3. Control structures and functions in R

Transfer exploration seminar: Statistics and Data Science

Dr. Uma Ravat

PSTAT 194TR

Data Essentials Summary

Types of Statistical Data

- Numerical - discrete or continous
- Categorical - ordinal or nominal

EDA - Simple Techniques

- Data wrangling - variables, observations, data types
- Quantative data summary - center, spread, 5 number summary
- Visual data summary - bar plots, histogram, box plots

Disclaimer: Lot's of new terminology. Focus on how `R` handles things

Review after lecture

Maintain a glossary of functions used.

Next we will see...

- Control structures
- (User defined) functions
- Simulations

Control structures

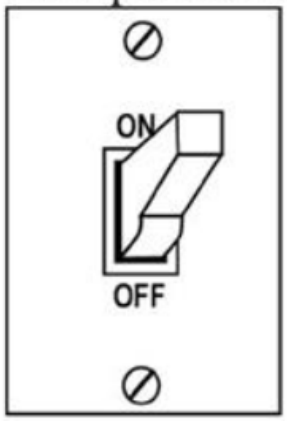
1. **Conditionals :** "If (condition is TRUE) do this... otherwise do that..."
2. **Iterators/loops :** "Repeat this action several times"
3. **user-defined functions:**

Conditionals in everyday language:

If you commit a crime

Then you go to jail

If I flip on the switch



Then the lights go on

`if` Syntax:

```
if (test expression){
  statement
}
```

- If the test expression is `TRUE`, the statement is executed.

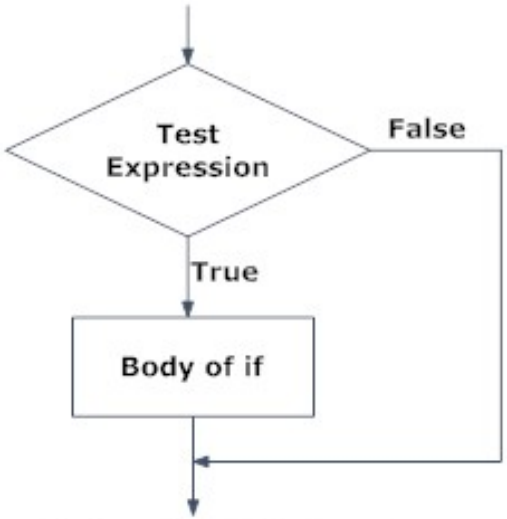


Fig: Operation of if statement

`if-else` Syntax:

- An `if` statement can be followed by an optional `else` statement

```
if (test expression1) {
  statement 1
} else {
  statement 2
}
```

