Intro to R: Week 3

Topics Covered: IF statements and FOR loops

Task 1: Explore IF statements
You have seen the function *ifelse(test, yes, no)* before. Today, we will look at another structure for IF statements that provide more versatility.

Step 1.1 Create a variable a and set it equal to 3. Write an IF statement that prints "a is smaller than 10" in your command window, only if a is indeed smaller than 10.

- Note: the syntax of an IF statement is as follows:
  if (logical test) {
  statements
  }

```
a <- 3

if (a < 10){  # logical test
  print("a is smaller than 10.")  # statement
  }
```

Step 1.2 Add an *else* condition that prints "a is not smaller than 10" in your command window when a is indeed not smaller than 10. Run your statement without changing the value of a, then set a equal to 15 and try again.

- Note: the syntax of an IF…ELSE statement is as follows:
  if (logical test){
  statements
  } else {
  statements
  }

```
if (a < 10) {  # logical test
  print("a is smaller than 10.")  # statement
  } else {
    print("a is not smaller than 10.")
  }
```

Step 1.3 Create a variable b that is equal to 5. Modify your IF…ELSE statement such that when a AND b are smaller than 10, it prints "both a and b are smaller than 10.", when a OR b (but not both) is smaller than 10, it prints "one of a and b is smaller than 10.", and when none are smaller than 10, it prints "both a and b are not smaller than 10."

```
b <- 5

if (a < 10 & b < 10) {
  print("a and b are smaller than 10.")  # statement
  } else {
    if (a < 10 | b < 10) {
      print("one of a and b is smaller than 10.")
      }  else {
        print("both a and b are not smaller than 10.")
        }
  }
```

Task 2: Create a Fibonacci sequence using a FOR loop
A number in the Fibonacci Sequence (1, 1, 2, 3, 5, 8, . . .) is found by adding the two numbers preceding it. In this case, the following number would be 13 (5+8). Create a FOR loop that produces a Fibonacci Sequence of variable length.

Step 2.1 Create a variable called *nFib* where you can specify the length of your sequence. Select a length of 10 for now.

```
nFib <- 10  # Specifies the length of your sequence
```

Step 2.2: Initialize a vector called *Fib* where you will store your sequence. This vector must be the same length as that of your sequence.

- It is more efficient to initialize a vector of the length needed and to fill its positions than to constantly add to an existing vector, making it always longer. An initialized vector can be filled with anything, we chose **NA** in this case.

```
Fib <- rep(NA, nFib)
```

Step 2.3: Using indexing, change position 1 and position 2 of *Fib* so that they are both equal to 1.

```
Fib[1:2] <- 1
```

Step 2.4: Write the formula to calculate the value in position 3. Remember that you must add the two previous positions.

```
Fib[3] <- Fib[2] + Fib[1]

# or:
Fib[3] <- Fib[3-1] + Fib[3-2]
```

Step 2.5: Now, generalize this statement to write a FOR loop that starts at position 3 and ends at the last position of your vector.

- Note: the syntax of a FOR loop is as follows:
  for (variable in sequence){
  statements
  }

```
for (ia in 3:nFib) {  # variable (ia) in sequence (from 3 to nFib)
  Fib[ia] <- Fib[ia-1] + Fib[ia-2]  # statement of what to do
}
```

Step 2.6: To understand what the loop does, print *ia* and *Fib[ia]* to your screen as the loop runs.

```
for (ia in 3:nFib) {  # variable (ia) in sequence (from 3 to nFib)
  Fib[ia] <- Fib[ia-1] + Fib[ia-2]  # statement of what to do

  # shows what the loop does every time
  print(paste("This iteration, ia equals", ia, "and Fib equals", Fib[ia]))

}
```

Step 2.7: Modify the loop you created in Step 1.5 to include an IF statement that will make positions 1 and 2 equal to 1, and perform the calculation otherwise. This means that your FOR loop should now start at position 1 and that the command changing positions 1 and 2 can be commented out.

```
for (ia in 1:nFib) {  # variable (ia) in sequence (from 3 to nFib)
  if (ia==1 | ia==2) {
    Fib[ia] <- 1
    } else {
      Fib[ia] <- Fib[ia-1] + Fib[ia-2]  # statement of what to do
    }
}
```

- Note: whenever you code loops for your own purposes, always start small and check every step. We are not breaking it down only because this is an introduction class; this is good practice.

Step 2.8 Change nFib to make sure your loop can produce as long of a Fibonacci Sequence as desired.

Task 3: Calculate averages and make plots for certain groups in a data frame
The file *Week3_data.csv* contains randomly generated data for a pretend survey of imaginary islands. For each quadrat on any given island (Kilika, Johto, Oxbay, Sula or Vanutu), the percentage cover of hard corals, soft corals, macroalgae and sponges was recorded. You are tasked with finding an average cover of each category for each island. You must also generate a histogram for each island.

