CALLING FUNCTIONS - CONDITIONAL STATEMENTS - IF ELSE IFELSE

DSC 205 - Advanced introduction to data science

Marcus Birkenkrahe

February 3, 2023

54

README

You will learn:

- ☐ How to control the flow and execution order in R code
- ☐ How to use `if`, `else` and `ifelse` statements in R

Download the **raw** practice file from GitHub and save it as `3_ifelse_practice.org`. To test your Emacs mettle, open it on the CMD line with the command `emacs -nw` (no graphics - not needed for this exercise).

`if` statements

- An `if` statement runs a block of code only if its condition is `TRUE`.
- In R, condition and loop statements are bundled in "Control Flow":

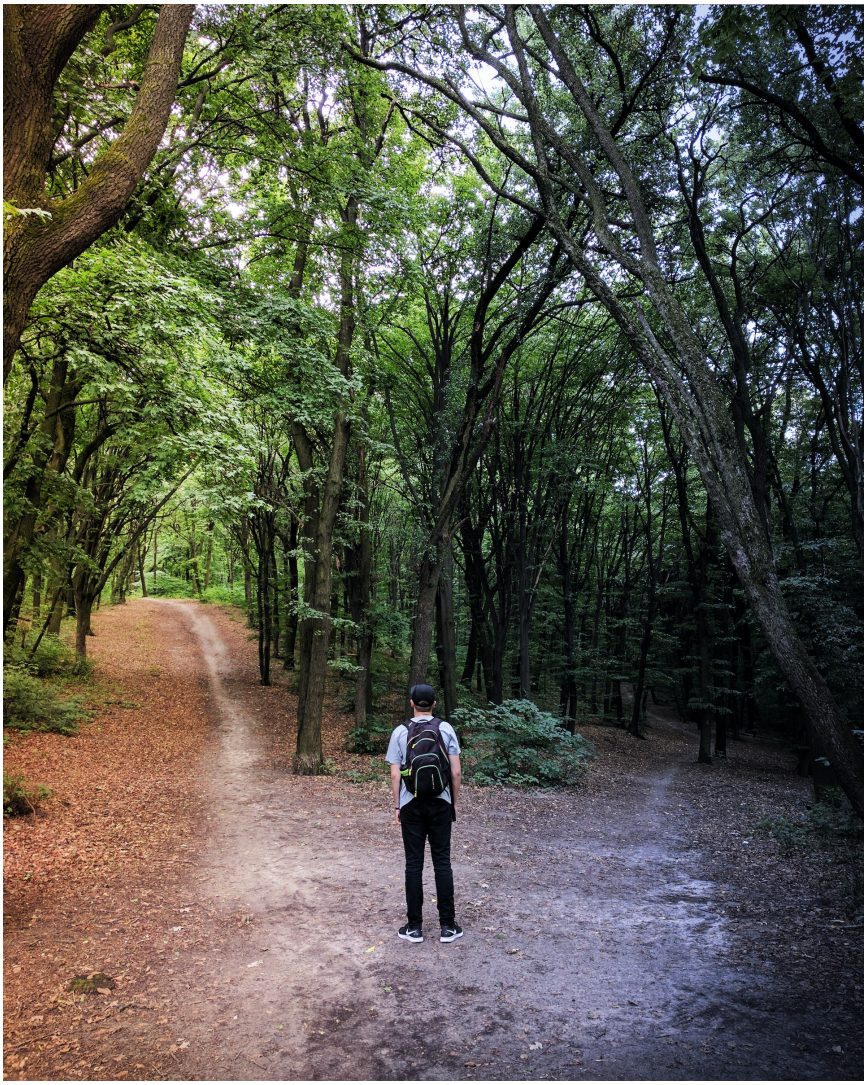

Figure 1: Photo by Vladislav Babienko on Unsplash

Figure 2: Photo by Robert Anasch on Unsplash

Control {base} R Documentation

Control Flow

Description

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language. They are all reserved words.

