**Assignment 1 – Getting Started**

The primary goal of this assignment is to make sure you have a working version of R that you are able to access. Besides this, you will earn how to make and run a script and how to print output to the user window.

If you plan on using R in the ESF computer labs, accessing R should be as easy as clicking on R in the Windows program list.

If you plan on running R from your own personal computer, R can be downloaded at:

https://www.r-project.org/

*Entering Code at the Command Line*

Programming is like learning a new sport, a musical instrument, or how to navigate around a new city. You might start out following some very specific instructions, but you don’t become more accomplished until you become comfortable enough to start playing around and trying things on your own.

One critical but somewhat subtle aspect of programming is testing out small pieces of code before placing them in a larger program. When you write a program, you don’t just type everything in, step back, and expect it to work. Good programming involves testing your code as you incrementally add new pieces to it.

In a language like R, one way to test out code is to use it at the command line. Any line of code you can run within your program ( a “script”) can also be run at the command line. By trying a piece of code at the command line, you can immediately see if it is working how you expect, if all your syntax is right, etc.

The command to write text is: **print(“ WORDS TO PRINT “, quote=FALSE)** where quote=False removes the quotes when the words are printed in the user window. Let’s try this at the command line. Try:

Print(“Welcome to the new semester”, quote=FALSE)

Do you get what you expect? What if you try it again, but now set FALSE to TRUE?

Any time you have a built-in command in R that you would like to learn more about, you can type: **?”command name”()** . Let’s try this at the command line:

?print()

What do you get? It should bring up a webpage providing additional details on using print(). For instance, if you were printing numbers, you could include an argument that would limit the number of significant digits (i.e. “digits=2”). While a useful reference, some of the information provided can admittedly be overly technical for beginning programmers.

*Writing a Script*

A script is a self-contained program. When a script is called from the command line, all code in the script will be executed (assuming there are no bugs). Creating a script is just like creating in a document in Microsoft. In the drop down “File” menu, select “New Script”. It should open a new window. Save this document to a central location where you plan to keep any files related to R programming.

In the script, you can type code just as you would at the command line. Your task is to create a picture, using letter and symbols on the keyboard.

Something like (a wind turbine):

O ~ ~ ~ ~ ~ ~ ~

O ~ ~ ~ ~

COOOOO> ~ ~~ ~ ~ ~ ~

HH O ~ ~ ~ ~ ~ ~ ~

HH O ~ ~ ~ ~ ~ ~

HH

HH

HH

HH

Once you write your script, you need to “run” it or “call” it. To call our script from the command line, we can use the **source** command along with the path to the file directory. Your file path will vary depending on where you have files stored on your computer. The example code for my laptop is:

source("C:\\Users\\sbshaw\\Documents\\ERE335\\TestFig.R")

Objectives:

1. First, make sure you test out **print** at the command line.

2. Write an R script that creates a picture using letters and symbols on the keyboard. In the comments of the R script, **indicate where you completed this assignment**: on your own personal computer or on a lab computer.

**To receive credit for the assignment,** you need to: 1) hand in a copy of your R script and 2) a copy of the output of your script (use a screenshot). You must hand this in by **noon on Friday, Aug 30** (so we can make sure everyone has access to a working version of R).

**Assignment 2 - MadLibs**

The primary goal of this assignment is to become familiar with the concept of assigning and using variables. Using variables allows us to avoid repetition in the code, makes it easier to update and revise the code, and makes the code more generalizable.

Besides practicing using variables, we will also use scripts and learn a little about inputting user information into a script and then outputting information from a script to the user window.

The idea of today’s assignment is to automate a MadLib. You might remember MadLibs from long car trips; you would pick words of a specified part of speech (noun, verb, etc. ) and then have them placed into a paragraph of some unknown context. In the days of pencil and paper, someone had to transcribe the words into the text. The idea for this assignment is to have the code automate the process of placing words into the text, while also allowing for certain selected words to be repeated multiple times in the text (something that was tedious when doing this by hand).