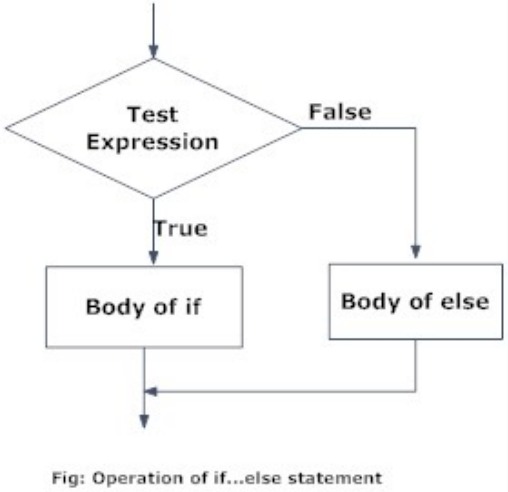
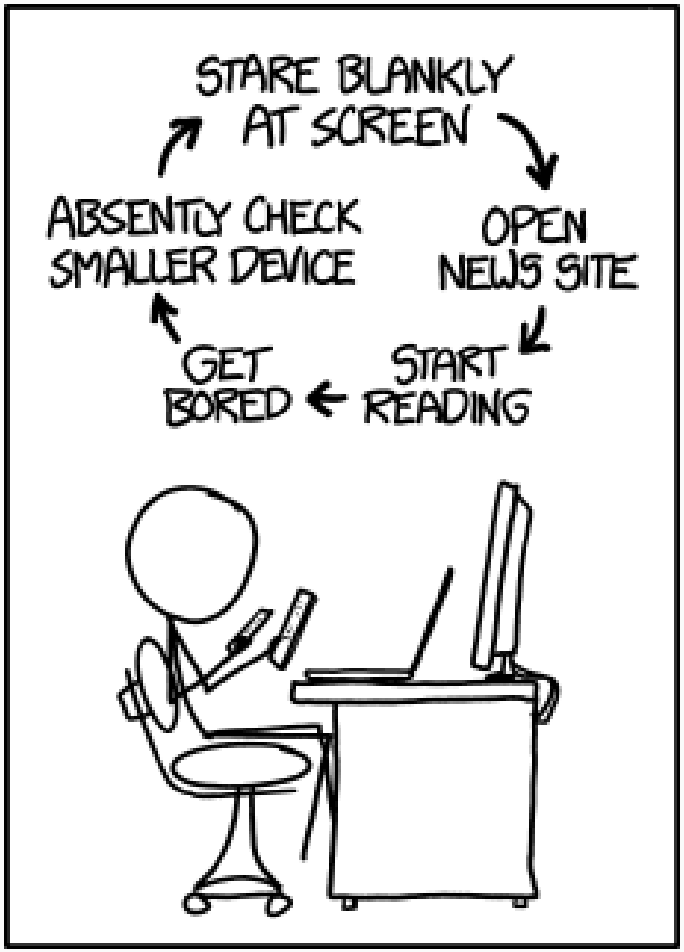


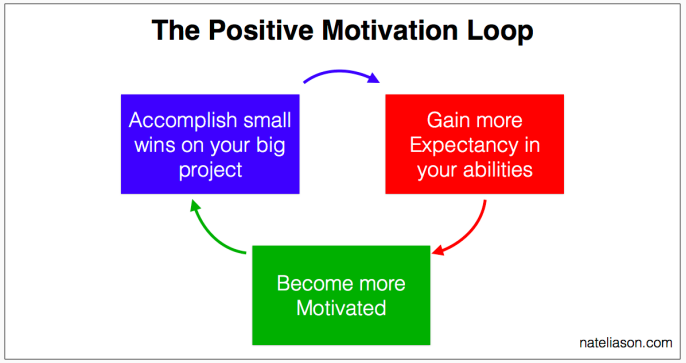
Fig: Operation of if...else statement

```r
x <- -5
if(x > 0){
print("Non-negative number")
} else {
print("Negative number")
}
```

```
## [1] "Negative number"
```

Loops or Iterators





A loop construct allows us to execute the same code multiple times.

Loops in `R`

- **`for` loop :** Executes a statement or sequence of statements multiple times. Tests the condition at the **end** of the loop.
- **while loop** Repeats a statement or sequence of statements while a given condition is true. Tests the condition **before** executing the loop.
- **repeat loo**p Executes a statement or sequence of statements multiple times until a stop condition is met.

for Loop SYNTAX

```
for (counter in counter-vector)
  {
    statements #body of for loop
  }
```

Fig: operation of for loop

Construct a table of logarithms

Use a `for` loop to construct a table of logarithms from 1 to 10.

```
table.of.logarithms <- vector(length=10, mode="numeric") #empty/null vector
table.of.logarithms
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
for (i in 1:length(table.of.logarithms)) {
  table.of.logarithms[i] <- log(i)
}

names(table.of.logarithms) <- 1:length(table.of.logarithms)
table.of.logarithms
```

```
##         1         2         3         4         5         6         7         8
## 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101 2.0794415
##         9        10
## 2.1972246 2.3025851
```

Tracing through the for loop is useful

counter: `i`	`table.of.logarithms[i]`
1	log(1)
2	log(2)
3	log(3)
...	...
10	log(10)

"**iterates over** the counter-vector"

Note, there is a better way to do this job!

