Programación científica en R

Introducción a R

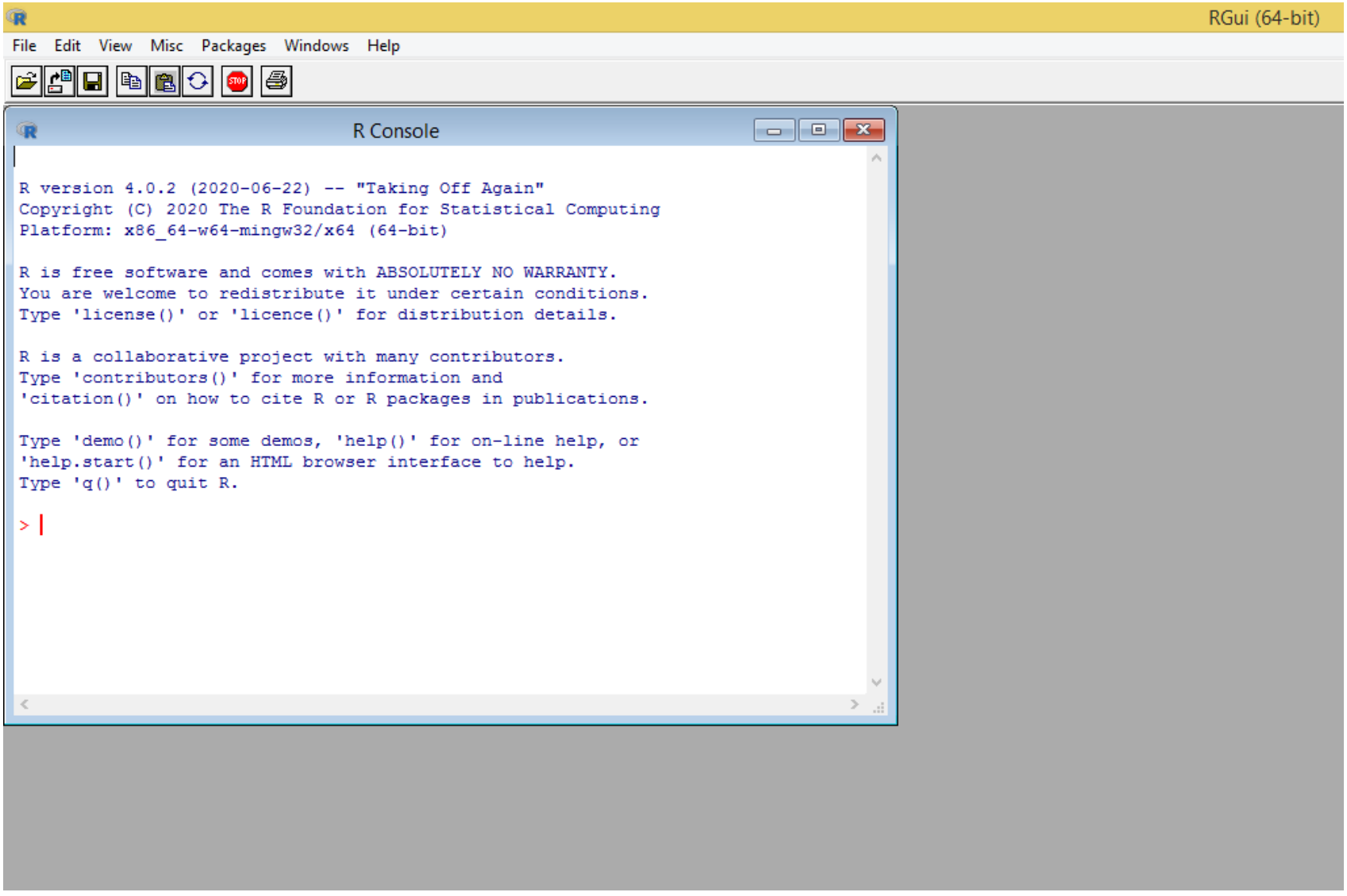
Marcos Ehekatzin García Guzmán

Agosto de 2024

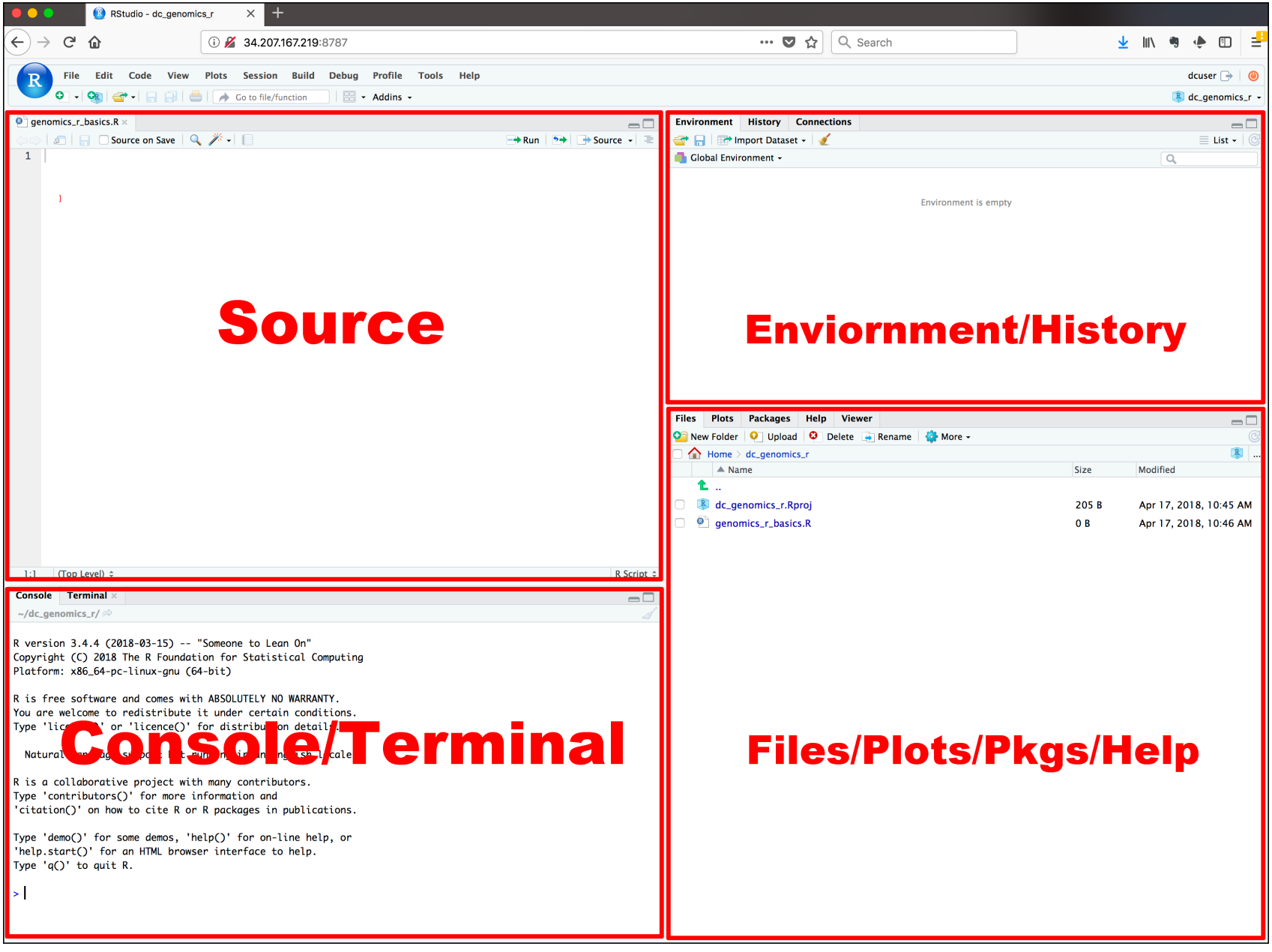
Introducción

Instalación de R

- Los recursos necesarios para instalar R en Windows, Mac y Linux se encuentran en The Comprehensive R Archive Network.
- Si abrimos R tal y como lo descargamos, nos daremos cuenta de que el programa solo consiste en una consola.
- En esta consola podremos escribir códigos en lenguaje R para hacer distintas operaciones y procesos (ver figura 5).
- Aunque esta versión de R es funciona, no es la forma más cómoda para trabajar o aprender a programar.