We will make use of the new command: **paste**. This command concatenates – links together – individual objects. We will use it to link together text objects called strings. For instance:

paste(“How”, “Does”, “This”, “Work?”)

results in a single object: “How does this work?”

We will also make use of the command **readline**. Readline requests a user to enter text at the command line as the script runs. For instance:

Age <- readline( prompt = “How old are you?”)

would ask the question “How old are you?” at the command line and the user-entered response would then be saved to the variable called “Age”. Try out the code and see what happens (note, sometimes R does not like the quotation mark syntax if you cut-and-paste). You can check the value contained within the variable Age by then typing Age at the command line.

Objectives:

1. First, start by testing out **paste** and **readline** at the command line.

2. Before starting to code, develop pseudocode that lays out the sequence and logic of the code. Remember, the intent of this exercise is to practice using variables, and the pseudo-code should indicate how variables come into play in this program.

3. Write an R script that requests from the user specified words of a given part of speech and that then outputs the completed text with the words filled in.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R script, and 3) show a TA a working version of your code. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.

The MadLib text is as follows (with the part of speech of the missing words shown in brackets):

SUNY ESF has some of the [superlative] [plural noun 1] in the country. The [plural noun 1] are known for their [ adjective 1] [noun 2]. The [adjective 1] [ noun 2] was refined by thousands of [unit of time] of tireless [noun 3]. However, if the [plural noun 1] had done less [noun 3], they SUNY ESF may have a better [noun 4] club.

**Assignment 3. Byte and Hexadecimal Calculators**

The primary goal of this assignment is to test our new knowledge of binary and hexadecimal number systems by writing some short scripts. We aren’t learning any new broad concepts (we just need to use basics of vectors, variables, mathematical operators), but it will make you practice the idea of using pseudo-code and debugging to get working programs.

Objectives:

1. Write a short script that converts a known 8-bit sequence (a byte) into a base-10 value. Show your code works for the byte: 1 1 1 0 0 1 1 1

Have the script print out the byte as well as the base-10 value.

2. Write a short script that converts a known 16-bit sequence into a hexadecimal value. Show your code for the sequence: 1 1 1 0 0 1 1 1 0 1 0 1 0 0 1 1

You’ll need to use a built-in function to help make your binary to hexadecimal conversion. To convert from a base 10 digit to hexadecimal, you can use **as.hexmode**. For instance, as.hexmode(15) results in “f”. Note, you will also need to use some indexing of vector values.

Have the script print out the binary sequence as well as the hexadecimal value.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R script, and 3) show a TA a working version of your code. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.

**Assignment 4. If/then & Arrays**

The primary goal of this assignment is to apply your knowledge of if/then statements as well as to use array indexing to select specific segments of array.

You will write two scripts. The first is simply to practice the idea of array indexing. The second – which should be a little harder – requires using both array indexing as well if/then statements.

Objectives:

1. Write a script that populates an array to create a map of features in a 21 by 21 block city. Print the bottom layer of the map. Assume the upper left corner is the origin and the top of the map is North.

The features that should be described in the map include:

1. A north-south running river that bisects the city. Label these cells with an “R”.
2. A square-loop highway centered in the city that is 9 by 9 blocks in its dimensions. Label these cells with “HW”. Note, the road can be built over the river.
3. A business district with 5-story office buildings with the loop highway. These should be labeled with “5S” and it should include 3-dimensions.
4. City Hall located on the northwest side of the northwest corner of the loop highway. Make City Hall a 2 by 2 block building. This should be label with “CH”.
5. All other open spaces should be labeled as “O”.

2. Write a script that takes the outcome of a sequence of 10 baseball pitches and determines how many runners advance to a base. The possible outcomes are: foul (F), hit (H), ball (B), and strike (K). Test the script using a vector of pitch outcomes as given by: Outcomes <- c( F, H, B, B, K, K, B, B, H, B). The script should print the number of total hitters that advance. Note, the array indexing required here could really just be thought of as vector indexing and will be much simpler than that used in the city map script.

