# CALLING FUNCTIONS - CONDITIONAL STATEMENTS - NESTING STACKING SWITCHING

DSC 205 - Advanced introduction to data science

Marcus Birkenkrahe

February 10, 2023

## README

Figure 1: Photo by La-Rel Easter on Unsplash

You will learn:

☐ How to nest `if` statements

□ How to stack `if` statements with `else if`

□ How to pick statements with `switch`

Download the **raw** practice file from GitHub and save it as `4_switch_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).

## Nesting and stacking

- An `if` statement can be placed within the outcome of another `if` statement: by *nesting* or *stacking* conditional statements, you can design specific decision-making patterns.

## Example: nesting

- In pseudocode notation:

```
IF a <= mynumber
   a <- a^2
   IF mynumber > 3
      b <- seq(1,a,length=mynumber)
   ELSE
      b <- a * mynumber
ELSE
   a <- a-3.5
   IF mynumber >= 4
      b <- a^(3-mynumber)
   ELSE
      b <- rep(a + mynumber, times=3)
```

- Nesting different statements for two variables `a` and `mynumber`:

```
if(a <= mynumber) {
  cat("First condition was TRUE\n")
  a <- a^2
  if(mynumber > 3) {
    cat("Second condition was TRUE\n")
    b <- seq(1,a,length=mynumber)
  } else {
    cat("Second condition was FALSE\n")
    b <- a * mynumber
  }
} else {
  cat("First condition was FALSE\n")
  a <- a - 3.5
  if(mynumber >= 4) {
    cat("Second condition was TRUE\n")
    b <- a^(3-mynumber)
  } else {
    cat("Second condition was FALSE\n")
    b <- rep(a + mynumber, times=3)
  }
}
a
b
```

- Run this code with `a <- 3` and `mynumber <- 4`:

```
a <- 3
mynumber <- 4
if(a <= mynumber) {
  cat("First condition was TRUE\n")
  a <- a^2
  if(mynumber > 3) {
    cat("Second condition was TRUE\n")
    b <- seq(1,a,length=mynumber)
  } else {
    cat("Second condition was FALSE\n")
    b <- a * mynumber
  }
} else {
  cat("First condition was FALSE\n")
  a <- a - 3.5
  if(mynumber >= 4) {
    cat("Second condition was TRUE\n")
    b <- a^(3-mynumber)
  } else {
    cat("Second condition was FALSE\n")
    b <- rep(a + mynumber, times=3)
  }
}
a
b
```

```
First condition was TRUE
Second condition was TRUE
[1] 9
[1] 1.000000 3.666667 6.333333 9.000000
```

- Reset `a <- 6` and `mynumber <- 4` and run the nested statements again. This time the first condition is not met but the second is, and `b` is computed with the new value of `a`[1].

```
a <- 6
mynumber <- 4
if(a <= mynumber) {
  cat("First condition was TRUE\n")
  a <- a^2
  if(mynumber > 3) {
    cat("Second condition was TRUE\n")
    b <- seq(1,a,length=mynumber)
  } else {
    cat("Second condition was FALSE\n")
    b <- a * mynumber
  }
} else {
  cat("First condition was FALSE\n")
  a <- a - 3.5
  if(mynumber >= 4) {
    cat("Second condition was TRUE\n")
    b <- a^(3-mynumber)
  } else {
    cat("Second condition was FALSE\n")
    b <- rep(a + mynumber, times=3)
  }
}
```

[1] In the code block, `«nested»` inserts the named code block (`#+name: nested`) and runs it.

```
a
b

First condition was FALSE
Second condition was TRUE
[1] 2.5
[1] 0.4
```

## Example: stacking

- You can stack `if` statements by placing a new `if` immediately after an `else` declaration:
- In pseudocode notation:

```
IF a <= mynumber AND mynumber > 3
   a <- a^2
   b <- seq(1,a,length=mynumber)
ELSE IF a <= mynumber AND mynumber <= 3
   a <- a^2
   b <- a * mynumber
ELSE IF a > mynumber AND mynumber >= 4
   a <- a-3.5
   b <- a^(3-mynumber)
ELSE
   a <- a-3.5
   b <- rep(a + mynumber, times=3)
```

- In R code:

```
if (a <= mynumber && mynumber > 3) {
  cat("First condition TRUE and second TRUE\n")
  a <- a^2
  b <- seq(1,a,length=mynumber)
} else if (a <= mynumber && mynumber <= 3) {
  cat("First condition TRUE and second FALSE\n")
  a <- a^2
  b <- a^(3 - mynumber)
} else if (mynumber >= 4) {
  cat("First condition FALSE and second TRUE\n")
  a <- a - 3.5
  b <- a^(3 - mynumber)
} else {
  cat("First condition FALSE and second FALSE\n")
  a <- a - 3.5
  b <- rep(a + mynumber, times=3)
}
a
b
```

