In the first course, we explored the basics of programming in R. We helped you calculate your grades, pick a university, pick a major & find employment using the four most common data structures. We've been executing our code, so far, from top to bottom. Each line of code is run sequentially. However, there'll likely be scenarios where you don't want your code to run sequentially. Perhaps, you want your model to execute a certain piece of code *if* a stock price goes up. Other times, you want to run the same line of code *multiple* times to simiulate the results of a sports match.

The controls of how our code is executed are called **control structures**. Control structures allow you to direct the flow of code execution. You can re-run specific lines of code multiple times or run lines of code *only* if a condition is met. In this mission, we'll use control structures to build a simple algorithm that tells us who won a soccer/football match. Let's say that we're watching a match that decides which teams makes the playoffs. Let's look at this scenario through a flow chart:

When two teams play each other, there is one winner. Therefore, there are only two possible outcomes. Given that the winner will make the playoffs, we'll write a program that will print out the winner between team\_A and team\_B. To write this conditional, we'll use something called an if statement. An if statement, tells the interpreter to run a line of code *if* a condition returns TRUE:

condition should be an expression that evaluates to TRUE or FALSE. If the expression returns TRUE, then execute all code between { }. If FALSE, then no code will be executed. Knowing this, let's look at an example of an if statement thats prints the team name that won.



team\_A <- 3 # Number of goals scored by Team A

team\_B <- 1 # Number of goals scored by Team B

​

if (team\_A > team\_B){

print ("Team A wins")

}

Running this code, will return the following results:



[1] "Team A wins"

Now that we've printed the result of team\_A vs. team\_B, let's use an if statement to print out the winner of the red\_dragons vs. deep\_sea\_sailors!

instructions

* Assign the results of the following game to separate variables:
  + red\_dragons: 5,
  + deep\_sea\_sailors: 4
* Use an if statement to check if red\_dragons scored more goals than deep\_sea\_sailors.
  + If red\_dragons scored more goals than deep\_sea\_sailors, print "red dragons".
* Store the if statement expression in winner.

In the previous exercise, we printed the name of the team that'll make the playoffs based on our expression. Let's look at a code block that's similar to the previous exercise:



team\_A <- 3 # Number of goals scored by Team A

team\_B <- 1 # Number of goals scored by Team B

​

if (team\_A > team\_B){

print ("Team A will make the playoffs")

}

But what if team\_A had 1 goal and team\_B had 3 goals. Our conditional would evaluate to FALSE:



team\_A <- 1 # Number of goals scored by Team A

team\_B <- 3 # Number of goals scored by Team B

​

if (team\_A > team\_B){

print ("Team A will make the playoffs")

}

If we ran this code, the interpreter won't return anything, because team\_A > team\_B will evaluate to FALSE. So far, we've only coded out one branch of our flow chart:

However, we'd ideally like our program to print "Team B will make the playoffs" if the expression evaluates to FALSE. We'd like our program to be able to handle both cases:

To do this, we'll need to add an else statement. An else statement tells the interpreter to run specific lines of code if our comparison operator evaluates to FALSE:

If our conditional expression (team\_A > team\_B) returns FALSE, well need to add another print statement. Returning to our example that returns no values, let's add an else statement:



team\_A <- 1 # Number of goals scored by Team A

team\_B <- 3 # Number of goals scored by Team B

​

if (team\_A > team\_B){

print ("Team A will make the playoffs")

} else {

print ("Team B will make the playoffs")

}

This will return:



[1] "Team B will make the playoffs"

Now, let's add an else statement to the red dragons vs. deep sea sailors match!

instructions

* Add an else statement to our if statement.
* Within the else statement, print "deep sea sailors".
* For this problem, we've changed the goals scored: red dragons: 2 deep sea sailors: 3
* Store this entire expression in result.

Now that we've returned the results of one match. What if we wanted to find the results of *multiple* matches? Let's say, we had a list of vectors containing the results of our match: matches <- list(c(2,1),c(5,2),c(6,3)). Assuming that the team\_A's goals fall in the first index of the vector and team\_A 's opponent falls on the second, if we were to find the results using if-else statements, this would look like this:



if (matches[[1]][1] > matches[[1]][2]){

print ("Win")

} else { print ("Loss") }

​

if (matches[[2]][1] > matches[[2]][2]){

print ("Win")

} else { print ("Loss") }

​

if (matches[[3]][1] > matches[[3]][2]){

print ("Win")

} else { print ("Loss") }

And this would print:



[1] "Win"

[1] "Win"

[1] "Win"

Keep in mind, we're using [[]] when indexing, since we want to return the single value, not the value with the list object(see list indexing in the previous course). [] will return the value within the list object. Writing the results using if-else statements *can* work, but if our matches list contains 100 matches, this would be *extremely* cumbersome to write out each statement.

