

Control structures - `if...else` and `for` loops

CRI R Workshop

-
- What, and why?
 - `if` and `if...else`
 - `if`
 - `if...else`
 - *Challenge*
 - `for` loops
 - *Challenge*
-

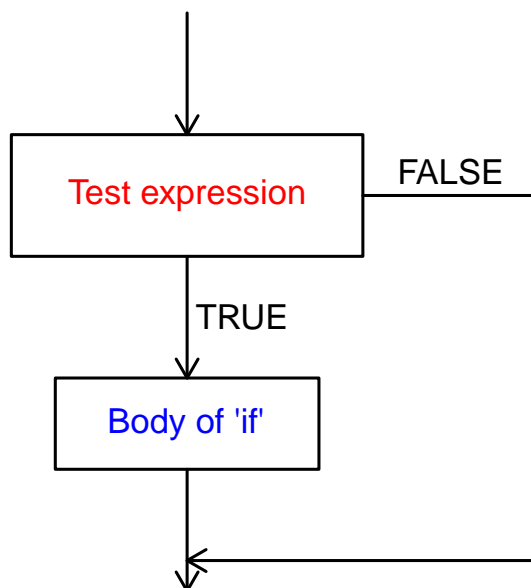
Why, and what?

Control structures allow us to alter the flow of a program, and to make decisions about when or how many times an action occurs. Here we will introduce you to two of them: `if...else` statements and `for` loops.

`if` and `if...else`

`if`

if statements test whether a condition is true, and then execute a piece of code if so. This can be diagrammed as follows:



In R, an if statement looks like the following:

```
if (test condition is true) {  
  run this line of code  
}
```

Note that the conditional test is in parentheses, and the action to be executed is in curly braces { }. It is not necessary to put the body of the if statement on a separate line, but this helps your code to be more readable. The same is true for the indentation - it is there to help you read your code more easily.

As an example, we can write an if statement to act based on whether a number is positive.

```
a <- 4 # Set value for a  
if (a > 0) {  
  print("positive")  
}
```

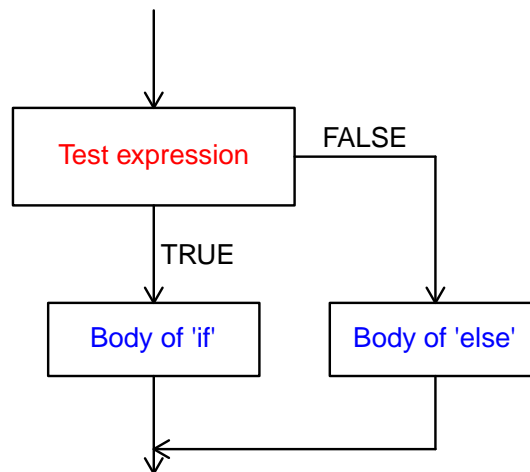
```
## [1] "positive"
```

But what if the number is negative? Try this out for yourself.

With `a` set to a negative value, nothing was printed. Let's change the statement so that "negative" is printed if the condition is not true. Here's where the `else` part comes in.

if...else

An `if...else` statement controls flow in this way:



To accomplish this, the code will look like this:

```
if (test condition is true) {  
  run this line of code  
} else {  
  run this other line of code instead  
}
```

Note that the `else` has to be on the same line as the closing bracket of the `if` statement.

Building onto the earlier example, we can add the `else` part to have the function print “negative” if the value is not above 0. We’ll set the value of `a` to a negative number to demonstrate the flow:

```
a <- -3
if (a > 0) {
  print("positive")
} else {
  print("negative")
}
```

```
## [1] "negative"
```

Great, our statement works! But what if the value of `a` is 0? Try this out for yourself.

In this case, our assessment is incorrect. We can add a second test condition to the statement using `else if`, which effectively nests one `if...else` inside the other.

