Repeating code

● We’ve talked about how functions are used to minimize repetitive code

● If we have a block of code that we want to be able to bring up and use any time, and with different data, we can define a function and put that block of code in it

● Today we’re going to look at another way we can deal with code that needs to be repeated
Repeating code

● Often there will be times where writing a function isn’t going to be enough to get rid of your need to repeat code.

● For example:
  ○ What if you wanted to call a function \textbf{1000 times}?
  ○ You can write a function that calls that function 1000 times, but that is going to require a file that is \textbf{1000 lines long}
  ○ Not very efficient...
Repeating code

● More relevant example for us:
  ○ What if we wanted to do something with each *individual* element of a vector?
  ○ If we had a vector \( v \leftarrow c(4, \ 35, \ 23, \ 12, \ 7, \ 2) \), right now we would have to index every element manually:
    \( v[1], v[2], v[3], \ldots \) and so on.
  ○ Not very efficient...

● We want a way to get to all of the values individually without writing out the numbers each time
Loops

- In order to help us with this problem, we’re going to introduce a new concept: **Loops**

- Simply: Loops allow us to repeat a block of code

- Like if-statements, this is another way to change the program flow of R.

- Today we will be looking at one type of loop in R, the **for** loop
for loops

- A **for** loop is a statement that allows us to repeat code a set number of times.

- The number of times it repeats depends on some ordered set of values
  - In our case, vectors!

- For loops take each element in a vector, save it to a variable, and execute the code in the loop
  - It then repeats this process for every element.

- Let's take a closer look
for loop Format

for (element in vector) {
    # loop body
}

This block is considered one for loop

Let’s talk about what these words all mean
for loop Format

```javascript
for (element in vector) {
  # loop body
}
```

for Indicates that this is a for loop statement
for loop Format

```javascript
for (element in vector) {
    # loop body
}
```

element in vector

element is the variable name we are going to give to every element in vector as we repeat the code

We call this iterating over a vector:
“For every element element in the vector vector”
for loop Format

for (element in vector) {
    # loop body
}

# loop body

● These lines of code (which are indented in the for loop), will repeat for every element in the vector.
● We can use the value of the variable element and work with it all the way to the end of the for loop.
Let’s see an example

```r
numbers <- c(23, 5, 47)
for (n in numbers) {
    cat(n)
    cat("\n")
}
```
Let’s see an example

```r
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}
```

**for**

Indicates that this is a for loop statement
Let’s see an example

```r
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}
```

We are going to “iterate” over the vector number. Every time we repeat the loop body, we will change the value of the variable `n` to be the next number in the vector `numbers`.
Let's see an example

```r
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}
cat(n) and cat("\n")
```

This is the loop body. We are using the variable `n` which is the current value from `numbers` that we have iterated to.

**Reminder:**

"\n" is a string with the newline character. It has nothing to do with the variable `n`. 
Running the example

numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}

1st iteration of loop:
Current value of n: 23

R Console output after running loop body:
23
Running the example

```r
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}
```

2nd iteration of loop:
Current value of n: 5

R Console output after running loop body:

```
23
5
```
Running the example

```
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}
```

3rd iteration of loop: Current value of n: 47

R Console output after running loop body:

```
23
5
47
```
Running the example

```r
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}

# program continues
.
# program continues
.

No more values in numbers.
We’re done! We now move on to the statements after the for loop

23
5
47
```
Something to be careful about

```
n <- 900
m <- 1
numbers <- c(23, 5, 47)
for (n in numbers) {
  cat(n)
  cat("\n")
}
p <- n + m
```

- **Do not** use variables that we assigned outside of the for loop as the name for each element inside the for loop.
- You might need it later, but it will still be assigned to the last element of the vector.
Something to be careful about

The last value of \texttt{n} in the loop was 47

The value of \texttt{p} is:
\[ p = n + m = 47 + 1 = 48 \]

\textbf{NOT:}\n\[ 900 + 1 = 901 \]

\textbf{Rule:} use a different variable name in the loop
Looping over a sequence

- We can loop over a sequence of numbers

```r
for (i in 1:5) {
  ...}
```

1:5 gives us a vector equivalent to `c(1, 2, 3, 4, 5)`

**Convention:** Use the variable `i` when looping over a sequence of numbers
Nested loops

- We can put a loop inside a loop

```r
for (i in 1:5) {
    for (j in 1:3) {
        ...
    }
}
```

Here we loop through the sequence 1:5, and for every element of that sequence, we also loop through 1:3.

We will see how this works in RStudio.

Convention: Use the variables `i` and `j` when looping over a sequence of numbers with nested loops.
Examples in RStudio