Worksheet 5: Loops

Lab worksheet: For loops

This worksheet builds on your past work. Make sure you have defined in your environment `fibSeq` and `fibMat`.

Loops are used in programming to repeat a task multiple times, often with a slight variation each time. For example, say you want to print out statements about how the Fibonacci numbers relate to flowers. Without using a loop, you might make a script snippet

```
print(paste("a flower with", FibDF$number[1], "petal is", FibDF$flower[1]))
print(paste("a flower with", FibDF$number[2], "petal is", FibDF$flower[2]))
print(paste("a flower with", FibDF$number[3], "petal is", FibDF$flower[3]))
print(paste("a flower with", FibDF$number[4], "petal is", FibDF$flower[4]))
print(paste("a flower with", FibDF$number[5], "petal is", FibDF$flower[5]))
print(paste("a flower with", FibDF$number[6], "petal is", FibDF$flower[6]))
print(paste("a flower with", FibDF$number[7], "petal is", FibDF$flower[7]))
```

However, that is very clunky, and if you wanted to change the output slightly (capitalize the A, put a period at the end, for example), you would have to change each line. In my experience, this is a recipe for copy and paste errors that are a pain to debug.

Instead, we can use a for loop. For loops have an iterator (`i`) which increases with each repetition. To set up a for loop, you start the iterator at some value (usually 0), and set it iterate through a set of other values. For instance, consider the script snippet

```
for(i in 1:7) {
  print(paste("a flower with", FibDF$number[i], "petal is", FibDF$flower[i]))
}
```

The for loop statement has `i` iterate through the numbers 1 through 7 as the command in the curly brackets is carried out. In this case, the sentence relating each Fibonacci number to a flower is printed. Note as `i` increases, the `[i]` specified in the loop, runs through the rows of the data frame.

1. To practice, make a script with for loop that computes the square of the first seven Fibonacci numbers (you stored in `fibSeq`). Save those values into a new vector (defined before the loop) `fibSeqSq`.

2. Now, if necessary, modify that for loop so that it `i` takes values between 1 and the length of `fibSeq`. This makes the code more flexible so that it will work on `fibSeq` vectors of different lengths.

3. Set `fibSeq` to the first 10 Fibonacci numbers and run your loop again.

4. Now, before the for loop, define an empty vector called `fibSquares` to be the same length as `fibSeq`. As you compute the squares, save the values into `fibSquares`.

Nested for loops

Loops can be nested inside of loops. This is particularly useful to iterate through all entries in a matrix or multi-dimensional array. For example, add this to your script and execute it

```
for(i in 1:dim(fibMat)[1]) {
   for(j in 1:dim(fibMat)[2]) {
      print(paste("fibMat element", i, ",", j, "is", fibMat[i,j]))
   }
}
```

Because the command inside the loops calls `fibMat[i,j]`, the outer `i` loop iterates through the rows, while the inner `j` loop iterates through the columns.

This is like reading a book: the outer for loop iterates over pages, and the inner for loop iterates over words on a page. So when you're on page `i`, you read every word as `j` increases.

Let's try it out on the board together so that it's clear (if Rori hasn't done this yet, ask her to!).

5. Modify the script snippet above to print the elements of `fibMat` in sequence order.

6. Create a 10x10 matrix `FibMults` with the first ten multiples of the first ten Fibonacci numbers. Each row will represent a Fibonacci number and each column a multiple. For example the top left 4x4 of your matrix will have values

1	2	3	4
1	2	3	4
2	4	6	8
3	6	9	12

   a. Using an indefinite index, access the Fibonacci numbers multiplied by 3.

   b. Using an indefinite index, access multiples of the fourth Fibonacci number.