STT 301 In-Class Exam Solutions
October 10, 2018

Name: _____ **PID:** _____

Directions: The in-class portion of the exam is worth 48 total points. Show as much work as possible on every problem to earn partial credit. Incorrect answers that appear with no work will earn 0 points.

The exam consists of three parts. Read the directions for each part.

Part I (12 points)

Answer six of the eight questions that follow below. Write "Do Not Grade" in the blank space of the questions you do not want me to grade. If you answer all eight questions, I will only grade the first six questions.

Determine if each statement is true or false. Please write out "True" or "False". Do not put "T" or "F". If you claim the statement to be false, then you must provide justification. No justification is needed for statements you claim to be true.

1) Lists are heterogeneous structures, while data frames are homogeneous structures. (2 points)

   False. Both are heterogeneous structures. Each column of a data frame can be a vector of a different variable type.

2) If `x` is a character vector and `y` is a numeric vector, then `c(x, y)` will result in an error because R can not concatenate vectors of different variable types. (2 points)

   False. R will convert all componenets to be of the same variable type. No error will occur.

3) Suppose a vector `x` is subset with a logical vector `y`. The code `x[y]` will return all the components of `x` where `y` has a `TRUE` component. (2 points)

   True.

4) Suppose `my_list` is a list with a vector named `x` as its first element. `my_list$x`, `my_list[1]`, `my_list[[1]]` all return the same result. (2 points)

   False. `my_list[1]` returns a list with one element. The other two commands return the vector `x`.

5) To check for inequality in R you use `!=`, and to check for equality in R you use `=`. (2 points)

   False. To check for equality `==` is used. The single `=` is for object assignment.

6) `1/0` and `0/0` both result in `NaN`. (2 points)
   False. `1/0` will result in `Inf`.

7) The three main features of a function are its name, arguments, and body. (2 points)
   True.

8) A while loop can result in an infinite loop if the condition is always true. (2 points)
   True.

Part II (16 points)

Answer four of the five questions that follow. Write "Do Not Grade" in the blank space of the question you do not want me to grade. If you answer all five questions, I will only grade the first four questions.

The `>` you see on a given line before any code is the prompt as you would see in your RStudio console.

1) Which R chunk option will show the code, but not the result of `table(iris$Species)`? (4 points)

```
> table(iris$Species)
```

Chunk option: `eval = FALSE`. Other answers were acceptable.

2) Consider a vector `z`. What does `mean(!is.finite(z))` tell you about the vector `z`? (4 points)

The proportion of infinite components in the vector `z`.

3) Consider the below code.

```
> x <- c(0.3, 0.3, 4, 4, 0.3)
>
> if (x == 0.3) {
+     x <- 3
+ }

Warning in if (x == 0.3) {:  the condition has length > 1 and only the first
element will be used

> x

[1] 3
```

Why is `x` not a vector with 3, 3, 4, 4, 3 as its components?

In an `if` statement the condition must evaluate to single TRUE or FALSE value. Here, `x == 0.3` will result in a vector of TRUE and FALSE values. In R, only the first element of that vector will be checked for the condition. Since it happens to be true, the body of the `if` statement assigns `x` the value 3.

To do this correctly, R's vectorized capabilities should be used or a loop should be used.

4) Why is `shopping_list[1][2]` `NULL`?. (4 points)

```
> shopping_list <- list(food = c("apples", "milk", "salmon"),
+                       price = c(3.4, 1.56, 7.99),
+                       store = c("Meijer"))
> shopping_list

$food
[1] "apples" "milk"   "salmon"

$price
[1] 3.40 1.56 7.99

$store
[1] "Meijer"

> shopping_list[1][2]

$<NA>
NULL
```

`shopping_list[1]` returns a list with one element in the list, the vector food. `shopping_list[1][2]` tries to return the second element of the list `shopping_list[1]`, but there is no second element. Hence the result is null.

```
shopping_list[1]

$food
[1] "apples" "milk"   "salmon"

shopping_list[1][2]

$<NA>
NULL
```

5) Consider the small data frame, `exam1`, given below.

```
> exam1

  score  name pass undergrad
1    53  Abby  yes      TRUE
2    45 Katie  yes      TRUE
3    52  Bill  yes      TRUE
4    44  Jrue  yes      TRUE
```

