R

Alejandra E. Medina Rivera
Licenciatura en Ciencias Genómicas.
Centro de Ciencias Genómicas, UNAM

Cuernavaca, Mexico
Feb, 2010

ïf'

- `if` is the most simple control structure, and usage is simple: `if (cond1=vdd) {cmd1} else {cmd2}`
- `ifelse`, usage: `ifelse(prueba, valor-vdd, valor-falso)`

Example

ïf'

IF

```
> if (1 == 0) {
+     print(1)
+ } else {
+     print(2)
+ }
```

```
[1] 2
```

```
> x <- 1:10
> ifelse(x < 5 | x > 8, x, 0)
```

```
 [1]  1  2  3  4  0  0  0  0  9 10
```

Cycling

In R there are three cycling structures.

- `for`. Usage: `for(variable in sequence) {comandos}`
- `while`. Usage: `while(condition) {comandos}`
- `repeat`.

Example

Cycling

while

```
> y <- 12345
> x <- y/2
> while (abs(x * x - y) > 1e-10) x <- (x +
+       y/x)/2
> x

[1] 111.1081

> x^2

[1] 12345
```

Example

Cycling

repeat

```
> repeat {
+       x <- (x + y/x)/2
+       if (abs(x * x - y) < 1e-10)
+           break
+ }
> x
[1] 111.1081
```

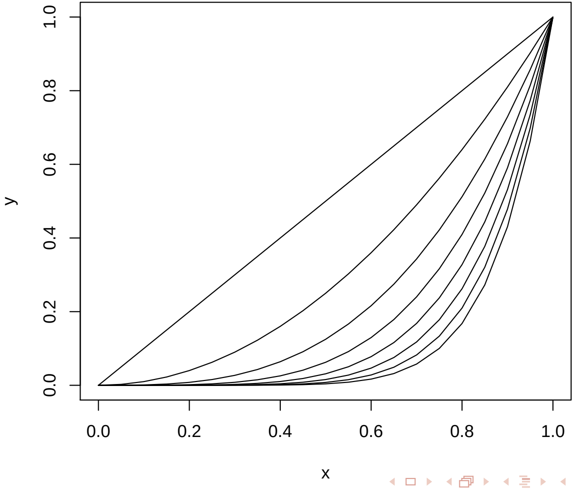
Example

Cycling

for

```
> x <- seq(0, 1, 0.05)
> plot(x, x, ylab = "y", type = "l")
> for (j in 2:8) {
+     lines(x, x^j)
+ }
```

Cycling



A common application of loops is to apply a function to each element of a set of values or vectors and collect the results in a single structure. In R this is abstracted by the functions lapply and sapply .

- lapply, returns a list.
- sapply, simplifies the result to a vector or a matrix if possible.

Example

```
> lapply(thuesen, mean, na.rm = T)
> sapply(thuesen, mean, na.rm = T)
```

Why this and not explicit loops?

- this functions attach meaningful names to the results
- is faster

An other function of this family is apply, which allows you to apply a function to the row or the columns of a matrix

```
> m <- matrix(rnmorm(12), 4)
> m
> apply(m, 2, min)
```

What happened?

Functions

- Functions are objects of type *function* and all functions in `R` as `mean` are of the same type.
- Functions are defined with:

```
> mifun <- function(arg1, arg2, ...) {
+     lo_que_sea
+ }
> mifun(arg1 = ..., arg2 = ...)
```

- Is better if all arguments have a *default* value. `arg1 = val.def`.
- When a function is called, the arguments can be set in the same order these are defined in the function, or you can use their *tag* (names)

Functions

- Inside the function there can be one or several instructions.
- The returned value of a function is the last one to be evaluated or the on defined with `return`.

```
> fact <- function(x = 1) {
+     ret <- 1
+     for (i in 1:x) {
+         ret = ret * i
+     }
+     return(c(x, ret))
+ }
> fact()
```

```
[1] 1 1
```

```
> fact(x = 5)
```

```
[1]   5 120
> fact(6)
[1]   6 720
```