The R code will be structured like this:

```
if (test condition is true) {
  run this line of code
} else if {
  run this other line of code instead
} else {
  run this line of code when none of the previous conditions are true
}
```

As an example, with `a` now equal to 0, and specifying all expected conditions (with the last `else` as a silly warning):

```
a <- 0
if (a > 0) {
  print("positive")
} else if (a < 0) {
  print("negative")
} else if (a == 0) {
  print("zero")
} else {
  print("Houston, we have a problem.")
}
```

```
## [1] "zero"
```

Try changing the value of `a` and re-running the `if...else` statement to see for yourself how it works.

Is it possible to get to the last `else` statement? How, or why not?

There is another way to write an `if...else` statement, and that is with the `ifelse` function. This function has 3 arguments: the `test` case; `yes`, or what to do if the test case is true; and `no`, or what to do if the test case is false. Let’s use this function to test whether a number is even. We can do that using `%`, which gives the remainder after division.

```
a <- 43
a %% 2
```

```
## [1] 1
```

```
ifelse(test = a %% 2 == 0, yes = "even", no = "odd")
```

```
## [1] "odd"
```

Here, if the remainder after division by two is equal to 0, the test case is true, and “even” is printed. Otherwise, if the test case is false, “odd” is printed. With `a` equal to 43, the remainder was 1, and therefore “odd” was printed. Try a few values of `a` for yourself to see how this works.

Let’s work with the `nutrients` data to demonstrate when this could be helpful. As a reminder, here is the data frame:

```
nutrients
```

```
##   Treatment Replicate Ammonium Nitrate Nitrite
## 1         1         1      8.2     1.7     0.4
## 2         1         2      6.9     3.6     1.5
## 3         1         3     12.1     2.8     0.8
## 4         2         1     10.5     0.4     0.7
## 5         2         2      8.6     2.7     1.2
## 6         2         3      7.8     4.1     0.9
```

The instruments and methods that are used to measure nutrient concentrations often have detection limits, and values below this limit cannot be distinguished from 0. For this reason, one might choose to set these values equal to 0 or to NA. Let’s say the detection limit for nitrite is 0.5 ppm. Rather than manually checking which values are less than or equal to 0.5, and then changing those values using subsetting, we can use an `ifelse` statement within `mutate` to do this for us. We’ll add the “corrected” data as a new variable.

```
nutrients <- nutrients %>%
  mutate(Nitrite_corr=ifelse(Nitrite<=0.5, NA, Nitrite))
```

```
nutrients
```

```
##   Treatment Replicate Ammonium Nitrate Nitrite Nitrite_corr
## 1         1         1      8.2     1.7     0.4           NA
## 2         1         2      6.9     3.6     1.5           1.5
## 3         1         3     12.1     2.8     0.8           0.8
## 4         2         1     10.5     0.4     0.7           0.7
## 5         2         2      8.6     2.7     1.2           1.2
## 6         2         3      7.8     4.1     0.9           0.9
```

Challenge

- Write an if statement that returns a message if any values of `Nitrate` in `nutrients` is less than 2.

...did you get the error message below?

```
if (nutrients$Nitrate < 2) { print("Less than two!") }
```

```
## Warning in if (nutrients$Nitrate < 2) {: the condition has length > 1 and  
## only the first element will be used
```

```
## [1] "Less than two!"
```

What happened here?

The if statement can only accept a single element, so if you try to use a vector with multiple elements, it will only act on the first one. We can use the function `any` to assess whether any elements of a vector are `TRUE` and return only a single `TRUE` or `FALSE` value. Similarly, we can use the function `all` to assess whether all elements of the vector are `TRUE`.

For example:

```
any(nutrients$Nitrate < 2)
```

```
## [1] TRUE
```

Challenge redux

- Try the above challenge again, using the `any` function.

