

@somaquadrados

Tipos de objetos

Data frame

vector



unidimensional
Una clase (numérico o de texto)

Matrix

	col1	col2	col3	col4	col5
linea 1					
linea 2					
linea 3					
linea 4					
linea 5					

columnas

bidimensional
Una clase (numérico o de texto)

Data frame

	col1	col2	col3	col4	col5
linea 1					
linea 2					
linea 3					
linea 4					
linea 5					

columnas

bidimensional
múltiples clases

Tipos de objetos

Data frame

area = [urb, rur, urb, rur, urb, rur]

mes = [1, 1, 2, 2, 3, 3]

presencia = [T, T, F, F, T, T]

temperatura = [20,1 21,2 22,2 23,8 22,9 23,0]



temp	pres	mes	area
20,1	T	1	urb
21,2	T	1	rur
22,2	F	2	urb
23,8	F	2	rur
22,9	T	3	urb
23,0	T	3	rur

Tipos de objetos

Data frame

Cómo construir un **data frame** en R: `data.frame()`.

```
# Vamos a crear 4 vectores
area <- c("urb", "rur", "urb", "rur", "urb", "r
mes <- c(1, 1, 2, 2, 3, 3)
presencia <- c(T, T, F, F, T, T)
temperatura <- c(20.1, 21.2, 22.2, 23.8, 22.9,
                area; mes; presencia; temperatura
## [1] "urb" "rur" "urb" "rur" "urb" "rur"
## [1] 1 1 2 2 3 3
## [1] TRUE TRUE FALSE FALSE TRUE TRUE
## [1] 20.1 21.2 22.2 23.8 22.9 23.0
```

```
# Unamos los vetores en un dataframe.
# Observe que cada vector se convierte en una columna
dtf <- data.frame(area, mes, presencia, temperatura)
dtf
```

```
##   area mes presencia temperatura
## 1  urb  1      TRUE     20.1
## 2  rur  1      TRUE     21.2
## 3  urb  2     FALSE    22.2
## 4  rur  2     FALSE    23.8
## 5  urb  3      TRUE     22.9
## 6  rur  3      TRUE     23.0
```

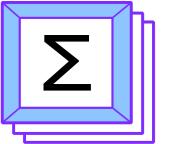
Tipos de objetos

Data frame

Ejercicio

Cree un [data frame](#) con los datos de su investigación (máximo de 6 filas y 6 columnas). Si no tiene datos, utilice los datos a continuación:

localidad	tiempo	poblacion	accidentes
A	1	10326	396
A	2	9658	400
B	1	6985	236
B	2	6300	123
C	1	3265	238
C	2	4005	632

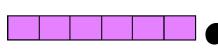
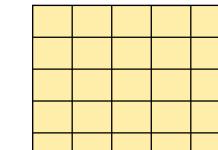
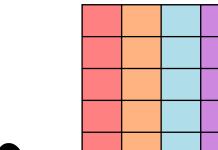
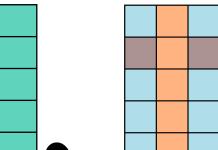
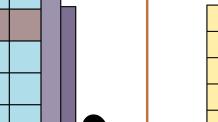
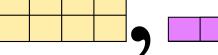


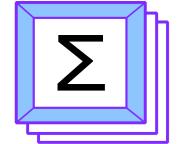
@somaquadrados

Tipos de objetos

List

- Colección **unidimensional** de objetos.
- Almacena datos de ≠ tipos (**vectors, arrays, data frame, lists**).
- Es un vector especial que acepta objetos como elementos.
 - Muchas funciones que usamos para analizar datos en R tienen listas como salida.

```
mi_lista = [  ,  ,  ,  ,  ,  ]
```



@somaquadrados

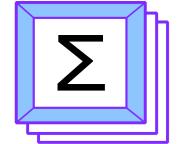
Tipos de objetos

List

crea una **lista** en r: `list()`.

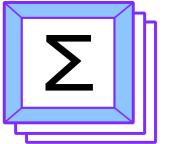
```
lis <- list(rbind(c(3,6), c(4,5)),  
            sample(1:100, 5),  
            factor(2, 6, 9))  
lis
```

```
## [[1]]  
##      [,1] [,2]  
## [1,]    3    6  
## [2,]    4    5  
##  
## [[2]]  
## [1] 42 61 51 26 64  
##  
## [[3]]  
## [1] <NA>  
## Levels: 9
```



@somaquadrados

Estructuras de Control



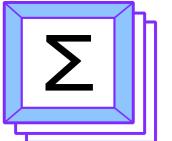
Estructuras de Control

- **¿Por qué programar?** 

- Evite la repetición innecesaria de análisis o cálculos que se repiten con frecuencia.
- Documente los pasos que tomó para llegar a un resultado.
- Fácil recuperación y modificación del programa.