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>

- Hoy en día, lo más común es que utilicemos **RStudio** para programar utilizando el lenguaje R.
- **RStudio** es una aplicación como lo podría ser Microsoft Word, solo que en lugar de ayudarnos a escribir en español o inglés, nos ayuda a escribir en lenguaje R.
- Además, la interfaz de **RStudio** es mucho más amigable y nos permite acceder a múltiples herramientas adicionales.
- La aplicación es gratis y puede descargarse en: https://posit.co/products/open-source/rstudio/


RStudio - dc_genomics_r
34.207.167.219:8787
Search
File Edit Code View Plots Session Build Debug Profile Tools Help
dcuser
Go to file/function
Addins
dc_genomics_r
genomics_r_basics.R
Source on Save
Run
Source
Source
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
Enviornment/History
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Home dc_genomics_r
Name Size Modified
dc_genomics_r.Rproj 205 B Apr 17, 2018, 10:45 AM
genomics_r_basics.R 0 B Apr 17, 2018, 10:46 AM
Console Terminal
~/dc_genomics_r/
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Console/Terminal
Files/Plots/Pkgs/Help

La interfaz de *RStudio* tiene cuatro elementos:

1. Source: En esta parte de la interfaz podremos ver y escribir **Scripts**.[1] Esta es el área que más vamos a utilizar, ya que nos permite darle replicabilidad a nuestros códigos.
2. Console/Terminal:

- La consola es igual a la interfaz de R que tendríamos si no usaramos RStudio, aquí podemos ejecutar código directamente pero no lo podemos guardar para replicarlo más tarde.
- La terminal nos da acceso al shell del sistema, que nos permite interactuar con el sistema interactivo (no la usaremos).

[1] Los *Scripts* son archivos de texto que contienen instrucciones de programación que se ejecutarán en R.

3. Environment/History:

- En la pestaña Environment podremos ver los datasets, objetos, funciones, etc. que hemos creado y que se encuentran en la memoria.
- En la pestaña History veremos el historial de comandos que hemos ejecutado.

4. El último elemento de la interfaz es multipropósito. Aquí podremos navegar y establecer el directorio de trabajo (Files), visualizar las gráficas que hemos hecho (Plots), activar y desactivar paqueterías (Packages) y acceder a información sobre comandos y paqueterías (Help).

Consideraciones iniciales

help y example

Para obtener información detallada sobre una función, podemos utilizar el comando `help`.

- Por ejemplo, para obtener información sobre la función `solve` utilizaremos el comando `help(` de la siguiente manera:

```
> help(solve)
```

- La información aparecerá en la pestaña Help. [2]
- También podemos pedir un ejemplo sobre la función utilizando el comando `example()`.

```
> example(solve)
```

[2] Alternativamente podemos utilizar `?solve`

El operador **asignar**

En R podemos crear y manipular `objetos`.

- Los objetos guardan información que puede estar en distintas formas (números, funciones, datasets, listas, etc.). Esto lo veremos más adelante.
- Para asignar información a un objeto utilizaremos el operador "`<-`".
- Por ejemplo, si escribimos `x <- 50`, estaremos guardando el número 50 en el objeto llamado **`x`**.[3]

```
x <- 50
x
```

```
## [1] 50
```

[3] Noten que ahora el objeto x aparecerá en la pestaña Environment.

- Para hacer un nuevo Script basta con hacer clic en: *File –> New File –> R script.*
  - Tip: Para hacer notas en un script utilizaremos `#` al inicio de cada línea que no sea código.
- Muchas veces tendremos que saber en qué directorio de nuestra computadora estamos trabajando. Para ello utilizaremos el comando `getwd()`
- Muchas otras veces necesitaremos cambiar el directorio. Para ello utilizaremos el comando `setwd()`.[4]

```
setwd("C:/Users/user/Desktop")
```

[4] Asegurense de que la dirección utiliza `/` en lugar de `\`

Vectores y números

- R utiliza difeentes **estructuras de datos**, donde la más simple es el **vector numérico**
  - Un vector numérico no es más que una coleccion ordenada de números.
  - Para crear un vector (y) que contenga los siguientes números: 10.4, 5.6, 3.1, 6.4 y 21.7 utilizaremos la siguiente línea de código:

```
x <- c(10,5,3,6,9)
x
```

```
## [1] 10  5  3  6  9
```

Vectores y números

- Noten que, además del operador **asignar**, estamos usando una nueva función: `c()` para definir un vector numérico.
  - La función `c()` en este contexto está tomando varios argumentos y combinándolos en uno solo, que es el vector numérico.
- Noten que también podemos manipular los vectores numéricos.
  - Por ejemplo si utilizamos la expresión `1/x` obtendremos el inverso de los valores del vector. ¿Qué le pasa al nuestro objeto `x`?
  - Si, por ejemplo hacemos un nuevo vector tal que: `y <- c(x,0,x)` ¿Qué obtendríamos?

