*\*Please save this file as “LAST NAME\_Assignment 1.docx”*

***Open-Ended Responses***

1. **I.1** If you are defining an object called **vec.x**, you can do so by typing it into a script file first and then executing or by typing it directly into the console. Which way is better for reproducibility and why?

Typing into the script will ensure that all commands are saved for future running. When you run something into the console, it will be saved to the history, but this can be unruly to re-run and reproduce the exact environment that you have. This is best done by coding the final product into the script file.

1. **I.5** Explain how R came up with the following result:

x <- 1:10

y <- 1:3

x-y

[1] 0 0 0 3 3 3 6 6 6 9

Warning message:

In x - y : longer object length is not a multiple of shorter object length

The object **x** is longer than the object **y**, so when it’s asked to subtract **y** from **x**, a warning is given. The result actually comes from subtracting the first, second, and third elements of **y** and subtracting it from the same elements of **x** (1 – 1 = 0; 2 – 2 = 0; 3 – 3 = 0). But then, **y** begins over again along **x** and the first, second, and third elements of **y** are subtracted from the fourth, fifth, and sixth elements of **x** (4 – 1 = 3; 5 – 2 = 3; 6 – 3 = 3). This goes on until the last (tenth) element of **x** where the first element of **x** is subtracted (10 – 1 = 9).

1. **I.6** Explain the behavior of the **round()** function observed below where 0.5 is rounded down, but 1.5 is rounded up.

round(.5)

[1] 0

round(1.5)

[1] 2

According to the help file for the **round()** function, it uses the IEC 60559 standard which states that for the rounding off of 5, it will go to the even digit. Therefore, 0.5 won’t be rounded up to 1 (odd) and will instead be rounded to 0 but 1.5 will be rounded to 2 (even) and not 1 (odd). What a whacky rule!

1. **I.7** A researcher wants to create a data set by sampling 100 integers ranging from 50 to 75 with replacement, center those data (subtract the mean from each data point), and then calculate the centered mean divided by the centered standard deviation. Spot the silent error in the following code written to do this:

1 data <- sample(50:75, 100, replace = TRUE) # sample the data

2 data.cen <- data - mean(data) # center data on mean

3 mean(data) / sd(data) # calculate mean / sd

The silent error is in line 3: The intention was to calculate the mean of the centered data divided by the standard deviation of the centered data. The centered data has been stored to an object called **data.cen**, but it is actually the object **data** (uncentered mean and sd) that is being computed.

1. **I.7** Why does the following generate an error? Special note: if you copy/paste from this word document, it brings what are called “smart quotes” which R can’t recognize. You might need to type the quotes manually.

x <- c(1, 5, 3, 4, “3”) # runs fine

sum(x) # error generated

Typing “3” indicates a character element. Even one character element turns the whole vector to character, so when the sum function is called, it cannot take the sum of characters.

1. **1.8** Below are two sections of code that accomplish the same thing. Which one would you think is the “best” way? There isn’t necessarily a right answer here, I just want to hear your rationale.

x <- sample(-50:50)

# Code Section 1

x.fil <- x[x > 0] # filter out all negative values

x.fil.sq <- x.fil^2 # square results

mean(x.fil.sq) # compute mean

# Code Section 1

mean((x[x > 0])^2) # filter, square, compute mean

The first section is more readable, but creates more objects, which can get cumbersome. However, you can also use these objects in the future, so if you need to use the objects in the future, you can do that. The second section is efficient and doesn’t create excess objects. However, it does this at the expense of readability; three things are done in one line and it can be hard to parse out exactly what happened and why.