Introduction to R: **Control Flow**
Session 1, Part D

Nick Graetz[1]

[1] University of Pennsylvania, Population Studies Center

9/4/2020

1. `if/else` statements
2. Assertions
3. `for` loops
4. `break` and `next`
5. `while` loops

if statements can be used to specify that a line of code should only be
executed if a given condition is met:

```
> x <- -5
> if (x < 0) x <- -1 * x
> x
[1] 5
```

```
> x <- 5
> if (x < 0) x <- -1 * x
> x
[1] 5
```

else statements can be used to specify alternate code that should be executed if a given condition is NOT met:

```
> x <- -5
> if (x < 0) y <- -1 * x else y <- 100 * x
> y
[1] 5
```

```
> x <- 5
> if (x < 0) y <- -1 * x else y <- 100 * x
> y
[1] 500
```

if and else can be used alongside curly brackets to specify blocks of code that should be executed if a condition is or is not met:

```
> x <- rnorm(1)
>
> if (x < 0) {
+    y <- -1 * x
+ } else {
+    y <- 100 * x
+ }
>
> x; y
[1] 1.30109
[1] 130.109
```

Note: else must follow on the same line as the close brace for the if block.

Adding in **else if** allows you to specify additional alternatives:

```
> x <- rnorm(1)
>
> if (x < 0) {
+   y <- -1 * x
+ } else if (x < 100) {
+   y <- x * 100
+ } else {
+   y <- x * 5
+ }
>
> x; y
[1] -0.4630561
[1] 0.4630561
```

Note: you can have as many else if's as you want, and you do not necessarily need to follow with an else

If needed, you can also nest if, else, and else if types of statements:

```
> x <- rnorm(1)
>
> if (x < 0) {
+   y <- x * -10
+ } else {
+   if (x < 10) {
+     y <- x * 100
+   } else {
+     y <- x * 10
+   }
+ }
>
> x; y
[1] 0.8235842
[1] 82.35842
```

Any kind of logical statement (i.e., something that returns TRUE or FALSE) can be used in an `if`, `else`, or `else if`, so long as it returns a *single* value.

```
> x <- rnorm(4)
> x
[1] -0.8082732  0.1396544 -0.5649409  0.6087641
```

This is fine:

```
> if (sum(x) > 0) print("the sum of x is greater than 0")
> if (x[1] > 0) print("the first value of x is greater than 0")
```

This is (usually) wrong:

```
> if (x > 0) print("x is greater than 0?")
Warning in if (x > 0) print("x is greater than 0?"): the
condition has length > 1 and only the first element will be
used
```

In addition to relational operators (!=, ==, <, <=, >, >=) and logical operators (&, |), **functions** that return TRUE or FALSE can be used in if, else, and else if statements:

```
> today <- as.character(Sys.time())
> today
[1] "2020-08-30 10:54:33"
> if (grepl("2017", today)) print("The year is 2017")
```

```
> x <- log(rnorm(4))
Warning in log(rnorm(4)): NaNs produced
> x
[1] NaN NaN NaN NaN
> if (sum(is.na(x)) > 0) print("Vector x contains missing valu
[1] "Vector x contains missing values"
```

One particularly useful application of if statements are assertions: testing to ensure that some condition that should be met is actually met, and stopping the code from executing otherwise.

if can be combined with **stop()** to accomplish this:

```
> if (sum(is.na(x)) > 0) stop("There is missingness!")
Error in eval(expr, envir, enclos): There is missingness!
```

Or you can use the **stopifnot()** function:

```
> stopifnot(sum(is.na(x)) == 0)
Error: sum(is.na(x)) == 0 is not TRUE
```

In some cases, it makes sense to provide a message or warning instead of completely halting execution.

```
> if (sum(is.na(x)) > 0) message(paste('x has', sum(is.na(x)),
x has 4 missing values
```

```
> if (sum(is.na(x)) > 0) warning(paste('x has', sum(is.na(x)),
Warning: x has 4 missing values
```

`message()` is also frequently useful apart from `if/else` type statements, just for reporting information the user of some code may want to know.

for loops allow you to repeat a block of code while iterating through a vector of values that (usually) impact that block of code in some way.

```
> for (x in c("red", "orange", "blue", "green", "yellow")) {
+   print(paste("I like the color", x))
+ }
[1] "I like the color red"
[1] "I like the color orange"
[1] "I like the color blue"
[1] "I like the color green"
[1] "I like the color yellow"
```

for loops are generally used to repeat a set of operations for a series of closely related values.

For example:

1. Loading a series of closely related files (e.g., one for each calendar year, country, or age group).

2. Making a series of similar graphs (e.g., one for each location or indicator).

3. Carrying out transformations to a series of data frame columns (e.g., calculating z-scores for all variables).

If you want to store information generated in each iteration of a for loop in R's memory, you'll need to set up a data structure to hold that information.

```
> mat <- matrix(nrow = 3, ncol = 3)
> val <- 4:6
> for (i in 1:3) {
+   mat[, i] <- val * i
+ }
> mat
     [,1] [,2] [,3]
[1,]    4    8   12
[2,]    5   10   15
[3,]    6   12   18
```

Sometimes it is useful to print statements in a loop. This can help with debugging (it makes it obvious where you were when something went awry) and may also be useful if you just want to provide a status update to the user.

```
> for (year in 2010:2013) {
+   file <- paste0(main_dir, "data/wa_data_", year, ".csv")
+   if (!file.exists(file)) {
+     print(paste0("File does not exist for year ", year))
+   }
+ }
[1] "File does not exist for year 2010"
[1] "File does not exist for year 2011"
[1] "File does not exist for year 2012"
[1] "File does not exist for year 2013"
```

You can nest loops! This is useful if you need to iterate over multiple dimensions:

```
> types <- c("data", "income")
> years <- c(1997:2003, 2010:2013)
> for(type in types) {
+   for(year in years) {
+     file <- paste0(main_dir, "data/wa_", type, "_", year, ".csv")
+     if(file.exists(file)) {
+       print(paste(type, year))
+     }
+   }
+ }
```

Sometimes you may want to end a single iteration of a `for` loop early. This is accomplished with the **next** function:

```
> files <- c()
> for (year in 2010:2013) {
+   file <- paste0(main_dir, "data/wa_data_", year, ".csv")
+   if (!file.exists(file)) next
+   files <- c(files, file)
+ }
> files
NULL
```

Similarly, you may want to stop the entire loop (not just a single iteration) early. **break** can accomplish this.

```
> files <- c()
> for (year in 2010:2013) {
+    file <- paste0(main_dir, "data/wa_data_", year, ".csv")
+    if (!file.exists(file)) break
+    files <- c(files, file)
+ }
> files
NULL
```

R also has **while** loops, though these are used less commonly than for loops.
A while loop continues to execute until some pre-specified condition is met.

```
> x <- 0
>
> while (x < 5) {
+     x <- x + abs(rnorm(1))
+     print(x)
+ }
[1] 0.8283707
[1] 2.296895
[1] 2.582284
[1] 2.602422
[1] 3.471336
[1] 3.793506
[1] 5.181087
```