- Los vectores se pueden usar en expresiones aritméticas, de tal forma que la operación se realiza elemento por elemento.
- Ejercicio: Tomando los vectores `x` y `y` que ya definimos haremos un vector nuevo (`v`) que sea igual a la multiplicación de `x` y `y`.[5]
  1. ¿De qué longitud es el vector `v`?

[5] Usen el símbolo `*` para hacer la multiplicación

Aritmética vectorial

- Los vectores se pueden usar en expresiones aritméticas, de tal forma que la operación se realiza elemento por elemento.
- Ejercicio: Tomando los vectores `x` y `y` que ya definimos haremos un vector nuevo (`v`) que sea igual a la multiplicación de `x` y `y`.
  1. ¿De qué longitud es el vector `v`?

```
v <- x*y
```

```
## Warning in x * y: longitud de objeto mayor no es múltipl
## menor
```

```
v
```

```
##  [1] 100  25   9  36  81   0  50  15  18  54  90
```

2. ¿Cómo se obtuvieron los valores del vector `v`?

Aritmética vectorial

- Los vectores se pueden usar en expresiones aritméticas, de tal forma que la operación se realiza elemento por elemento.
- Ejercicio: Tomando los vectores `x` y `y` que ya definimos haremos un vector nuevo (`v`) que sea igual a la multiplicación de `x` y `y`.
  1. ¿De qué longitud es el vector `v`?
  2. ¿Cómo se obtuvieron los valores del vector `v`?
     - Debido a que el vector `y` es más largo, el vector resultante tendrá esa longitud. Además, los elementos del vector más corto (`x`) se irán reciclando hasta cumplir con la operación para todos los elementos de `y`.

Aritmética vectorial

- Los operadores aritméticos son los habituales (`+`,`-`,`*`,`/`, y `^`).
- También podemos usar otro tipo de funciones como `log`, `exp`, `sin`, `cos`, `tan`, `sqrt`.
- Otras funciones muy utilizadas son `max` y `min` que seleccionan respectivamente el valor más grande y pequeño de un vector; `range` que nos da un vector de longitud dos; `length` que nos da la longitud de un vector; `sum` nos da la suma de todos los elementos y ‘`prod` que nos da el producto de todos los elementos.
- Ejemplos:

```
max(x)
```

```
## [1] 10
```

```
sum(x)
```

```
## [1] 33
```

Ejercicio:

1. Hacer un vector que se llame `edad` que contenga la edad de todos los asistentes en la clase.
2. Calcular lo siguientes valores y guardar cada uno en un objeto nuevo.

a. El rango de las edades.
b. El número de elementos del vector (N).
c. El promedio de las edades. $\bar{x} = \frac{1}{N} \sum_{i=1}^{n} x_i$
d. La varianza de las edades. $\sigma^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{N-1}$

Vectores lógicos

- Además de los vectores numéricos, R también maneja *vectores lógicos*.
  - Los vectores lógicos solamente pueden tomar dos valores: `True` o `False`.
  - Para hacer un vector lógico deberemos utilizar condiciones.
  - Ejemplo: Recordemos el vector `x` que contiene los valores 10, 5, 3, 6 y 9. Para hacer un vector lógico podríamos utilizar la siguiente expresión:

```
temp <- x == 3
temp
```

