)/(1+0.1*bpop) - 0.05*bpop*fpop

fpop_next = 0.4 * fpop + 0.02 * fpop * bpop
```

What will be the population of bunnies and foxes after 6 years? Your program should print the population of both every year including the last year using the `print` statements.

Let's think about how to solve this problem. Start by creating a new file in the Wing IDE to work with. You will need variables to store the current population of bunnies and foxes. You need a function for computing the population of bunnies for the next year. Similarly, another function for computing the population of foxes for the next year. Now, call these functions six times to find the population after 6 years. You will need only need four variables: `bpop, fpop, bpop_next, fpop_next`. Try your program with the following initial populations:

Bunnies	Foxes
100	5
100	10
100	12
100	18

Remember two crucial things:

- Population is an integer. You never see half bunnies running around. You can convert a float to an integer using the built-in function `int()`.
- The second one is a bit tricky. The population of animals cannot be negative (something worse than extinct?). Don't listen to people around you who are telling you to use an if statement. You are forbidden to use an if statement. Think about how to use the `max()` function to make sure that your function does not return a negative number.

In the next few weeks, we will see that we can use loops to make this calculation much simpler. Loops will allow you avoid repeatedly typing (or copying-and-pasting) the same code and make your programs shorter, clearer, more general, and less likely to contain errors.

To complete Checkpoint 3: show a TA or a mentor the completed solution and illustrate its use with the first and last of the values given above.

Make sure your program follows the structure we outlined earlier in this lab: function definitions first, then your variables `bpop, fpop` are assigned their initial value, and finally the code to compute the seven year population.