I want to check the variable type for each column in `exam1`. Below is the result of what my code should return. List three mistakes in my code, given further below, as to why it will not produce the desired result? (4 points)

```
[1] "double"    "character" "character" "logical"
```

```
# my beautiful code

varaible.type <- NULL
for (i in 1:dim(exam1){
  variable.type <- typeof(exam[i , ])
}

variable.type
```

- Incorrect spelling of `variable.type`
- `dim(exam1)` should be `dim(exam1)[2])`
- `exam` does not exist

Part III (20 points)

Answer five of the seven questions that follow. Write "Do Not Grade" in the blank space of the questions you do not want me to grade. If you answer all seven questions, I will only grade the first five questions.

1) Turn the below snippet of code into a function that has one argument, `x`, as a numeric vector.

```
below_mean <- sum(x < mean(x))
above_mean <- sum(x > mean(x))
at_mean <- sum(x == mean(x))
c(below_mean, above_mean, at_mean)
```

```
mean_spread <- function(x) {
    below_mean <- sum(x < mean(x))
    above_mean <- sum(x > mean(x))
    at_mean <- sum(x == mean(x))
    return(c(below_mean, above_mean, at_mean))
}
```

2) Consider the code below.

```
j <- 0
i <- 1
results <- 0

while (j < 4) {
    results[i] <- j + (j - 1)
    i <- i + 1
    j <- j + 1
}

results
```

If possible, re-write the above while loop as a for loop. If it is not possible, explain why. (4 points)

```
results <- 0
j <- 0
for (i in 1:3) {
    results[i] <- j + (j - 1)
    j <- j + 1
}
results

[1] -1  1  3
```

3) Consider the code below.

```
k <- 1
repeat {
    # randomly select 'heads' or 'tails' (each with probability 0.5)
    coin.result <- sample(c("heads", "tails"), size = 1, replace = T)

    if (coin.result == "tails") {
        break
    }
    k <- k + 1
}
```

If possible, re-write the above repeat loop as a for loop. If it is not possible, explain why. (4 points)

This can not be written as a for loop. The number of loop iterations based on the code above is random, and a for loop requires a known number of loop iterations.

4) Consider the function given below.

```
my_fcn <- function(a, b, x = 2, power = TRUE) {
    if (power) {
        z <- a^x
    } else {
        z <- b * x
    }
    return(z)
}
```

What are the results of the below function calls? (4 points)

```
my_fcn(a = 2)

[1] 4

my_fcn(a = 2, power = FALSE)

Error in my_fcn(a = 2, power = FALSE): argument "b" is missing, with no default
```

5) Consider the list below.

```
> my_list

$chicks
  weight      feed
1    179 horsebean
2    160 horsebean
3    136 horsebean
4    227 horsebean
5    217 horsebean
6    168 horsebean

$x
[1] -0.02071030  0.72535847 -0.08308397  2.08111841  0.32942064

$y
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

$message
[1] "This is the last element of the list."
```

What is the result for each of the below lines of code? (4 points)

```
> my_list[c(3, 4)]

$y
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

$message
[1] "This is the last element of the list."

> my_list[["chicks"]][3, 1]

[1] 136
```

6) Re-write the below code using the vectorization capabilities of R. Your code should only require one line and not depend on the length of `x`. (4 points)

```
> x <- c("Philly", "philly", "PHILLY", rep("filly", 2))
>
> for (i in 1:length(x)) {
+     if (x[i] == "filly") {
+         x[i] <- "philly"
+     }
+ }
>
> x

[1] "Philly" "philly" "PHILLY" "philly" "philly"
```

```
x[x == "filly"] <- "philly"
```

7) Consider the code below.

```
> i <- 1
> j <- 0
> k <- 1
> results <- NULL
> while ((i < 5) & (j <= i)) {
+     results[k] <- abs(j - i)
+     j <- i^2
+     i <- i * 2
+     k <- k + 1
+ }
>
> results
```

Give the value of `results` on the last line of the above code. (4 points)

```
> i <- 1
> j <- 0
> k <- 1
> results <- NULL
> while ((i < 5) & (j <= i)) {
+     results[k] <- abs(j - i)
+     j <- i^2
+     i <- i * 2
+     k <- k + 1
+ }
>
> results

[1] 1 1 0
```