Principios de Estadística

Leonardo Collado Torres y María Gutiérrez Arcelus
Licenciatura en Ciencias Genómicas, UNAM
www.lcg.unam.mx/~lcollado/index.php
www.lcg.unam.mx/~mgutierr/index.php

Cuernavaca, México
Febrero - Junio, 2009

Introducción y R básico parte 2

El famoso "if"

R ofrece las estructuras de control más clásicas con lo cual luego podremos hacer funciones.

- El if es la estructura más simple y su sintaxis es bastante sencilla: `if (cond1=vdd) {cmd1} else {cmd2}`
- El ifelse no se diferencia tanto, aunque es una función. Mas bien es como en Excel; su sintaxis es: `ifelse(prueba, valor-vdd, valor-falso)`
- Aquí les mostramos un par de ejemplos:

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

El famoso "if"


```
[1] 2
```

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

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

Ciclos

- El for ya no es tan similar a lo que conocemos. Su sintaxis base es: `for(variable in secuencia) {comandos}`
- El segundo tipo de ciclo más usado es while. Su sintaxis base es: `while(condición) {comandos}`
- El último y que casi nunca se usa es el repeat. Realmente no se los recomendamos... en fin, aquí tienen un ejemplo de un ciclo `for`:

```
> x <- 1:10
> z <- NULL
> for (i in 1:length(x)) {
+     if (x[i] < 5) {
+         z <- c(z, x[i] - 1)
+     }
```

Ciclos


```
+      else {
+          z <- c(z, x[i]/x[i])
+      }
+ }
> z
 [1] 0 1 2 3 1 1 1 1 1 1
```

Usando un while

- Se acuerdan del problema de los números de ¿Fibonnaci? Bueno, lo podemos hacer con un `while` fácilmente.

```
> i <- 0
> j <- 1
> res <- c(i, j)
> while (2 * i + j < 1000) {
+     temp <- j
+     i <- i + j
+     j <- i + temp
+     res <- c(res, i, j)
+ }
> res
 [1]   0   1   1   2   3   5   8  13  21
[10]  34  55  89 144 233 377 610
```

Matrices

- R te permite tener variables de tipo matriz. Estas simplemente son vectores con un vector dimensional que es diferente de NULL.
- Si le cambias el vector de dimensiones a un vector, lo puedes volver una matriz[1]. Esto afecta como se imprime como ven a continuación:

```
> V <- runif(81)
> print(V[1:9])

[1] 0.06665838 0.50140651 0.92618926
[4] 0.74712320 0.87476979 0.33251160
[7] 0.22798309 0.50289956 0.59002158

> dim(V)
```

Matrices


```
NULL
> dim(V) <- c(9, 9)
> print(V[1:2, ])
           [,1]      [,2]      [,3]
[1,] 0.06665838 0.1233769 0.4924105
[2,] 0.50140651 0.4658742 0.2602117
           [,4]      [,5]      [,6]
[1,] 0.08575325 0.3205740 0.7239938
[2,] 0.05443288 0.6745646 0.3106217
           [,7]      [,8]       [,9]
[1,] 0.8000470 0.9766076 0.45549137
[2,] 0.6200741 0.2415377 0.05021652
```

[1] No a fuerzas es de 2 dimensiones!!

Con matrix

- Otra forma de definir una matriz es con la función matrix:

```
> args(matrix)

function (data = NA, nrow = 1, ncol = 1, byrow = I
NULL

> X <- matrix(1:16, 4, 4, byrow = TRUE)
```

- ¿Cómo es nuestra matriz X? Pueden poner nombres a las líneas o columnas usando rownames o colnames.

Índices en matrices

Para recuperar alguna columna o línea de una matriz usen los índices con el formato [línea,columna]. Por ejemplo, la línea 1 con `X[1,]` o la columna 2 con `X[,2]`.

Leer un archivo

- Algo esencial que todos sepan es como abrir un archivo o directorio en `R`. Dudamos mucho que quieran usar `scan()` y llenar los datos manualmente :P.
- `R` puede manejar varios archivos con números para una sola variable, tablas de números, archivos tipo `csv` y más. Por ejemplo, podríamos haber leido la info de los fagos así: `fagos <- scan(file="fagos.txt")`[2]
- Las funciones principales para leer archivos son `scan()`[3], `read.table()`, `read.csv()` y `source()`.
- Si quieren especificar el archivo de entrada cuando ejecuten el comando, pueden usar `read.table(file=file.choose())`.

[2] El archivo tendría que estar en el mismo folder donde estamos trabajando

[3] Especificando el archivo de entrada

Data Frame

Data Frames

Un formato muy usado en `R` son los *data frames*. Estos en realidad son como una hoja de cálculo donde cada columna es una variable. Pueden accesar a cada columna con `dataframe$variable` o `dataframe[["variable"]]`. Además pueden usar las funciones `attach` y `detach` para agregar las variables de un `data frame` al ambiente de `R`.[4]; la función `with(data.frame, comando)` hace lo mismo. Finalmente, pueden ver el principio o el final de un `data frame` o matriz usen `head()` o `tail()`.