Recall the basic rules of baseball. A hit implies the hitter advances to base. Three strikes make an out, and the hitter does not advance. A foul counts as a strike up the accumulation of two strikes (i.e. a runner cannot be out on a foul). Four balls leads to a walk and the hitter advances.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R scripts, and 3) show a TA a working version of your code. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.

**Assignment 5. Loops**

The primary goal of this assignment is to practice using loop structures in your code.

Objectives:

1. A microbial colony increases in mass 150% each hour plus a fixed addition of 0.01 kg of microbes are added each hour. The colony starts off with a mass of 0.01 kg. Write a script that calculates the colony’s mass after 20 hours.

2. Write a script to score across multiple frames of bowling. You will need to use a loop as well as matrix indexing. Test your script on the following five frames: Frame 1 – 10 & 0 ; Frame 2 – 10 & 0; Frame 3 – 5 & 5 ; Frame 4 – 3 & 4; Frame 5 – 8 & 0.

Recall the rules of scoring bowling. In each frame, a bowler has two chances to knock down 10 pins. If all 10 pins are knocked down in the first turn (a strike), the frame is over but the sum of pins knocked down in the next two turns is included in the score for this frame. If all ten pins are knocked down during both turns in a frame (a spare), the sum of pins knocked down in the next turn is included in the turn for this frame. If less, than 10 pins are knocked down in a frame, the score for the frame is the sum of the pins knocked down.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R script, and 3) show a TA a working version of your code. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.

**Assignment 6. Functions**

The primary goal of this assignment is to practice writing functions.

Objectives:

1. Write a function to convert a streamflow given in cubic feet per second into cm per day. This requires dividing the volumetric discharge through by watershed area. Test the function for a watershed of 125 mi2 with a streamflow of 550 cfs.

2. Write a function to convert a coordinate given in degrees, minutes, and seconds into a decimal degree value. Test the function for a coordinate of 35◦ 35’ 15” .

3. The At-Bat script you wrote in Assignment 4 wasn’t a particularly efficient way to write code, but now that you know how to write functions you can improve your code. Rewrite the At-Bat script from Assignment 4 but now use functions as appropriate.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R script, and 3) show a TA a working version of your code. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.

**Assignment 7. Tic-Tac-Toe**

The assignment doesn’t introduce anything new, but it does require you to integrate and practice all the concepts you have learned so far. To efficiently write this code, you will most likely need to use multiple functions.

Objective: Write a script that allows two people to play tic-tac-toe. The script should allow each player to take sequential turns and to interactively enter their choice of position at the command line on each turn. For each turn, the script should output the set-up of the board. Note, you don’t have to use “x” and “o” as the symbol of each player. Note, the main body of this code should be able to be written in less than 30 lines.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R script, and 3) show a TA a working version of your code. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.

**Assignment 8. Dry Days Due: 10/31/2018**

R is known as data science software, but we haven’t really used it do data analysis yet. In this assignment we will practice using standard features of R to do data analysis (reading in data files, using dataframes, and plotting) as well as applying standard programming concepts.

Objective: Farmers are not only interested in the total amount of rain in a growing season but also the duration of rain free periods. There is speculation that anthropogenic climate change may not always change total rainfall amounts but may lead to an increase in the duration of rain free periods. To evaluate this possibility of extended dry periods, it would be ideal to have a program that could automatically identify dry periods over many years of data.

You want to write a script that reads in a meteorological data file (Batavia\_mod.txt) with multiple years of daily rainfall data and that then identifies the duration and number of dry periods in each year. We will define a dry period as any continuous period with less than 0.03” of precipitation on any given day. We don’t need to identify very short dry periods; have the script ignore dry periods that are less than 8 days in duration.

The script should store the year and duration of the 8+ dry periods in a matrix. Additionally, the script should generate a figure that summarizes the findings of the analysis.

**To receive credit for the assignment,** you need to: 1) hand in a typed version of your pseudo-code, 2) hand in a copy of your R script, 3) print the first 10 lines of a matrix showing the year and duration of the dry days, and 4) develop a figure that summarizes your results. When you hand in your pseudocode and script, be sure to indicate anyone you may have worked with on your code.