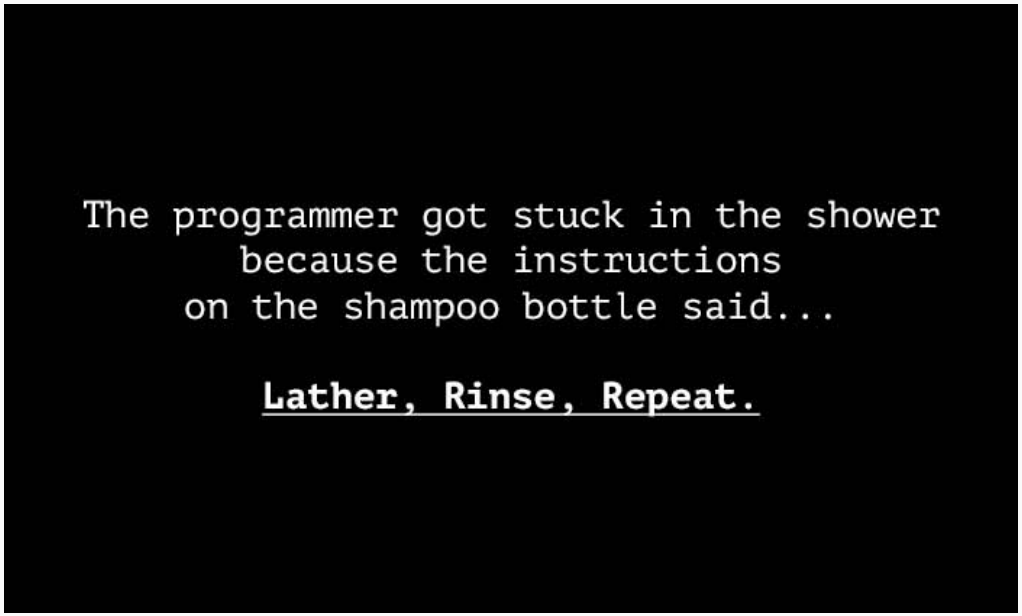
More about for loops in R

The body can contain just about anything, including:

- `if()` clauses
- other `for()` loops (nested iteration)

- `for` loops are not limited to numeric vectors in the counter vector. We can pass character vectors, logical vectors or expressions

Programming humor



Function in R

Functions in R are either

- built-in (free for you to use!)
- user-defined (you need to code them up)

Functions are (most often) verbs, followed by what they will be applied to in parentheses

function and arguments

```
do_this(to_this)
```

Here `do_this` is the function and `to_this` is the **argument** to the function

```
do_that(to_this, to_that, with_those)
```

Here `do_that` is the function and `to_this, to_that, with_those` are the three **arguments** to the `do_that` function

What should be a function?

- Things you're going to re-run, especially if it will be re-run with changes
- Chunks of code which are small parts of bigger analyses
- Chunks which are very similar to other chunks

`function` SYNTAX:

```
function_name <- function(arg1, arg2, ...)
  {
    code that does something
    return(object)
  }
```

1. **function Name:** choose a name for your function
2. **arguments** `arg1`, `arg2`, ...
   - An argument is a placeholder
   - When a function is called or invoked, you pass a value to an argument.
3. **function body:** Write code that does something
4. **return** or **print**: The result

A function to convert fahrenheit to centigrade.

Call (or invoke) the function with a fahrenheit temperature of 82.
(Test the function with input 82.)

```
fahrenheit_to_centigrade <- function( temp_F ) {
  temp_C <- ((temp_F - 32) * (5/9))
  return(temp_C)
}

fahrenheit_to_centigrade(82)
```

```
## [1] 27.77778
```

Summary:

- Control structures
  - Conditionals - `if`, `if-else`
  - Loops or iterators - `for`, `while`, `repeat`
- user-defined functions
- Simulations and probability

We just scratched the surface of this important topic in any programming language!

Maintain a glossary of functions used.

Next we will see...

Introduction to Probability