Section 1

Jacob Jameson

Notes build on previous TFs (Ibou Dieye, Laura Morris, Amy Wickett & Emily Mower)

The following code is meant as a first introduction to R. It is therefore helpful to run it one line at a time and see what happens. To run one line of code in RStudio, you can highlight the code you want to run and hit "Run" at the top of the script.

On a mac, you can highlight the code you want to run and hit Command + Enter. On a PC, you can highlight the code you want to run and hit Ctrl + Enter. If you ever forget how a function works, you can type `?` followed immediately (e.g. with no space) by the function name to get the help file.

Part 1: R Fundamentals

Values can be assigned names and used in subsequent operations. Instead of the traditional "=" sign, the convention in R is "`<-`". The "`=`" sign also works, but I will use "`<-`" to be consistent with convention.

The general form for calling R functions is:

```
# FunctionName(arg.1 = value.1, arg.2 = value.2, ..., arg.n = value.n)
```

Part 2: Vectors

First, we will learn how to make different types of vectors. Our first vector will contain integers 1 through 4:

```
vec.1 <- c(1, 2, 3, 4)
print(vec.1)
```

```
## [1] 1 2 3 4
```

In R, we use square brackets for indexing. So, for example, if we want to print the 1st element of our vector:

```
print(vec.1[1])
```

```
## [1] 1
```

If we want to print the 4th element of our vector:

```
print(vec.1[4])
```

```
## [1] 4
```

If we want to print the 1st and the 4th elements in one go, we make a vector with the desired indices and place that index vector within square brackets:

```
print(vec.1[c(1,4)])
```

```
## [1] 1 4
```

An alternative way to create `vec.1` would be using the `seq()` command, which allows us to generate a vector according to a sequence:

```
print(seq(4))
```

```
## [1] 1 2 3 4
```

```
print(seq(-4))
```

```
## [1]  1  0 -1 -2 -3 -4
```

```
vec.2 <- seq(4)
print(vec.2)
```

```
## [1] 1 2 3 4
```

```
print(seq(from = 100, to = 120, by = 5))
```

```
## [1] 100 105 110 115 120
```

```
# Help file for seq function
?seq

# Breaking the sequence command into multiple lines
print(seq(from = 100,
          to = 120,
          by = 5))
```

```
## [1] 100 105 110 115 120
```

```
print(1:10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
print(10:1)
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

Vectors don't have to be numeric. They could also be character/string vectors:

```r
word.vec <- c("Hello", "Time To", "Learn", "R", "!")
print(word.vec)
```

```
## [1] "Hello"   "Time To" "Learn"   "R"       "!"
```

```r
# Using the which() function
print(which(word.vec == "Learn"))
```

```
## [1] 3
```

```r
print(which(word.vec == "Hi!"))
```

```
## integer(0)
```

```r
# Finding the length of a vector
print(length(vec.1))
```

```
## [1] 4
```

```r
print(length(seq(10)))
```

```
## [1] 10
```

```r
print(length(seq(from = 100, to = 120, by = 2)))
```

```
## [1] 11
```

```r
# Calculating statistics about vectors
print(mean(vec.1))
```

```
## [1] 2.5
```

```r
print(median(vec.1))
```

```
## [1] 2.5
```

```r
print(min(vec.1))
```

```
## [1] 1
```

```r
print(max(vec.1))
```

```
## [1] 4
```

```r
# Variance and standard deviation
print(var(vec.1))
```

```
## [1] 1.666667
```

```r
print(sd(vec.1))
```

```
## [1] 1.290994
```

```r
# Comparing vectors
vec.4 <- c(1, 4, 9, 16)
print(vec.4)
```

```
## [1]  1  4  9 16
```

```r
print(vec.1 == vec.4)
```

```
## [1]  TRUE FALSE FALSE FALSE
```

```r
# Checking if two vectors are exactly the same
print(all.equal(vec.1, vec.2))
```

```
## [1] TRUE
```

```r
print(all.equal(vec.1, vec.4))
```

```
## [1] "Mean relative difference: 2.222222"
```

Part 3: Logical statements

"if statements" can be very useful. They work as follows:

```
if (the logical statement in these parentheses is TRUE) {
  do this}
else {
  do that
}
```

Let's try it.

```r
## Example 1:
if (2 + 2 == 5) {
  print("Yikes")
} else {
  print("Good job!")
}
```

```
## [1] "Good job!"
```

```r
## Example 2:
if (vec.1[2] == 2) {
  print("Hello")
}else {
  print("Goodbye")
}
```

```
## [1] "Hello"
```

Part 5: Matrices

To make a matrix, use the `matrix()` command. The first element fed in is the data you want to put in matrix form. Then, you specify the number of rows and columns. By default, it fills information down the columns, but you can tell it to do by row

```r
mtx.1 <- matrix(vec.1, nrow = 2, ncol = 2)
print(mtx.1)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```r
mtx.2 <- matrix(vec.1, nrow = 2, ncol = 2, byrow = TRUE)
print(mtx.2)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

Note that `mtx.2` is the transpose of `mtx.1`. If you want to transpose a matrix, you can use `t()`

```r
print(mtx.1)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```r
print(t(mtx.1))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

As with vectors, you can check if two matrices are equal

```r
print(mtx.1 == mtx.2)
```

```
##       [,1]  [,2]
## [1,]  TRUE FALSE
## [2,] FALSE  TRUE
```

```r
print(all.equal(mtx.1, mtx.2))
```

```
## [1] "Mean relative difference: 0.4"
```

```r
print(mtx.1 == t(mtx.2))
```

```
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
```

```r
print(all.equal(mtx.1, t(mtx.2)))
```

```
## [1] TRUE
```

Matrices are indexed by `[row,column]`

```r
print(mtx.1[1,2])
```

```
## [1] 3
```

```r
print(mtx.2[1,2])
```

```
## [1] 2
```

Let's make a bigger matrix

```r
mtx.3 <- matrix(c(vec.1, vec.4), nrow = 4, ncol = 2)
print(mtx.3)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    4
## [3,]    3    9
## [4,]    4   16
```

Part 6: Random numbers

You can also generate random numbers in R. However, one concern with analyses. done using random numbers is that you might not be able to reproduce them. One way to avoid this is to set the "seed". Here's a reference for random seeds: https://en.wikipedia.org/wiki/Random.seed

```r
set.seed(222)
```

We can generate random numbers from all kinds of distributions. For now, we will generate a random normal variable. If I don't specify a mean or variance, it will assume mean = 0, standard deviation = 1.