Instead, we can perform the same action using a **for loop**. A for loop repeats a chunk of code, multiple times for each *element* within an object:

In this diagram, for each value in the sequence, the loop will execute the code block. However, if there are no more values left in the sequence, this will return FALSE and exit the loop.

Let's break down what's going on here.

* **sequence**: This is a set of objects. For example, this could be a vector of numbers c(1,2,3,4,5).
* **value**: This is an iterator variable you use to refer to each value in the sequence. See variables naming conventions in the first course for valid variable names.
* **code block**: This is the expression that's evaluated.

Let's look at a concrete example. If we were to write a loop for the following code:



teams <- c("red dragons","deep sea sailors")

​

for (value in teams){

print(value)

}

Since team has two values, the loop will run twice. Let's look at the loop:

Once the loop displays the result from the first iteration, the loop will look at the next value in the position. As a result, it'll go through another iteration. Since there aren't any more values in the sequence, the loop will exit after "deep sea sailors". In aggregate, the final result will look like this:



[1] "red dragons"

[1] "deep sea sailors"

The result of each loop will have it's own line of code. Let's loop through all our match results and print the scores of each match!

instructions

* We have a list of vectors called matches with the scores for each match.
* Write a for loop and print the scores for each match.
* Store this expression in all\_matches.

Now that we've written out our loop, when we figure out the winner, we'll want to store each result of our loop. To store these values, we could store it back into a data structure. In this context, we'll store our values in a vector, since we're dealing with the same data types.

In the previous course, we learned how to combine vectors using the c() function. We'll use the same method to store the results of our for loop. Let's take the following for loop:



for (match in matches){

print(match)

}

And let's say we wanted to get the total goals scored in a game and store them in the vector:



matches <- list(c(2,1),c(5,2),c(6,3))

​

for (match in matches){

sum(match)

}

Now, if we want to save the total goals for each match, we'll can initialize a new vector and then append each additional calculation onto that vector:



matches <- list(c(2,1),c(5,2),c(6,3))

​

total\_goals <- c()

for (match in matches){

total\_goals <- c(total\_goals, sum(match))

}

The resulting vector would be total\_goals <- c(3,7,9). Let's create a new vector called total\_goals using our for loop.

instructions

* Using the for loop from the previous exercise, find the total goals scored for each game and store this in a vector called total\_goals.

Now that we've learned if-else & for loops, we can use these control structures to tell us whether red dragons won or lost the match. To combine two control structures, we'll place one control structure in between the { } of the other. Let's take the following match results of team\_A:



matches <- list(c(2,1),c(5,2),c(6,3))

And then, let's loop through it:



for (match in matches){

}

This time, rather than print our results, let's add an if-else statement into the expression. In our scenario, we want our program to print whether team\_A won or lost the game. Assuming team\_A's goals is the first index of each pair of values and the opponents is the second index, we'll need to use a comparison operator to compare the values. After we make the comparison, if team\_A's score is higher, we'll print "Win". If not, we'll print "Lose":



for (match in matches){

if (match[1] > match[2]){

print("Win")

} else {

print ("Lose")

}

}

Running this code displays:



[1] "Win"

[1] "Win"

[1] "Win"

Now, let's store the results in our code by creating a vector and appending to the vector for each loop:



result <- c()

​

for (match in matches){

if (match[1] > match[2]){

print("Win")

result <- c(result, "Win")

} else {

print ("Lose")

result <- c(result, "Lose")

}

}

Printing result will show:



[1] "Win" "Win" "Win"

When indexing into the iterable variable match, you can use either [] or [[]] since the iterable is a vector, not a list. Now, let's print the results for all the matches of red dragons!

instructions

* Write a for loop and add an if-else statement using the matches variable that displays whether red dragons won or lost the match. Use "Win" and "Lose".
* Store the result in a vector called get\_results.
* Once finished writing the loop, store get\_results into results like this: results <- get\_results.

