Computer Science 1 — CSCI 1100
Lab 4 — Functions
Fall Semester 2019

Lab Overview

The main goal of this lab is to learn to define and use functions in our programs, learn good program structure and also build some skills in reading error messages.

A good program is easy to read and debug. Your program should be well-structured. Similarly, your variables and functions should have meaningful names. This will help you and others debug code.

First, in structuring your code, follow the guidelines we presented during lecture:

1. A general comment describing the program.
2. All import statements.
3. All function definitions.
4. The main body of your program, organized roughly as,
   (a) All variables and input commands
   (b) All computation,
   (c) All output.

We will work on this throughout this lab. Before coming to lab, go to the Course Materials section on Submitty and download the file `Lab04_files.zip` and unzip it. All of the files in this lab have been compressed into that file. Now, go through the code in `good_program.py` and read up on how to write a good program. Then, look at `bad_program.py` and try to find all the problems this code has. Do this before checking your solution in `bad_program_with_comments.py`. Use this as a guideline for all your code from this point on.

Second, we would like to see comments that explain the main purpose of the program and also tricky steps if there are any. As Python programs are short and generally very easy to understand, you do not need a lot of comments if you have well-structured code and meaningful variable names, but you will need some, and it always helps to define your functions and the meaning of the function parameters.

The final thing is debugging code. Often programmers see debugging as a chore. In fact, debugging is really 80% of programming. So, you must learn to solve problems, read error messages, and figure out how to relate error messages to your code. Debugging is a little like solving a puzzle or being a detective. You need to find clues and think through what could be causing the problem. The first checkpoint will test this.

Checkpoint 1: Using existing functions

First, we will experiment with using functions and debugging code. Create a new directory in your dropbox for Lab 4, and copy the program called `lab04_check1.py` from Submitty into it. Run this code and see that it gives an error message. (Don't worry about what you enter in response to the prompts, you can use any floating point value.)

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)`. The program starts from an initial point and then reads the next `(x,y)` coordinates of an object. It computes the distance between these two and outputs the result.

The code contains many bugs. Often, a program aborts after an initial error and you need to fix it and rerun the program. So, we would like you to go through this process with this code. In fact, even if you can see a bug in the program, we recommend that you do not fix it until you see the error it produces first.

Each time you run the program, read the error message. Sometimes it will be easy to interpret and sometimes not. But very often it will tell you the line number and the exact location. Learn to read these messages and interpret them. You will learn a lot from fixing mistakes, so do not let others tell you what each error means. If you are having trouble figuring them out, read the relevant course notes and think. If you figure out it yourself, you will learn much more than if you are told the same information.

Please take the time to figure out all the bugs. Also, format the output nicely so that it looks like this:

```
The next x value ==> 12
The next y value ==> 12
The line has moved from (10,10) to (12,12)
Total length traveled is: 2.83
```

After you fixed all the bugs, show the corrected program to a TA or a mentor. They will ask you about different errors and discuss strategies for figuring out what they mean.

To complete Checkpoint 1: Show the TA the working version of the program. Be prepared to discuss with them debugging methods and ask for advice.

Checkpoint 2: Restructuring code

In this section, we will restructure a program with the help of a function, and then extend it just a little. First start by copying the program called `lab04_check2.py` and save it in your lab4 folder. Create a copy of it `lab04_check2v2.py` by saving it a second time with the new name. 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 (skew tells you if your running times have outliers in higher values with positive skew or lower values with negative skew. No skew means your running times are pretty similar.). You can see that this program contains repetitive code. If you found a bug in one computation, you would have to fix it in 5 places. Instead, you can put all this computation into a function improving both readability and maintainability.

Write a function that takes as input all the necessary data for the repeated code, does the computation and returns the skew number. Your function definition must come before any other code in the program. Your function must have no print statements. What should the arguments for this function be? Make sure your function has no global variables.

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

Next, write some additional code. We will experiment with functions that do not return anything. See the end of this lab description on the distinction between the two types of functions. Write a function that takes the name of a person and all five running times. The function must then print min, max and the average running time of the remaining 3 values (after subtracting min and max), but it should not return anything. Call this function in your program for each person. Here is the additional output you should get by calling the function:

```
Stan's stats-- min: 29, max: 35, avg: 33.0
Kyle's stats-- min: 28, max: 31, avg: 29.3
Cartman's stats-- min: 31, max: 36, avg: 32.7
Kenny's stats-- min: 31, max: 35, avg: 33.0
Bebe's stats-- min: 27, max: 30, avg: 28.7
```

`What happens if you print the results from this new function?` The value **None** is the return value from a function with no return value.

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

Checkpoint 3: Intro Icebreaker!

This checkpoint is aimed at making you more familiar with your lab-mates. The TAs will break the class off into pairs and give the class some time to get to know one another. Once the allotted time has passed, each person will introduce their partner instead of themselves.

Checkpoint 4: Prey and Predator Population

Please come to lab for the last checkpoint.