[4] No es recomendable si piensan modificar los valores del `data frame` o si ya tienen variables con los mismos nombres

read.table

Example (Leer una tabla)

Para leer una tabla con algo de info sobre unos fagos usen:

```
> arch <- "10biggestPhages.txt"

> fagos.gr <- read.table(file = arch,
+     header = TRUE)
```

R también te permite leer archivos que están en servidores web. Esta misma tabla también la pueden leer así[5]:

read.table


```
> sitio <- "http://kabah.lcg.unam.mx/~lcollado/E/"
> sitio <- paste(c(sitio, "data/10biggestPhages.txt")
+     collapse = "")
> fagos.gr <- read.table(file = url(sitio),
+     header = TRUE)
> fagos.gr <- read.table(file = sitio,
+     header = TRUE)
> fagos.gr[c(2:4)]
```

	GenomeSize	EMBL	Taxid
1	280334	AF399011_GR	169683
2	252401	AY939844_GR	268746
3	244834	AY283928_GR	75320
4	233234	AY266303_GR	227470
5	211215	AJ697969_GR	273133

read.table


```
6       196280 AJ630128_GR 238854
7       185683 AP008983_GR  12336
8       180500 AY967407_GR 115991
9       178249 AY940168_GR 268747
10      176847 DQ149023_GR 382359
```

```
> fagos.gr$Taxid[2:3]
```

```
[1] 268746  75320
```

- Noten que las 2 formas de `read.table` son iguales, solo que una es más clara en su sintaxis. Además, el output de `read.table` es un `data frame`.
- Chequen los argumentos de la función `read.table`; en especial `sep` y `header`.

[5] Mejor usen `sitio <- "http://kabah.lcg.unam.mx/~lcollado/E/data/10biggestPhages.txt"` – tuvimos que hacerlo de otra forma por el espacio

Directorios

- Muchas veces quieres abrir más de un archivo de un directorio o folder. Tal vez no quieres abrir todos, así que tienes que buscar un patrón en sus nombres.
- La forma más automática de hacerlo es así:

```
> files <- list.files(pattern = "s.txt$")
> for (i in files) {
+     x <- read.table(i, header = TRUE)
+     assign(i, x)
+     print(i)
+ }

[1] "10biggestPhages.txt"
[1] "fagos.txt"
```

Que son

- R ofrece diferentes objetos como son los vectores atómicos[6], matrices y data frames. Otro de estos son los list.
- Las `list` en realidad consisten de una colección de objetos conocidos como sus componentes. Estos pueden ser de cualquier tipo como ven aquí:

```
> lista <- list(nombre = "Leo", hermano = "Alex",
+     edad = 21, calif.alumnos = c(6,
+         9, 10, 8, 7))
> lista$nombre == lista[[1]]
```

```
[1] TRUE
```

```
> lista$calif.alumnos[1] == lista[[4]][1]
```

```
[1] TRUE
```

Que son


```
> var <- "hermano"
> lista[["hermano"]] == lista[[var]]
```

```
[1] TRUE
```

[6] Donde todos los elementos son del mismo tipo

Accesando una lista

- Como se habrán dado cuenta, hay diferentes formas de accesar a una lista. En general, puedes accesar a cada elemento usando `lst[[i]]` donde i va desde 1 hasta `length(lst)`.
  - $ es útil por si no se acuerdan de que posición corresponde al elemento que quieren recuperar.
  - `lista[[var]]` es bastante útil si el nombre del elemento que quieren accesar está en una variable.
  - Si el elemento de la lista es un vector, pueden accesar a las diferentes posiciones como en el ejemplo de `lista[[4]][1]`.
- Es muy importante que noten la diferencia entre `lista[1]` y `lista[[1]]`. El primero te regresa una *sublista* mientras que el segundo te regresa el primer elemento de la lista.

Crear una lista

- Crear una lista es bastante sencillo como ya vieron. Es recomendable que especifiques los nombres de cada elemento aunque no es obligatorio.

```
> lista <- list(nom.1 = ele.1, ...,
+     nom.n = ele.n)
```

- Una vez creada una lista, pueden añadirle elementos así:

```
> lista[n + 1] <- list(nom.m = ele.m)
```

- Pueden contatenar listas usando `c()`:

```
> lista.ABC <- c(lista.A, lista.B,
+     lista.C)
```

- Finalmente, pueden borrar elementos de la lista usando `<- NULL`

Que son

- Otro tipo de objeto en `R` son los factor. Estos los pueden ver como vectores que tienen alguna información con respecto a la clasificación de los datos.
- En sí son como enumeraciones en otros lenguajes y son útiles para generar datos tabulares.
- Cuando usan la función `read.table`, todo lo que parece un caracter es leido como un `factor`
- Luego lean más sobre la función cut para aprender a generar datos tabulares.
- Un factor no es de tipo numérico! Por ejemplo, no pueden usar la función `mean`.

Un ejemplo

- Aquí les mostramos un ejemplo donde usamos un `factor` ^^:

```
> fiesta <- factor(sample(c("muerto",
+     "happy", "pedo", "sobrio"),
+     100, replace = TRUE, prob = c(0.1,
+         0.4, 0.3, 0.2)))
> fiesta[1:4]

[1] sobrio happy  sobrio pedo
Levels: happy muerto pedo sobrio

> table(fiesta)

fiesta
 happy muerto   pedo sobrio
    45      7     24     24
```

Un ejemplo

Substituciones

Perl es excelente para manejar strings, pero R también puede hacer sustituciones con la función sub. Por ejemplo:

```
> fiesta2 <- sub("o$", "os", as.character(fiesta),
+     perl = TRUE)
> fiesta2[1:10]

 [1] "sobrios" "happy"   "sobrios"
 [4] "pedos"   "muertos" "sobrios"
 [7] "happy"   "pedos"   "happy"
[10] "sobrios"
```