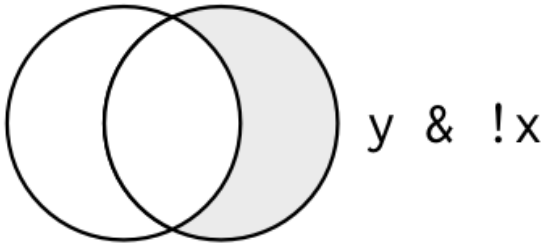
```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

- Noten que a diferencia de los vectores numéricos, en los vectores lógicos es necesario utilizar condiciones y los valores del vector dependerán si los elementos cumplen con dicha condición.

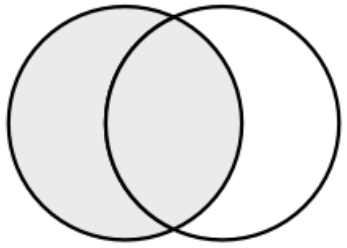
Vectores lógicos

- Para establecer condiciones es necesario utilizar operadores lógicos.
  - Los operadores lógicos son `==` (igual), `<` (menor), `<=` (menor o igual), `>` (mayor), `>=` (mayor o igual) y `!=` (distinto).
  - Además, si tenemos dos condiciones (por ejemplo `c1` y `c2`) podemos combinarlas de distintas maneras:
    - `c1&c2` (conjunción): Las dos condiciones deben cumplirse.
    - `c1|c2` (disyunción): Cualquiera de las dos condiciones debe cumplirse.
    - En la siguiente slide hay más ejemplos sobre la combinación de condiciones.

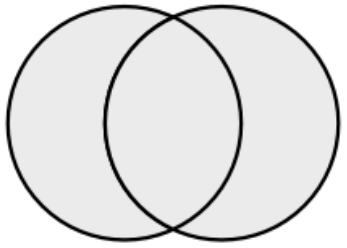
Vectores lógicos



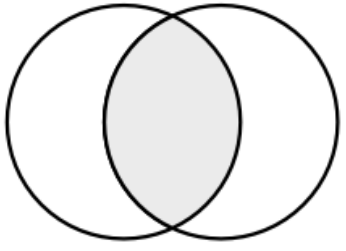
`y & !x`



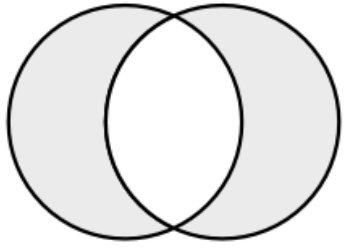
`x`



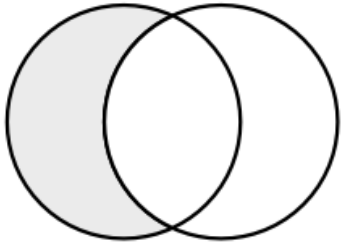
`x | y`



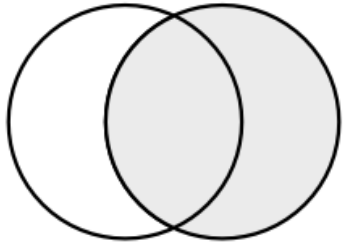
`x & y`



`xor(x, y)`



`x & !y`



`y`

Vectores lógicos

- Los vectores lógicos los podemos utilizar en conjunto con vectores numéricos.
- Ejemplo: si queremos saber cuantos valores del vector `x` son mayores o iguales a 6 podemos utilizar:

```
length(x[x>=6])
```

```
## [1] 3
```

- Ejemplo: si queremos saber cuantos valores de `x` son distintos a 3 y a 10

```
length(x[x!=3 & x!= 10])
```

```
## [1] 3
```

- Ejercicio: Con ayuda de vectores lógicos conteste las siguientes preguntas y guarde las respuestas en objetos nuevos:

a. ¿Cuántos asistentes tienen más de 28 años?
b. ¿Qué edades tienen dichos asistentes?
c. ¿Cuál es el promedio de edades de los asistentes si no tomamos en cuenta a quienes tienen más de 28 años?

Valores faltantes

Valores faltantes

- A veces algunos elementos de un vector son desconocidos, a éstos les llamamos **valores faltantes**.
- A los valores faltantes se le asigna el valor `NA`.
- Es importante saber si los vectores tienen o no valores faltantes, ya que (casi) cualquier operación que contenga valores faltantes tendrá por resultado un valor faltante.
- Podemos usar la función `is.na()` para detectar los valores faltantes en un vector.
- Ejemplo:

```
z <- c(1:3,NA)
ind <- is.na(z)
ind
```

```
## [1] FALSE FALSE FALSE  TRUE
```

Vectores de caracteres/strings

Vectores de caracteres/strings

- Un vector de strings se contruye escribiendo entre comillas la sucesión de caracteres que la define:

```
label <- c("Altura", "Peso", "IMC")
label
```

```
## [1] "Altura" "Peso"   "IMC"
```

- También podemos usar la función `paste()` para unir vectores de caracteres suministrados en uno solo.