Usage

```
if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

- In this lecture, we'll discuss `if`, `else`, and `ifelse`.

Stand-alone `if` statements

- The *condition* must yield a single `logical` value, `TRUE` or `FALSE`:

```
if (condition) {
   do anything here
}
```

- Simple example:

  1. store two values
  2. compare the values (condition)
  3. do something inside the conditional statement

```
a <- 3
mynumber <- 4
if (a <= mynumber) {
  a <- a^2
}
```

- What value will `a` have afterwards?

```
a
```

```
[1] 9
```

- Run the code chunk again - what value will `a` have now?

`if` statement in the R console

Solution:

```
> ls()  # show object listing
[1] "a"        "mynumber"
> rm(a,mynumber)  # remove a, mynumber from objects
> ls()
character(0)
> a <- 3 ; mynumber <- 4  # assign values to a, mynumber
> if(a<=mynumber) {       # if a <= mynumber, square a
+ a <- a^2
+ }
> a   # print a
[1] 9
> options()$continue   # show console continuation character
[1] "+ "
> options(continue=">> ") # change continuation character
> ls(
>> )
[1] "a"        "mynumber"
```

1. Open the R console `*R*` (in Emacs: `C-x b`)
2. Clear the console ("flush output") with `C-c C-o`
3. Show the listing of all current R objects
4. Remove `a` and `mynumber` and check that they're gone
5. Enter `a <- 3` and `mynumber <- 4`
6. Enter the `if` statement on three different lines
7. Print `a`
8. Check the console continuation character `continue` in `options`
9. Change the console continuation character to `"» "`
10. Check the new character by running a command over 2 lines

Extended `if` example

- Create two new R objects to build a more complicated `if` statement:

```
myvec <- c(2.73, 5.40, 2.15, 5.29, 1.36, 2.16, 1.41, 6.97, 7.99, 9.52)
myvec
mymat <- matrix(c(2,0,1,2,3,0,3,0,1,1), 5, 2)
mymat
```

```
 [1] 2.73 5.40 2.15 5.29 1.36 2.16 1.41 6.97 7.99 9.52
     [,1] [,2]
[1,]    2    0
[2,]    0    3
[3,]    1    0
[4,]    2    1
[5,]    3    1
```

- Use `myvec` and `mymat` in this statement and run it:

```
if(any((myvec-1) > 9) || matrix(myvec,2,5)[2,1] <= 6) {
  cat("Condition satisfied -- \n")
  new.myvec <- myvec
  new.myvec[seq(1,9,2)] <- NA
  mylist <- list(aa = new.myvec, bb = mymat + 0.5)
  paste("-- a list with", length(mylist), "members now exists.")
}
```

```
Condition satisfied --
[1] "-- a list with 2 members now exists."
```

- You should have got this output:

```
: Condition satisfied --
: [1] "-- a list with 2 members now exists."
```

- Examine the list `mylist` you just created:

```
str(mylist)
mylist
```

```
List of 2
 $ aa: num [1:10] NA 5.4 NA 5.29 NA 2.16 NA 6.97 NA 9.52
 $ bb: num [1:5, 1:2] 2.5 0.5 1.5 2.5 3.5 0.5 3.5 0.5 1.5 1.5
```

```
$aa
 [1]   NA 5.40   NA 5.29   NA 2.16   NA 6.97   NA 9.52

$bb
     [,1] [,2]
[1,]  2.5  0.5
[2,]  0.5  3.5
[3,]  1.5  0.5
[4,]  2.5  1.5
[5,]  3.5  1.5
```

- Let's take the statement apart:

```
if(any((myvec-1) > 9) || matrix(myvec,2,5)[2,1] <= 6) {
  cat("Condition satisfied -- \n")
  new.myvec <- myvec
  new.myvec[seq(1,9,2)] <- NA
  mylist <- list(aa = new.myvec, bb = mymat + 0.5)
  paste("-- a list with", length(mylist), "members now exists.")
}
```

1. Subtract `1` from each value of `myvec` and compare with `9`, then check if `any` of the elements are `TRUE`:

```
myvec
(myvec-1) > 9
any((myvec-1) > 9)
```

```
 [1] 2.73 5.40 2.15 5.29 1.36 2.16 1.41 6.97 7.99 9.52
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[1] FALSE
```

2. Construct a 2 x 5 matrix using `myvec`, extract row 2, column 1, and compare with `6`:

```
matrix(myvec, 2, 5) # nrow = 2, ncol = 5
matrix(myvec, 2, 5)[2,1] <= 6
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] 2.73 2.15 1.36 1.41 7.99
[2,] 5.40 5.29 2.16 6.97 9.52
[1] TRUE
```

3. The condition is evaluated as `FALSE || TRUE`, hence `TRUE`, and the `if` statement is entered:

```
any((myvec-1) > 9) || matrix(myvec, 2, 5)[2,1] <= 6
```

```
[1] TRUE
```

4. `cat` is like `print` or `paste` with fewer coercions

```
cat("Condition satisfied -- \n")
```

```
Condition satisfied --
```

5. Copy `myvec` to `new.myvec` and replaces the odd-numbered indices of `new.myvec` and overwrites them with `NA`:

```
foo <- myvec
foo
foo[seq(1,9,2)] # seq from=1 to=9 by=2
foo[seq(1,9,2)] <- NA
foo
```

```
 [1] 2.73 5.40 2.15 5.29 1.36 2.16 1.41 6.97 7.99 9.52
[1] 2.73 2.15 1.36 1.41 7.99
 [1]   NA 5.40   NA 5.29   NA 2.16   NA 6.97   NA 9.52
```

6. Create a list `mylist` and store `new.myvec` as element `aa`. Increase all elements of `mymat` by `0.5` and store them as element `bb`.

```
list(aa = new.myvec, bb = mymat + 0.5)
```

```
$aa
 [1]   NA 5.40   NA 5.29   NA 2.16   NA 6.97   NA 9.52

$bb
     [,1] [,2]
[1,]  2.5  0.5
[2,]  0.5  3.5
[3,]  1.5  0.5
[4,]  2.5  1.5
[5,]  3.5  1.5
```

7. Print the `length` of the resulting list.

```
length(mylist)
```

```
[1] 2
```

`else` statements

- If you want something to happen if the *condition* is `FALSE`, add `else`:

```
if (condition) {
   do something if condition is TRUE
   } else {
     do something if condition is FALSE
   }
```

  - Example: initialize values

```
a <- 3
mynumber <- 4
```

  - Run the extended statement twice:

```
if (a <= mynumber) {
  cat("Condition was", a<=mynumber)
  a <- a^2
} else {
  cat("Condition was", a<=mynumber)
  a <- a - 3.5
}
a

Condition was TRUE> [1] 9
```

- After a few re-runs, the value of `a` will be smaller than `mynumber` again, and the first part of the `if` statement will be accessed.

`ifelse` for element-wise checks

- An `if` statement can only check the condition of a single value
- If you pass a `logical` vector for the condition, only the first element will be checked and operated on (and you'll be warned):

```
if (c(FALSE, TRUE, FALSE, TRUE, TRUE)) {}

Error in if (c(FALSE, TRUE, FALSE, TRUE, TRUE)) { :
  the condition has length > 1
```

- The function `ifelse` can perform vectorized checks.
- Example: create objects `x` and `y`

```
x <- 5
y <- -5:5
y
```

```
[1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

- Suppose you want to compute `x/y` but every time the result is `Inf` (division by zero) you want it to be replaced with `NA`. Running through `y==0` won't work because only the first element is checked:

```
y == 0
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

- Instead, use `ifelse` - the resulting object has the length of `test`:

```
result <- ifelse(
  test = (y==0),
  yes = NA,
  no = x/y)
result
```

```
[1] -1.000000 -1.250000 -1.666667 -2.500000 -5.000000        NA  5.000000
[8]  2.500000  1.666667  1.250000  1.000000
```

NEXT Exercises

Download the raw exercise file from GitHub and save it as `3_ifelse_exercise.org`.

TODO Glossary

TERM	MEANING

References

- Davies, T.D. (2016). The Book of R. NoStarch Press.