Assignment 2: Coding Basics

Rachael Stephan

OVERVIEW

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

Directions

1. Rename this file `<FirstLast>_A02_CodingBasics.Rmd` (replacing `<FirstLast>` with your first and last name).
2. Change "Student Name" on line 3 (above) with your name.
3. Work through the steps, **creating code and output** that fulfill each instruction.
4. Be sure to **answer the questions** in this assignment document.
5. When you have completed the assignment, **Knit** the text and code into a single PDF file.
6. After Knitting, submit the completed exercise (PDF file) to Canvas.

Basics, Part 1

1. Generate a sequence of numbers from one to 55, increasing by fives. 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.

```
#1. A sequence of numbers increasing by 5 generated by the function seq.

seqby5 <- seq(1,55,5)
seqby5
```

```
##  [1]  1  6 11 16 21 26 31 36 41 46 51
```

```
#2. The mean and median of seqby5 using the mean and median functions.

seqby5_mean <- mean(seqby5)
seqby5_mean
```

```
## [1] 26
```

```
seqby5_median <- median(seqby5)
seqby5_median
```

```
## [1] 26
```

```
#3. A logical statement to determine whether the mean is greater than the median.

seqby5_greater <- seqby5_mean>seqby5_median
seqby5_greater
```

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

```
#The result is FALSE, meaning the mean is not greater than the mean.
#The mean and median in this instance are equal.
```

Basics, Part 2

5. Create three vectors, each with four components, consisting of (a) student names, (b) test scores, and (c) whether they are on scholarship or not (TRUE or FALSE).
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. 3 vectors are being created, one each for student names, test scores, and scholarship
#designation. Each vector has 4 entries total.

#Student names (My siblings' names :D )
stud_name <- c("Emily", "Rachael", "Jack", "Maia")

#Test scores: I have used the sample function to generate 4 random integers to represent
#test scores valued from 1 to 100. I set replacement to true to allow multiple students to
#have the same score.
test_scores <- sample(1:100, 4, replace = TRUE)

#Scholarship designation: I have used the sample function to randomly decide whether each
#student has a scholarship.
scholarship <- sample(c("TRUE","FALSE"), 4, replace = TRUE)

#6. stud_name is a character vector. test_scores is an integer (type of numeric) vector.
#It could be converted to another type of numeric class (i.e., double). scholarship is a
#logical vector.

#7. & 8. Using the data.frame command to create a dataframe from past vectors. I have
#named the columns directly in the data.frame function.

stud_info <- data.frame("Name" = stud_name, "Test_Score" = test_scores,
                        "Scholarship" = scholarship)
stud_info
```

```
##      Name Test_Score Scholarship
## 1   Emily         76        TRUE
## 2 Rachael         19       FALSE
## 3    Jack          5        TRUE
## 4    Maia         22        TRUE
```

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

   Answer: A dataframe varies from a matrix by the type of data they contain. In a matrix, all of the elements in every column and row are of the same data type (e.g., character). In a dataframe, the columns of the 2d structure can vary in the data type they contain (i.e., one column could be character and another integer).

10. Create a function with one input. In this function, use `if...else` to evaluate the value of the input: if it is greater than 50, print the word "Pass"; otherwise print the word "Fail".

```
#10. This is a function to determine whether a number is a passing (>50) or failing
#(<= 50) using the if... else function.

pass_fail1 <- function(grade){
  if (grade > 50) {
    result <- "Pass"
  } else {
    result <- "Fail"
  }
  return(result)
}
```

11. Create a second function that does the exact same thing as the previous one but uses `ifelse()` instead if `if...else`.

```
#11. This is a function that will accomplish the same as the previous function, but is
#accomplished use the ifelse() function.

pass_fail2 <- function(grade){
  result <- ifelse(grade >50, "Pass", "Fail")
  return(result)
}
```

12. Run both functions using the value 52.5 as the input

```
#12.a Running pass/fail function 1 with 52.2 as input.
grade_result1 <- pass_fail1(52.5)
grade_result1
```

```
## [1] "Pass"
```

```
#12.b Running pass/fail function 2 with 52.2 as input.
grade_result2 <- pass_fail2(52.5)
grade_result2
```

```
## [1] "Pass"
```

13. Run both functions using the **vector** of student test scores you created as the input. (Only one will work properly...)

```r
#13a. Run the first function with the vector of test scores
#stud_results1 <- pass_fail1(stud_info$Test_Score)

#13b. Run the second function with the vector of test scores
stud_results2 <- pass_fail2(stud_info$Test_Score)
stud_results2
```

```
## [1] "Pass" "Fail" "Fail" "Fail"
```

14. QUESTION: Which option of `if...else` vs. `ifelse` worked? Why? (Hint: search the web for "R vectorization")

    Answer: A vectorized function in R will act on all elements of a vector without needing for a loop to manually called/coded. This is exemplified in pass_fail2, where the logical test is performed on each element. The if... else statement is not vectorized. This means it is not equipped to deal with a vector unless the loop has taken this into account. I have created a new function, pass_fail3, below to implement this change.

```r
pass_fail3 <- function(grade){
  #create empty character vector
  result <- character()

  #loop to test whether each element is pass/fail and append to result vector
  for (i in 1:length(grade)) {

    if (grade[i] > 50) {
      result <- append(result,"Pass")
    } else {
      result <- append(result,"Fail")
    }
  }

  return(result)
}

stud_results3 <- pass_fail3(stud_info$Test_Score)
stud_results3
```

```
## [1] "Pass" "Fail" "Fail" "Fail"
```

NOTE Before knitting, you'll need to comment out the call to the function in Q13 that does not work. (A document can't knit if the code it contains causes an error!)