Functions: control

- In R there are three control functions:
  return,stopwarning
- return specifies the value that has to be returned and ends the function.
- El stop stops the function and prints an error message.
- El warning prints a message but doesn't stop the function.

```
> mifun3 <- function(x1) {
+       if (x1 > 0) {
+             print(x1)
+       }
+       else if (x1 == 0) {
+             warning("Value must be > 0")
+       }
```

Functions: control

```
+       else {
+           stop("Hay un error porque x1 < 0")
+       }
+ }
```

```
> mifun3(x1 = 0)
> mifun3(x1 = -2)
```

```
[1] "Hay un error porque x1 < 0"
```

Regular Expression

R provides five regular expression functions in its base package. All these functions support three regular expression flavors. You have two parameters called `extended` and `perl` at your disposal to indicate the flavor you want. This time using the help you'll help me know what does each function do?

- `grep()`.

```
> grep("a+", c("abc", "def", "cba a",
+      "aa"), value = FALSE)
[1] 1 3 4
> grep("a+", c("abc", "def", "cba a",
+      "aa"), value = TRUE)
[1] "abc"   "cba a" "aa"
```

- `regexpr()`.

Regular Expression


```
> regexpr("a+", c("abc", "def", "cba a",
+       "aa"))
```

```
[1]  1 -1  3  1
attr(,"match.length")
[1]  1 -1  1  2
```

- gregexpr().

```
> gregexpr("a+", c("abc", "def", "cba a",
+       "aa"))
```

Regular Expression

```
[[1]]
[1] 1
attr(,"match.length")
[1] 1

[[2]]
[1] -1
attr(,"match.length")
[1] -1

[[3]]
[1] 3 5
attr(,"match.length")
[1] 1 1
```

Regular Expression


```
[[4]]
[1] 1
attr(,"match.length")
[1] 2
```

- sub().

```
> sub("(a+)", "z\\1z", c("abc", "def",
+       "cba a", "aa"))
```

```
[1] "zazbc"   "def"      "cbzaz a" "zaaz"
```

- sub().

```
> sub("(a+)", "z\\1z", c("abc", "def",
+       "cba a", "aa"))
```

```
[1] "zazbc"   "def"      "cbzaz a" "zaaz"
```

- sub().

Regular Expression


```
> gsub("(a+)", "z\\1z", c("abc", "def",
+      "cba a", "aa"))
[1] "zazbc"     "def"       "cbzaz zaz"
[4] "zaaz"
```

Data edition

R provides two ways of editing data interactively. One allow you to edit numeric variables in the workspace using the `data.entry` function, and the other lets you edit data frames. Both use the same spreadsheet-like interface. We will review her only the data frame editor.

- The interface is a bit rough but quiet useful for small data sates.
- This option only works if you are using the R interface provided for MAC and Windows. In Linux consoles it tends to fail depending on you graphic variables.

```
> data(airquality)
> aq <- edit(airquality)
```

- You can modify the data by typing on the cell.

Data edition

- You can change the type of variable by clicking on the header of the column.
- You can overwrite the data frame using function `fix()`
- To enter data into a blank data frame use

```
> dd <- data.frame
> fix(dd)
```

Exercise I

Let's do a small exercise

- Create a data frame using the next data

```
> Cylinder <- c(rep("V4", 5), "V6",
+       "V4", rep("V6", 3))
> Weight <- c(2170, 2655, 2345, 2560,
+       2330, 3325, 2745, 3735, 3450,
+       3265)
> Mileage <- c(33, 26, 33, 33, 26, 23,
+       25, 18, 22, 20)
> Type <- c("Sporty", "Compact", rep("Small",
+       3), "Large", "Compact", "Van",
+       rep("Medium", 2))
```

- Get the mean of the Milage

Exercise I

- Separate the data frame by Type into diferent data frames and get the mean
- Put this separated data frames into a list
- Save the original data frame in to a file ordered by Milage and Weight
- There is a function that will summarize all important things on a data frame like mean, sd, etc... Which is it? Use it

Hint: Yo used `read()` to read a file.

ExerciseII

- Read the file TFCFAutoTUGene.txt
- Separate into independent data frames the TFs that work like activators, repressors and duals
- Create a data frame containing the count of how many TFs are
  - activators and activate their own promotor
  - activators and repress their own promotor
  - activators with a dual function in their own promotor.
  - Same for repressors
  - Same for duals