Assignment 2: Coding Basics

Fiona Price

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. Use the seq() function to generate a sequence of numbers from 1 to 55.
#The "from" argument is 1, the "to" argument is 55, and the "by" argument is 5.
num_sequence <- seq(1,55,5)
num_sequence
```

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

```
#2. Use the mean() function to find the mean of my sequence. Use the median()
#function to find the median of my sequence.
mean_seq <- mean(num_sequence)
mean_seq
```

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

```
#Mean is 26.
median_seq <- median(num_sequence)
median_seq
```

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

```r
#Median is also 26.

#3. Use <, >, and == to see if the relationship between the mean and median.
#These will return "TRUE" or "FALSE" statements.
mean_seq < median_seq
```

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

```r
#False; mean is not less than median.
mean_seq > median_seq
```

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

```r
#False; mean is not greater than median.
mean_seq == median_seq
```

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

```r
#True; mean is equal to median.
```

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.

```r
#5. Create a vector with student names
student_names <- c("Alice", "Maddie", "Malaika", "Grace")
#Create a vector with grades
test_scores <- c(87, 83, 92, 95)
#Create a vector that indicates whether they are on scholarship or not
scholarship <- c(TRUE, FALSE, TRUE, TRUE)

#6. Use class() to identify what type each vector is
class(student_names) #Character
```

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

```r
class(test_scores) #Numeric
```

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

```
class(scholarship) #Logical
```

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

```
#7. Combine each vector into a data frame using data.frame(x,y,z)
student_info <- data.frame(student_names, test_scores, scholarship)

#8. Label the columns of my data frame with informative titles using names().
names(student_info) <- c("Student Names","Test Scores","Scholarship")

#Check my data frame
student_info
```

```
##   Student Names Test Scores Scholarship
## 1         Alice          87        TRUE
## 2        Maddie          83       FALSE
## 3       Malaika          92        TRUE
## 4         Grace          95        TRUE
```

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

   Answer: A data frame contains different classes of data, while a matrix only includes one. This data frame includes three different classes (character, numeric, and logical).

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".
11. Create a second function that does the exact same thing as the previous one but uses `ifelse()` instead if `if...else`.
12. Run both functions using the value 52.5 as the input
13. Run both functions using the **vector** of student test scores you created as the input. (Only one will work properly...)

```
#10. Create a function using if...else. First, create a function with one input.
over50 <- function(x){
  if (x > 50){
    print ("Pass")
  }
  else{
    print("Fail")
  }
}


#11. Create a second function that does the exact same thing as the previous one
#but uses `ifelse()` instead if `if`...`else `.
over50_rerun <- function(x){
  ifelse(x > 50, "Pass", "Fail")
}

#12a. Run the first function with the value 52.5.
over50(52.5)
```

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

```
#The function passes.

#12b. Run the second function with the value 52.5
over50_rerun(52.5)
```

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

```
#The function passes.

#13a. Run the first function with the vector of test scores.
#First, set my.function to be equal to test_scores.
#over50(test_scores)
#This does not work. It returns "the condition has length > 1.

#13b. Run the second function with the vector of test scores.
over50_rerun(test_scores)
```

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

```
#Each test score passes.
```

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

    Answer: The `ifelse` option worked, while the `if`…`else` failed. This is because `if`…`else` only operates on one value, while `ifelse` on the whole vector. In the case of test_scores there are four values, so we need to use `ifelse` . Vectorization is when an operation is performed on whole vectors rather than on individuals values. `ifelse` is therefore the vectorized version of `if`…'else.

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!)