- Let's run this twice as before to see if we get the same results:

```
a <- 3
mynumber <- 4
if (a <= mynumber && mynumber > 3) {
  cat("First condition TRUE and second TRUE\n")
  a <- a^2
  b <- seq(1,a,length=mynumber)
} else if (a <= mynumber && mynumber <= 3) {
  cat("First condition TRUE and second FALSE\n")
  a <- a^2
  b <- a^(3 - mynumber)
} else if (mynumber >= 4) {
  cat("First condition FALSE and second TRUE\n")
  a <- a - 3.5
  b <- a^(3 - mynumber)
} else {
  cat("First condition FALSE and second FALSE\n")
  a <- a - 3.5
  b <- rep(a + mynumber, times=3)
}
a
b
a <- 6
mynumber <- 4
if (a <= mynumber && mynumber > 3) {
  cat("First condition TRUE and second TRUE\n")
  a <- a^2
  b <- seq(1,a,length=mynumber)
} else if (a <= mynumber && mynumber <= 3) {
  cat("First condition TRUE and second FALSE\n")
  a <- a^2
  b <- a^(3 - mynumber)
} else if (mynumber >= 4) {
  cat("First condition FALSE and second TRUE\n")
  a <- a - 3.5
  b <- a^(3 - mynumber)
} else {
  cat("First condition FALSE and second FALSE\n")
  a <- a - 3.5
  b <- rep(a + mynumber, times=3)
}
a
b
```

```
First condition TRUE and second TRUE
[1] 9
[1] 1.000000 3.666667 6.333333 9.000000
First condition FALSE and second TRUE
[1] 2.5
[1] 0.4
```

## The `switch` function for `character` strings

- If you need to choose code based on the value of a single object, you can use a series of stacked `if` statements.
- Example: assign a `numeric` value to `foo` where the number depends on the value of `mystring`:

```
if ( mystring == "Homer" ) {
  foo <- 12
} else if ( mystring == "Marge" ) {
  foo <- 34
} else if ( mystring == "Bart" ) {
  foo <- 56
} else if ( mystring == "Lisa" ) {
  foo <- 78
} else if ( mystring == "Maggie") {
  foo <- 90
} else {
  foo <- NA
}
foo

Error: object 'mystring' not found
NULL
```

- Example runs:

```
mystring <- "Lisa"    # matched with foo = 78
if ( mystring == "Homer" ) {
  foo <- 12
} else if ( mystring == "Marge" ) {
  foo <- 34
} else if ( mystring == "Bart" ) {
  foo <- 56
} else if ( mystring == "Lisa" ) {
  foo <- 78
} else if ( mystring == "Maggie") {
  foo <- 90
} else {
  foo <- NA
}
foo
mystring <- "Peter"   # not in the list
if ( mystring == "Homer" ) {
  foo <- 12
} else if ( mystring == "Marge" ) {
  foo <- 34
} else if ( mystring == "Bart" ) {
  foo <- 56
} else if ( mystring == "Lisa" ) {
  foo <- 78
} else if ( mystring == "Maggie") {
  foo <- 90
} else {
  foo <- NA
}
foo
```

```
[1] 78
[1] NA
```

- The `switch` function behaves like a set of stacked `if` statements. Take a look at `help(switch)` to see its definition.

- Using the "Simpsons" example from before:

```
foo <- switch(
  EXPR = mystring,
  Homer=12,
  Marge=34,
  Bart=56,
  Lisa=78,
  Maggie=90,
  NA)
foo

[1] NA
```

- Example runs:

```
mystring <- "Lisa"    # matched with foo = 78
foo <- switch(
  EXPR = mystring,
  Homer=12,
  Marge=34,
  Bart=56,
  Lisa=78,
  Maggie=90,
  NA)
foo
mystring <- "Peter"   # not in the list
foo <- switch(
  EXPR = mystring,
  Homer=12,
  Marge=34,
  Bart=56,
  Lisa=78,
  Maggie=90,
  NA)
foo
```

```
[1] 78
[1] NA
```

- The first argument `EXPR` can be `numeric` or a `character` string
- The remaining arguments provide the values or operations based on the value of `EXPR`.

## `switch` for `integer` expressions

- If `EXPR` is an `integer`, the outcome is determined purely with *positional matching*:

  ```
  foo <- switch(EXPR=mynum,12,34,56,78,NA)
  foo
  ```

  ```
  NULL
  ```

- In the code, every other value for `mynum` than 1,2,3,4 will set `foo` to `NULL`, the "null" object (value is undefined).

  ```
  class(NULL)
  ```

  ```
  [1] "NULL"
  ```

- Examples:

  ```
  mynum <- 3
  foo <- switch(EXPR=mynum,12,34,56,78,NA)
  foo
  mynum <- 0
  foo <- switch(EXPR=mynum,12,34,56,78,NA)
  foo
  mynum <- 100
  foo <- switch(EXPR=mynum,12,34,56,78,NA)
  foo
  ```

  ```
  [1] 56
  NULL
  NULL
  ```

## TODO Exercises

Download the raw exercise file from GitHub and save it as `4_switch_exercise.org`.

## TODO Glossary

| TERM | MEANING |
|---|---|

## References

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