```
label <- paste(c("X", "Y"), 1:8, sep = "")
label
```

```
## [1] "X1" "Y2" "X3" "Y4" "X5" "Y6" "X7" "Y8"
```

Vectores de índices

Vectores de índices

- Puedes seleccionar parte de un vector añadiendo un *vector de índices* entre corchetes.Los vectores de índices pueden ser de cuatro tipos:

1. Vector lógico: El vector de índices debe tener la misma longitud del vector al que se refiere. El resultante será el conjunto de valores que cumplen con la condición.

Ejemplo:`y<-x[!is.na(x)]`

2. Vector de números naturales positivos: Los elementos del vector deben pertenecer al conjunto $1, 2, \ldots, length(x)$. El resultado sera el vector de valores que corresponden a los índices en el orden en el que aparecen.

Ejemplo: `y<-x[1:10]`

3. Números naturales negativos: En este caso se indican los elementos del vector que serán excluidos.

Ejemplo: `y<-x[-(1:5)]`

4. Vector de caracteres: Este solo puede utilizarse si el vector posee atributos `names` para identificar los componentes.

Ejemplo:

```
fruta <- c(5,10,1,20)
names(fruta) <- c("Naranja", "Plátano", "Manzana", "Pera")
fruta[c("Manzana", "Naranja")]
```

```
## Manzana Naranja
##       1       5
```

Clases de objetos

Clases de objetos

- Los vectores son el tipo más básico de objeto que `R` utiliza, pero existen más tipos que usaremos más adelante en el curso:
  - **Matrices**: También llamadas *Arrays* (variables indexadas) son vectores multidimencionales que se encuentran indexados por dos o más índices.
  - **Factotes**: Sirven para representar datos categóricos
  - **Listas**: Son una forma feneralizada de vectores en los que los elementos no tienen por qué ser del mismo tipo.
  - **Data frames**: Son similares a una matriz. Cada una de las columnas en un data frame puede contener un tipo distinto de valores. Este es uno de los objetos con el que más trabajaremos a lo largo del curso.
  - **Funciones**: Son objetos que nos permiten programar procesos para hacer distintos cálculos, lo que nos permite extender las capacidades de `R`.

Modos y atributos

Atributos intrínsecos: Modo y atributos

- Como ya vimos, las entidades que manipula R se conocen como *objetos*.
- Hasta ahora el tipo de objetos que hemos visto se denominan *atómicos* debido a que todos sus elementos son del mismo tipo o *modo* (numérico, lógico, caracteres).
- Así el vector será del mismo modo que sus elementos.
- Solo hay una excepción, que surge cuando un vector contiene valores faltantes (`NA`).
- Cabe destacar que, incluso si un vector es vacío éste tendra un modo.

Atributos intrínsecos: Modo y atributos

- Con el modo de un objeto podemos designar el tipo básico de sus elementos. Con la función `mode()` podemos obtener el modo de un objeto.
- Con `R` podemos modificar el modo de cualquier objeto.
- Por ejemplo, considere el vector `z`

```
z <- 0:9
mode(z)
```

```
## [1] "numeric"
```

```
#Para hacer z un vector de caracteres
digits <- as.character(z)
#Para hacer digits un vector numérico
d <- as.integer(digits)
```

- Se puede utilizar la función `as.lo que sea` para cambiar el modo.

Modificación de la longitud de un objeto

- Recordemos que aunque un objeto esté vacío, tiene modo. Por ejemplo:

```
#Ejemplo 1
mode(x[x<0])
```

```
## [1] "numeric"
```

```
#Ejemplo 2
v <- numeric()
mode(v)
```

```
## [1] "numeric"
```

Modificación de la longitud de un objeto

- Una vez creado un objeto pueden añadirse nuevos elementos simplemente asignándolos a un índice que esté fuera del rango original:

```
v[3] <- 17
```

- ¿Cuál será la longitud del vector `v`? ¿Cuáles serán sis elementos?
- De la misma manera, puede reducirse la longitud de un objeto simplemente volviendo a hacer una asignación

```
alfa <- 1:10
alfa <- alfa[2*1:5]
```

- ¿Cómo se modificó el objeto `alfa`?