```r
norm.var1 <- rnorm(1)
print(norm.var1)
```

```
## [1] 1.487757
```

Alternatively, we can specify the mean and standard deviation

```r
norm.var2 <- rnorm(1, mean = 100, sd = 10)
print(norm.var2)
```

```
## [1] 99.98108
```

The first element inside the parentheses is how many random variables I want to draw. For example, I could draw 10

```r
norm.vec  <- rnorm(10, mean = 5, sd = 1)
print(norm.vec)
```

```
##  [1] 6.381021 4.619786 5.184136 4.753104 3.784439 6.561405 5.427310 3.798976
##  [9] 6.052458 3.694936
```

You can also draw from other distributions, like the uniform distribution

```r
uni.var1 <- runif(1)
print(uni.var1)
```

```
## [1] 0.2442779
```

Part 7: Data frames

There are lots of great datasets available as part of R packages. Page 14 of Introduction to Statistical Learning with Applications in R. Table 1.1 lays out 15 data sets available from R packages. You can install a package in R using the `install.packages()` function. Once a package is installed you may use the `library` function to attach it so that it can be used. Then, every time you want to use the package, you use `library(package.name)`

```r
library(ISLR)
```

Sometimes we will use outside datasets, not contained in R. In order to read data from a file, you have to know what kind of file it is. The table below lists functions that can import data from common plain-text formats.

Data Type	Function
comma separated	`read.csv()`
tab separated	`read.delim()`
other delimited formats	`read.table()`
fixed width	`read.fwf()`

```
college.data  <- College
?College
```

Let's learn about our data. To get the names of the columns in the dataframe, we can use the function colnames()

```
colnames(college.data)
```

```
##  [1] "Private"     "Apps"        "Accept"      "Enroll"      "Top10perc"
##  [6] "Top25perc"   "F.Undergrad" "P.Undergrad" "Outstate"    "Room.Board"
## [11] "Books"       "Personal"    "PhD"         "Terminal"    "S.F.Ratio"
## [16] "perc.alumni" "Expend"      "Grad.Rate"
```

To find out how many rows and columns are in the dataset, use `dim()`. Recall that this gives us Rows followed by Columns

```
dim(college.data)
```

```
## [1] 777  18
```

To find out what type of data is in each column, we can use `typeof()`

```
typeof(college.data[,1])
```

```
## [1] "integer"
```

```
typeof(college.data[,2])
```

```
## [1] "double"
```

You can also look in the "environment" tab, press the blue arrow next to college.data and it will drop down showing the column names with their types and first few values. For college, all columns except the first are numeric. The first column is a factor column, which means it's categorical. To get a better sense of the data, let's look at it

```
View(college.data)
```

To grab a column from a dataframe in R, you have 3 popular options:

```
df$column.name
df[,column.number]
df[,"column.name"]
```

This will be useful so we can separate our outcome column from the feature columns. Let's try! So that we aren't overwhelmed by output, we will also use the function `head()`, which prints only the first few entries

```r
head(college.data$PhD)
```

```
## [1] 70 29 53 92 76 67
```

```r
head(college.data[,13])
```

```
## [1] 70 29 53 92 76 67
```

```r
head(college.data[,"PhD"])
```

```
## [1] 70 29 53 92 76 67
```

Time to Practice!

Exercise 1: Basic Operations and Functions

Create a new vector: Create a vector `my.vector` containing any five numbers. Print the vector. Basic calculations: Find the sum, product, and average of the numbers in `my.vector`.

Use a built-in function: Use the `length()` function to find the length of `my.vector`.

Exercise 2: Vector Manipulation

Create and modify a vector: Create a numeric vector `numbers` from 1 to 20. Then, extract and print the first 5 elements.

Logical indexing: From `numbers`, create a new vector `even.numbers` that contains only the even numbers. Print `even.numbers`.

Vector arithmetic: Create a new vector that is the square of each element in `numbers`.

Exercise 3: Matrices

Create a matrix: Convert `numbers` into a 4×5 matrix `matrix.1`. Print `matrix.1`.

Matrix transposition: Print the transpose of `matrix.1`.

Matrix indexing: Extract and print the element in the 2nd row and 3rd column of `matrix.1`.

Exercise 4: Logical Statements

Simple if-else: Write an if-else statement that prints "Big" if the average of `numbers` is greater than 10, and "Small" otherwise.

Nested if-else: Modify the above to include a check if the average is exactly 10, printing "Exactly 10".

Exercise 5: Random Numbers

Generate random numbers: Generate a vector of 5 random numbers drawn from a normal distribution with mean 0 and standard deviation 1. Print the vector.

Reproducibility: Set a seed of your choice and generate the same vector of random numbers as above.

Exercise 6: Data Frames

Explore `college.data`: Print the first 6 rows of `college.data`.

Column operations: Calculate the mean of the PhD column in `college.data`.

Subsetting: Create a new data frame small.college that only includes colleges with less than 5000 students (use `college.data$Enroll` for enrollment numbers).