Now that we've added an if-else statement, what if we wanted our for loop to *stop* running based on a certain condition? In our case, what if we wanted our for loop to *stop* running as soon as team\_A won a game? To stop the for loop, we can use a break statement to stop the loop. Let's return to our previous example:



for (match in matches){

if (match[1] > match[2]){

print("Win")

break

} else {

print("Lose")

}

}

Running this code displays:



[1] "Win"

Since team\_A won their first match, the loop will break after printing their "Win". Let's add a break statement to our current for loop!

instructions

* For red dragons, break the loop when their first loss occurs.

In the previous exercise, we used a for loop to repeat a chunk of code that gave us the result of the match. Now that we've returned the results of each match, what if we wanted to count the number of wins a team has to determine if they make the playoffs? One method of returning the results for the first four games, is to use a **while** loop.

A while loop is a close cousin of the for loop. However, a while loop will check a logical condition, and keep running the loop as long as the condition is true. Here's what the syntax of a while loop looks like:



while(condition){

expression

}

In flow-chart form:

If the condition in the while loop is *always* true, the while loop will be an infinite loop. When writing a while loop, we want to ensure, that at some point, the condition will be false, so the loop can stop running. Let's take a team that's starting the season with zero wins. They'll need to win 10 matches to make the playoffs. We can write a while loop to tell us whether the team makes the playoffs:



wins <- 0

​

while (wins < 10){

print ("Does not make playoffs")

wins <- wins + 1

}

Our loop will stop running when wins hits 10. Notice, that we continuously add 1 to the win total, so eventually, the win < 10 condition will return FALSE. As a result, the loop exits. Let's write our first while loop, counting red dragon wins.

instructions

* Note: we will not be using the existing matches list in this exercise to write the while loop.
* Create a variable wins and initialize it to 0.
* Write a while loop for the red dragons team. If the red dragons have 15 wins, they'll make the playoffs. Use wins as the counter.
* If they do not have 15 wins yet, within the while loop, print out "does not make playoffs" and append"does not make playoffs" to the vector playoffs.

Now that we've printed the status of the team when they *do not* have enough wins. Let's add a feature to our program that indicates when they *do* make the playoffs. To do this, we'll need to add an if-else statement into our while loop. Adding an if-else statement into our while loop is the same as adding it to our for loop. In our previous example, where 10 wins allowed us to make the playoffs, let's add the if-else conditional:



wins <- 0

playoffs <- c()

​

while (wins <= 10){

if (wins < 10){

print("does not make playoffs")

playoffs <- c(playoffs, "does not make playoffs")

} else {

print ("makes playoffs")

playoffs <- c(playoffs, "makes playoffs")

}

wins <- wins + 1

}

This will display:



[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "makes playoffs"

We've stored this all in playoffs. Now, let's add an if-else statement to our current while loop!

instructions

* Create a variable wins and initialize it to 0.
* Add an if-else statement into our original while loop to print "makes playoffs" if wins is equal to 15.
* Within the while loop condition, change wins < 15 to wins <= 15 to include 15.

Now, let's say the maximum number of wins a team can have in a season is 15. To make the playoffs, we'll still need 10 wins. We want to write a program that tells us if we made the playoffs or not. To do this, we can use a while loop and then insert a break statement when wins hits 10:



while (wins <= 15){

if (wins < 10){

print("does not make playoffs")

playoffs <- c(playoffs, "does not make playoffs")

} else {

print ("makes playoffs")

playoffs <- c(playoffs, "makes playoffs")

break

}

wins <- wins + 1

}

Running this code will display:



[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "does not make playoffs"

[1] "makes playoffs"

Let's throw a break statement into our win counter!

instructions

* Write a while loop for the red dragons team. If the red dragons have 20 wins, they'll make the playoffs. Use wins as the counter.
* If they do not have 20 wins yet, within the while loop, print out "does not make playoffs". Append "does not make playoffs" to the vector get\_result.
* If they do have 20 wins, print out "makes playoffs". Append "makes playoffs" to get\_result.
* Insert a break statement that exits the loop when wins hit 20.
* After completing the while loop, store get\_result in result, like this result <-get\_result.
* Congratulations! You've learned the three main control structures used in R and you can now write your own loops and add your own if-else statements. In the next mission, we'll use control structures to write our own functions!