**CLPS Matlab Programming Workshop**

**Session 1 of 3**

Wednesday, November 27, 2018

Written by Jae-Young Son (jaeyoung@brown.edu)

**Introduction:**

One of the simplest (yet most profound) insights you can have about computers is that they are nothing more than information processing units. To the untrained eye, programming might look like magic (just as you might utter a gibberish incantation to mysteriously refill your coffee mug, you type some gibberish commands to mysteriously animate a circle moving around the screen), but this could not be further from the truth. Everything a computer does is the direct result of an explicit computational process: you provide the computer with some informational input, command the computer to process that information in some meaningful way, and then return an informational output that is deterministically related to the commands you specified.

Today’s coding exercises are designed to reinforce the notion that computers are deterministic information processors. By the time you complete all of these coding exercises, you should have a good sense for how to use variables, functions, arrays and matrices, and control logic.

As you are writing each program, keep in mind that I will be doing everything in my power to break your program.

**Exercise #1: Add two numbers**

Write a program that accomplishes the following objectives:

* Asks the user to input two numbers
* Adds those two numbers together
* Displays the result of the calculation
  + See if you can make the display message look good! Instead of simply displaying “5”, can you instead display something like “The answer is 5” or “3 + 2 = 5”?
  + Hint: if you’re getting errors while using disp, try looking through the documentation using the help command. How many arguments does disp take? Try looking up what the square brackets do (pg. 35 of the tutorial) to solve this problem.

Useful functions:

* input
* disp
* num2str

Notes:

* There are many possible solutions (as is always the case with programming), but the most straightforward should only be 4 lines of code.

**Exercise #2: Four-function arithmetic calculator**

Program objectives:

* Asks the user which operation they’d like to perform (addition, subtraction, multiplication, or division)
* Asks the user to input two numbers
* Displays the result of the calculation

Useful functions:

* Is input now returning an error? Try looking through the documentation using the help command to see why.
* strcmp
* upper

Notes:

* You need to deal with all of the “edge cases.” From the programmer’s point of view, there is no such thing as user error! Your job is to idiot-proof your program so that your users can never break it.
* What happens if the user types “add” or “plus” instead of “addition”?
* What happens if the user types “sUbTrAcTiOn” or “SUBTRACTION” instead of “subtraction”?
* What happens if the user types something nonsensical like “turkey”?

**Exercise #3: Quadratic equation calculator**

You may remember solving the quadratic equation from high school algebra. You will now write a program that can solve for x given user-defined a, b, and c.



Program objectives:

* Asks the user to enter values for a, b, and c
* Calculates the two solutions for x (which may or may not be the same)
* Displays both solutions

Useful functions:

* sqrt

Notes:

* If the quantity under the square root is negative, the equation does not have any real solutions. Also, if a=0, assume that the equation does not have any real solutions. While Matlab can return complex numbers, let’s assume that your user is only interested in real solutions. Your program should kindly explain to the user that their input does not have real solutions (when appropriate).
* For pedagogical reasons, you should explicitly spell out the formula in your program. However, you should also try using the roots function to verify that your solutions are bug-free.

**Exercise #4: Summary statistics calculator**

If you’re a beginner, this is the first program you will write that we might consider properly sophisticated. In past calculator programs, you’ve assumed that the user only wants to work with two or three numbers. In statistics, however, two data points are rarely satisfactory. Theoretically, you could ask users to specify at the beginning of the program how many data points they have, but nobody wants to sit down and count their data points ahead of time. Instead, what you can do is ask users to keep inputting numbers until they type a special phrase to indicate that they’re done. In the lingo of programming, this special phrase is known as a “sentinel.”

Program objectives:

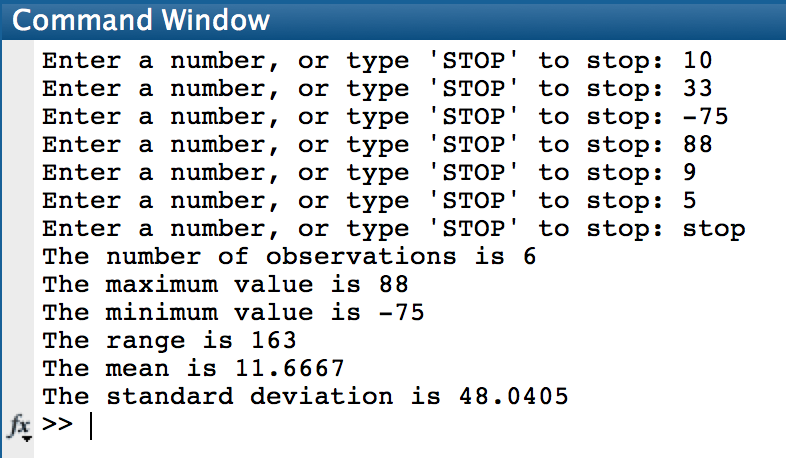
* Continually asks the user to either input a number or the sentinel (the word “STOP”)
* Records the value of every number the user enters
* Once the user enters the sentinel, computes the following statistics:
  + N (number of data points)
  + Maximum value (largest number entered)
  + Minimum value (smallest number entered)
  + Range (max – min)
  + Mean (arithmetic average)
  + Standard deviation
* Displays the computations in a user-friendly and easy-to-understand way

Notes:

* Because the sentinel is a string, input should take strings, not numbers.
* Because you’ve told input to take strings, you can’t perform arithmetic operations on the numbers. How might you get around this problem?
* As before, you need to ensure that your program recognizes the sentinel regardless of how it’s capitalized. You also need a way to deal with cases when the user enters something nonsensical.
* How do you store values in arrays if you aren’t sure how many times a loop will iterate?

Extra credit:

* Write a version of this program that displays “the statistics so far” every time a user enters a new number.



*An example of what your script might look like to your user*

**Exercise #5: Kind messages**

This is a fairly straightforward variant of Exercise #4 that gives you more practice with for loops and matrix indexing.

Program objectives:

* Continually asks the user to either input a name or the sentinel (“STOP”)
* Once the sentinel is entered, the program displays the following message with each name that is entered: “I appreciate NAME’s impact on my life!”

**Exercise #6: Text analysis**

This is an optional exercise for people who really want a challenge! Text analysis has recently become a popular method in the digital humanities and social sciences for quantifying patterns in large corpora. Although advanced text analysis is far beyond the scope of this workshop, you have (hypothetically) learned enough to build a simple text analysis tool.

Program objectives:

* Reads in a user-specified text file
  + If you need inspiration, how about downloading a plaintext copy of “The Tempest” by Shakespeare? https://raw.githubusercontent.com/martin-gorner/tensorflow-rnn-shakespeare/master/shakespeare/tempest.txt
* Separates words from whitespace, punctuation, paragraph breaks, etc.
* Stores all words in one giant cell array
* Possible text analysis operations (approximately from easiest to hardest):
  + Count the total number of words in the corpus
  + Find the longest word used in the corpus
  + Count how many times each unique word is used