Assignment 2: Coding Basics

Molly Bruce

OVERVIEW

This exercise accompanies the lessons in Environmental Data Analytics on coding basics.

Directions

1. Change "Student Name" on line 3 (above) with your name.
2. Work through the steps, **creating code and output** that fulfill each instruction.
3. Be sure to **answer the questions** in this assignment document.
4. When you have completed the assignment, **Knit** the text and code into a single PDF file.
5. After Knitting, submit the completed exercise (PDF file) to the dropbox in Sakai. Add your first and last name into the file name (e.g., "FirstLast_A02_CodingBasics.Rmd") prior to submission.

Basics Day 1

1. Generate a sequence of numbers from one to 100, increasing by fours. Assign this sequence a name.

2. Compute the mean and median of this sequence.

3. Ask R to determine whether the mean is greater than the median.

4. Insert comments in your code to describe what you are doing.

```r
#1. The sequence command generates a list of numbers beginning at the first number in the parenthesis,

S = seq(1, 100, 4)

#2. These two functions are fairly self-explanatory--include in the parenthesis the variable you've ass

M = median(S)
A = mean(S)

#3. I chose to use an if-else statement to determine how the mean and median compare. It's important to

if (A > M) {
  print("mean is greater than median")
  } else {
  print("mean is not greater than median")
  }
```

```
## [1] "mean is not greater than median"
```

Basics Day 2

5. Create a series of vectors, each with four components, consisting of (a) names of students, (b) test scores out of a total 100 points, and (c) whether or not they have passed the test (TRUE or FALSE) with a passing grade of 50.

6. Label each vector with a comment on what type of vector it is.
7. Combine each of the vectors into a data frame. Assign the data frame an informative name.
8. Label the columns of your data frame with informative titles.

```
#5. I have assumed that, by telling us to create these vectors, you'd also like us to populate them wit

Name <- c("Aaron", "Beth", "Charlie", "Debbie")
Grade <- c(84, 79, 48, 91)
PassFail <- c(TRUE, TRUE, FALSE, TRUE)

#6. Though the important function is the class() function, I've also included descriptive print stateme

print("The vector titled Name is vector type: ")
```

```
## [1] "The vector titled Name is vector type: "
```

```
  class(Name)
```

```
## [1] "character"
```

```
print("The vector titled Grade is vector type: ")
```

```
## [1] "The vector titled Grade is vector type: "
```

```
  class(Grade)
```

```
## [1] "numeric"
```

```
print("The vector titled PassFail is vector type: ")
```

```
## [1] "The vector titled PassFail is vector type: "
```

```
  class(PassFail)
```

```
## [1] "logical"
```

```
#7. I used the cbind function to assign the 3 individual vectors to a signle dataframe titled StudentPe

StudentPerformanceDF <- cbind(Name,Grade,PassFail)
print(StudentPerformanceDF)
```

```
##      Name      Grade PassFail
## [1,] "Aaron"   "84"  "TRUE"
## [2,] "Beth"    "79"  "TRUE"
## [3,] "Charlie" "48"  "FALSE"
## [4,] "Debbie"  "91"  "TRUE"
```

```
#8.Though I already gave my vectors fairly logical and informative titles at #5 and these titles became

colnames(StudentPerformanceDF)
```

```
## [1] "Name"     "Grade"    "PassFail"
```

```
StudentPerformanceDF_titled <- data.frame("StudentName"=Name,"TestScore"=Grade,"StudentPassed"=PassFail
print(StudentPerformanceDF_titled)
```

```
##   StudentName TestScore StudentPassed
## 1       Aaron        84          TRUE
## 2        Beth        79          TRUE
## 3     Charlie        48         FALSE
## 4      Debbie        91           TRUE
```

9. QUESTION: How is this data frame different from a matrix?

   Answer: A dataframe can have data of different types, while a matrix must have uniform datatypes. So, for instance, we have a character field, a numeric field, and a logical field in our dataframe. However, a matrix could only have one of these–perhaps only numeric values.

```
#10. Below, I first create a simply in/else function (hashed out). As written, this function would only

      #BadPassFailFunction = if (Grade >= 50) {
      #  print("Passed")
      #  } else {
      #  print("Failed")
      #  }

      # GoodPassFailFunction = ifelse(Grade >=50, "Passed", "Failed")

PassFailFunction <- function(x){
  ifelse (x >= 50, "Passed", "Failed")}

#11. I incorporate my Recipe into my Meal, using Grades as the input for our Function and outputting a

AppliedPassFailFunction <- PassFailFunction(x = Grade)
```

12. QUESTION: Which option of `if` and `else` vs. `ifelse` worked? Why?

    Answer: Both options "worked", though only the ifelse approach iterated through all 4 entries of the vector and output the results into a new vector. Basically, if/else doesn't play nicely with vectors, while ifelse does.