Step 3.1 Load *Week3_data.csv* into a variable called *data*. Remember the headers.

```
data <- read.csv("Week3_data.csv", header=TRUE)

head(data)  # confirms that you loaded the right data
```

Step 3.2 Take the time to think about how many times you will need to repeat your FOR loop and what will determine that. Because you are calculating averages for each island, it makes sense that the process in our FOR loop will need to be repeated for each island. Use the function *unique()* to identify the island names into a variable called *IslandNames*.

```
IslandNames <- unique(data$Island)

# shows the number of islands, i.e. how many times we need to repeat the FOR loop
nIsland <- length(IslandNames)
```

Step 3.3 Create a data frame where you will store all of your results. You need as many rows as there are islands, and as many columns as there are categories for the cover. Label your rows and columns in a way that makes sense.

```
cover <- c("Hard corals", "Soft corals", "Macroalgae", "Sponges")

# creates a data frame for your results, with rows labelled by island
results <- data.frame(rep(NA, 5),rep(NA, 5),rep(NA, 5),rep(NA, 5), row.names=IslandNames)

names(results) <- cover  # assigns column names based on the cover types
```

Step 3.4 Set up the structure of your FOR loop.

```
for (ib in 1:nIsland){

  }
```

Step 3.5 Now, take the time to work only on the core of your loop by working with your first island (Kilika). This is as though we are working with *ib* being equal to 1, we will replace things later. If you were to open Excel to do this, the first step would be to look for rows where you see *Kilika*. You must find a way to tell this to your code.

```
ib <- 1  # we will remove this later

# stores the row numbers where Kilika appears in I_island
I_island <- which(data$Island=="Kilika")

# note that the two options produce the same result
I_island <- which(data$Island==IslandNames[ib])
```

Step 3.6 Find the average cover of hard corals for *Kilika*, then for all cover types.

```
mean(data$Pct.hard.corals[I_island])

meanCover <- colMeans(data[I_island, 3:6])  # calculates the mean of each column
```

Step 3.7 Assign your results to the data frame *results*.

```
results[ib, ] <- meanCover
```

Step 3.8 Now, combine all the steps in your FOR loop. Test the inside for *ib* equals 1 and run it for all islands once it works.

```
for (ib in 1:nIsland){
  I_island <- which(data$Island == IslandNames[ib])
  meanCover <- colMeans(data[I_island, 3:6])
  results[ib, ] <- meanCover
  }
```

Step 3.9 Now that you have a working FOR loop, it is easy to add extra commands. As long as you are looking to do something to each island. Add a pie chart for the cover of each island.

```
for (ib in 1:nIsland){
  I_island <- which(data$Island == IslandNames[ib])
  meanCover <- colMeans(data[I_island, 3:6])
  results[ib, ] <- meanCover
  pie(meanCover, labels = cover, main = IslandNames[ib] )
  }
```

Task 4: Create a nested FOR loop
For this exercise, create a nested FOR loop that will create a vector with one row for each day of the year 2015. The first column must tell you the month and the second the day of the month. This way, if you type date[182 ,], you will know what date corresponds to the 182th day of the year.

Step 4.1 Create a vector *months* with all months of the year and a vector *ndays* with how many days there are in each month.

```
months <- month.abb  # see what month.abb does!

ndays <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
```

Step 4.2 Initialize a data frame *date* that has dimensions 365 by 2 where you will store your results. Also initialize a variable r that will help you keep track of which row you are filling in. Set r equal to 1.

```
date <- data.frame(rep(NA, 365), rep(NA, 365))

r <- 1
```

Step 4.3 Write your FOR loop and add 1 to r every time so that you know which row you are filling.

```
for (ia in 1:12){
  for (ib in 1:ndays[ia]){
    date[r,1] <- months[ia]
    date[r,2] <- ib
    r <- r + 1
    }
  }
```

Task 5: Run a handy FOR loop
For this exercise, make sure that all the monthly data is in a subfolder of Week3 called *Task5*.

Step 5.1 Run the following FOR loops to load the data, generate a plot, and save it to your folder without you having to do anything. Make sure you change the code for the working directory to the one on your computer.

```
setwd("/Users/Cadmium/Dropbox/IntroR/Week3/Task5")  # sets the directory
fname <- list.files()  # stores the file names in the directory

#creates file names for the plot (subsitutes .csv to .png)
plotnames <- gsub(".csv", ".png", fname)


for (ic in 1:length(fname)){
  data <- read.csv(fname[ic])
    size <- dim(data)
    p <- plot(1:size[1], data$Chlorophyll..ug.L., pch=".",
         xlab="Data point", ylab="Chlorophyll (ug/L)",)
    png(filename=plotnames[ic])
    p <- plot(1:size[1], data$Chlorophyll..ug.L., pch=".",
         xlab="Data point", ylab="Chlorophyll (ug/L)",)
}
```