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

Cuernavaca, Mexico
Feb, 2010

Introduction to R
Introduction to R and Statistics

1. Control Structures
2. Implicit loops
3. Function definition
4. Regular Expression in R
5. Data edition
6. Exercise
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
```r
> 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
```
In R there are three cycling structures.

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

Example
Cycling

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

```r
> 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 \texttt{R} this is abstracted by the functions \texttt{lapply} and \texttt{sapply}.

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

**Example**

\begin{verbatim}
> lapply(thuesen, mean, na.rm = T)
> sapply(thuesen, mean, na.rm = T)
\end{verbatim}

Why this and not explicit loops?
- this functions attach meaningful names to the results
- is faster

An other function of this family is \texttt{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 are objects of type `function` and all functions in R as `mean` are of the same type.

Functions are defined with:

```r
> 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 one defined with `return`.

```r
> 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)
```

```
13
```
Functions

> 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.
- `stop` stops the function and prints an error message.
- `warning` prints a message but doesn’t stop the function.

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

```r
+   else {
+       stop("Hay un error porque x1 < 0")
+   }
+
> mifun3(x1 = 0)
> mifun3(x1 = -2)

[1] "Hay un error porque x1 < 0"
```
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().**
  ```R
  > 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().**
> 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

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

- `sub()`.
  ```r
  > sub("(a+)", "z\1z", c("abc", "def", +    "cba a", "aa"))
  [1] "zazbc" "def" "cbzaz a" "zaaz"
  ```

- `sub()`.
  ```r
  > sub("(a+)", "z\1z", c("abc", "def", +    "cba a", "aa"))
  [1] "zazbc" "def" "cbzaz a" "zaaz"
  ```

- `sub()`.
Regular Expression

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

[1] "zazbc" "def" "cbzaz zaz"
[4] "zaaz"
```
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.

```r
> 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
  ```r
  > dd <- data.frame
  > fix(dd)
  ```
Exercise I

Let’s do a small exercise

- Create a data frame using the next data
  
  ```r
  > 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 Mileage
Exercise I

- Separate the data frame by Type into different data frames and get the mean
- Put this separated data frames into a list
- Save the original data frame into a file ordered by Mileage 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: You used `read()` to read a file.
Exercise II

- 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 promoter
  - activators and repress their own promoter
  - activators with a dual function in their own promoter.
  - Same for repressors
  - Same for duals