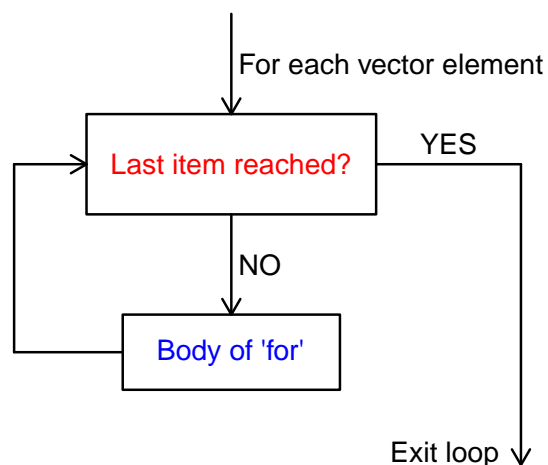
Solution:

```
if (any(nutrients$Nitrate < 2)) {  
  print("At least one value is less than 2.")  
}
```

```
## [1] "At least one value is less than 2."
```

for loops

Another useful control structure that we can use in R is a **for loop**. For loops iterate over values in a vector, and execute code for each value. This can be extremely helpful if you want to perform an action multiple times! For example, you might want to read in multiple files, or do the same thing to multiple data tables. The flow of a for loop looks like this:



The code in R has this general structure:

```
for (n in c(your vector)) {  
  do this thing involving n  
}
```

In this loop, `n` will be set iteratively to each of the values in your vector, and the code in brackets will be executed for each value of `n`. To demonstrate how this works, let's print every number in a vector.

```
for (n in c(1:5)) {  
  print(n)  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5
```

It's not important what variable we put in the place of `n`. We can name this variable `x`, or `trees`, or `fabulousvariable`, or `Constantinople`.

```
for (Constantinople in c(1:5)) {  
  print(Constantinople)  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5
```

This approach can be useful for applying the same action to multiple objects. For example, let's read in multiple data files and assign them to different data frames. Let's start with a data frame specifying the names of the files and the data frame names that we want to give to them.

```
ExperimentFiles <- data.frame(files = c("Data/Experiment_carbon.csv", "Data/Experiment_nutrients.csv"),  
                             ,names = c("carbon", "nutrients")  
                             ,stringsAsFactors = FALSE)  
ExperimentFiles
```

```
##               files      names  
## 1  ../Data/Experiment_carbon.csv  carbon  
## 2  ../Data/Experiment_nutrients.csv nutrients
```

Now we can step through the observations in `ExperimentFiles`, read in each file, and assign it to a data frame with a specified name.

```
for (n in 1:2) {  
  assign(x = ExperimentFiles$names[n], value = read.csv(file = ExperimentFiles$files[n]))  
  get(x = ExperimentFiles$names[n])  
}
```

There are several layers to this piece of code. The `assign` function assigns the value specified by `value` to a variable named `x`. When `n` is 1, the name of the variable will be first observation 1 in `ExperimentFiles$names`, and the value of the variable is the .csv file that we're reading in, specified by observation 1 in `ExperimentFiles$files`. When `n` is 2, the name and value of the variable will be the second observations in our data frame.

The last line in the for loop uses the `get` function. Try typing `ExperimentFiles$names[n]` without the `get` function.

The output is the `n`th character in that variable, i.e. `"carbon"` or `"nutrients"`. What we want, though, is the value of the variable that has that names. To do that, we use the `get` function, which searches for an object with the name specified by the `x` argument. This allows you to pass a list of names of variables to other functions.

Challenge

- Write a for loop that goes through the `nutrients` data frame and prints the value of each `Nitrite` observation plus 1.
 - Next, create a vector called `nitrite_plus`, and modify your for loop to add the “Nitrite plus 1” values to the end of this vector.
- Write a for loop that goes through the `nutrients` data and prints out whether or not dissolved inorganic nitrogen (the sum of nitrate, ammonium, and nitrite) is greater than 12. Use the `dplyr` function `slice`, which lets you select rows of a data frame by their order.