- **¿Cómo programar?** 

- ¡Creando programas! (guiones/scripts, rutinas, algoritmos).
- Crear una secuencia lógica de comandos que se deben ejecutar en orden.
- Utilizar las herramientas básicas de programación:
 - Estructuras de repetición (`for()`)
 - Estructuras de selección (`if()`)
 - Estructura de repetición (`while()`)



Estructuras de Control

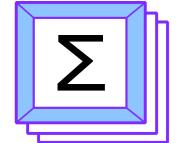
Estructuras de repetición: **for()**

- Sirve para repetir uno o más comandos varias veces.

```
for(<índice> in <valores>){code}  
      ↓          ↓          ↓          ↓          ↓  
  "para"    "cada i"    "en"     "objeto x"  "ejecutar el código"
```

- En otras palabras, estamos diciendo que para **cada elemento i** contenido en el **conjunto de valores** ejecutaremos los comandos que están dentro de las llaves (**code**).
- El índice no tiene que ser **i**, en realidad puede ser cualquier letra o palabra.

```
for(w in conjunto_de_valores){code}
```



@somaquadrados

Estructuras de Control

Estructuras de repetición: **for()**

- Para facilitar la comprensión, veamos dos ejemplos muy simples:
 - Imprimamos los números del 1 al 6.

```
# Creamos un vector con valores entre 1 y 6
objeto1 <- 1:6

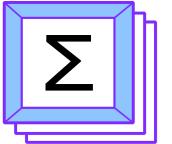
for(v in objeto1){ # Para cada valor v en 'objeto1'
  print(v) # imprime el valor v
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

```
# También podemos pedirle a R que imprima
# valores entre 1 y 6 directamente:

for(p in 1:6){ # Para cada valor p entre 1 y 6
  print(p) # imprime el valor p
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```



@somaquadrados

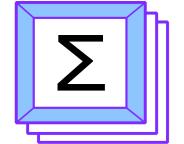
Estructuras de Control

Estructuras de repetición: **for()**

- Para facilitar la comprensión, veamos dos ejemplos muy simples:
 - Le pedimos a la r que sume +1 a cada valor entre 1 y 6 y luego imprima.

```
for(m in 1:6){ # Para cada valor m entre 1 y 6 (= i)
  a <- m + 1 # agregue el valor en m con 1 y guárdelo en "a"
  print(a) # imprime el objeto a
}

## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```



@somaquadrados

Estructuras de Control

Estructuras de repetición: **for()**

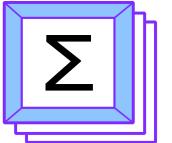
Muestreó especies en tres ubicaciones diferentes: loc1, loc2 y loc3. Al final del muestreo, desea calcular el total de especies muestreadas. Haga esto usando el comando "**for()**".

```
loc1 <- c(1, 8, 5)
loc2 <- c(4, 7, 1)
loc3 <- c(9, 4, 3)

mt <- rbind(loc1, loc2, loc3)
colnames(mt) <- c('T1', 'T2', 'T3')

mt
```

```
##      T1 T2 T3
## loc1  1  8  5
## loc2  4  7  1
## loc3  9  4  3
```



@somaquadrados

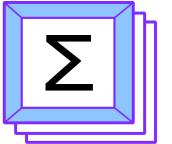
Estructuras de Control

Estructuras de repetición: **for()**

Muestreó especies en tres ubicaciones diferentes: loc1, loc2 y loc3. Al final del muestreo, desea calcular el total de especies muestreadas. Haga esto usando el comando "**for()**".

```
for(loc in 1:3){ # Para cada uno de los 3 loc's...
  s <- sum(mt[loc, 1:3]) # ... sumar las columnas entre 1 y 3
  print(s) # imprimir el resultado
}
```

```
## [1] 14
## [1] 12
## [1] 16
```



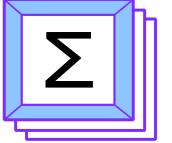
@somaquadrados

Estructuras de Control

Estructuras de repetición: **for()**

Ejercicio

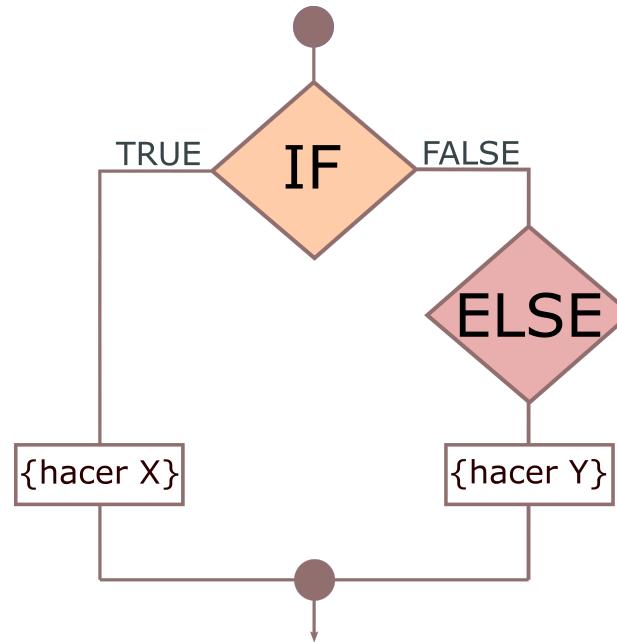
- 1 - Escriba un loop **for** que calcule el cubo de cada número entre 1 y 7 usando la función **print()**.
- 2 - Pesó a diferentes personas con 1.80 m de altura y obtuvo los siguientes valores: 70, 85, 90, 68. Cree un loop **for** para calcular el IMC de estas personas según la siguiente expresión matemática: $\text{IMC} = \text{Peso} \div (\text{Altura} \times \text{Altura})$.



Estructuras de Control

Estructuras de selección: `if()`

- Una estructura de selección - `if()` - sirve para ejecutar algún comando solo si se satisface alguna condición (en forma de expresión condicional).



En español, piense en el `if` como la palabra "*SI*" y el `else` como "*DEMÁS*"

Estructuras de Control

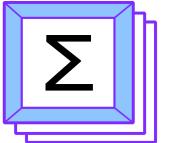
Estructuras de selección: **if()**

- La sintaxis siempre estará en la forma:



- En R:

```
if(<condicion 1>){
  # comandos que satisface la condición
}else{
  # comandos que NO satisface la condición
}
```



@somaquadrados

Estructuras de Control

Estructuras de selección: **if()**

Es posible que deseemos clasificar los niveles de vitamina D de los pacientes en "ideales (>20)" y "no ideales (<21)".

```
vitamina_D <- sample(0:60, 5)
vitamina_D

## [1] 48 43 37  7 53

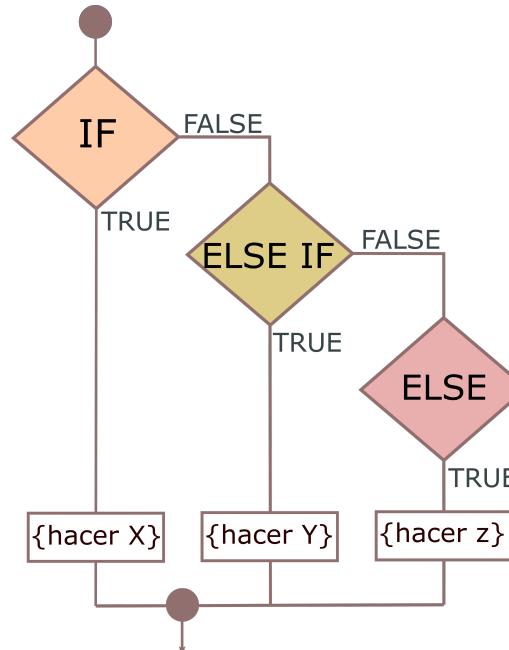
for(paciente in vitamina_D){ # para cada uno de los pacientes en "vitamina_D"...
  if(paciente > 21){print("ideales")} # si el paciente tiene un valor > 21, imprima "ideal"
  else{print("no ideales")} # demás, imprima "no ideales"
}

## [1] "ideales"
## [1] "ideales"
## [1] "ideales"
## [1] "no ideales"
## [1] "ideales"
```

Estructuras de Control

Estructuras de selección: `if()`

- Podemos usar `else if()` para poner otras condiciones.

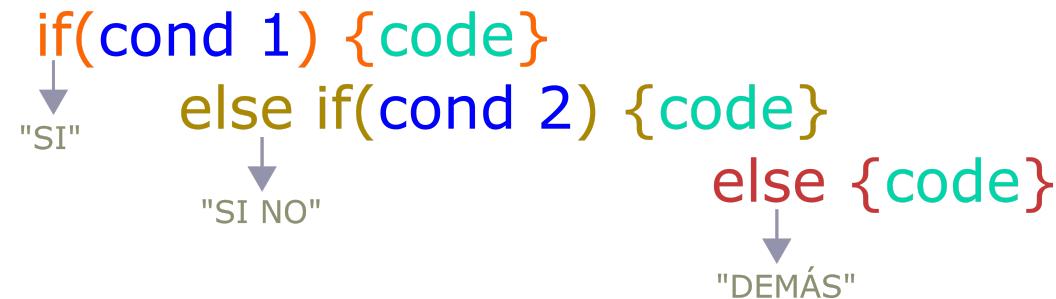


El `else if` es una condición intermedia entre `if` y `else`.

Estructuras de Control

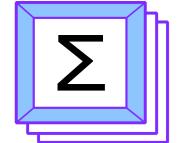
Estructuras de selección: **if()**

- La sintaxis es:



- En R:

```
if(condición 1){
  # comandos que satisface la condición 1
} else if (condición 2){
  # comandos que satisface la condición 2
} else { # comandos que NO satisface las condiciones
}
```



@somaquadrados

Estructuras de Control

Estructuras de selección: **if()**

Se realizaron pruebas de glucosa en cuatro pacientes. A partir de los resultados se desea realizar la siguiente clasificación: 70-99 mg/dl = normal; 100-125 mg/dl = prediabetes; > 126 mg/dl = diabetes.

```
glicose <- sample(70:130, 4); glicose
## [1] 100 120  90  85

for(paciente in glicose){ # para cada paciente con datos de glucosa...
  if(paciente <= 99){print("normal")} # si el paciente tiene un valor <=99 - diabetes normal
  else if(paciente >= 126){"diabetes"} # si el paciente tiene valores superiores o iguales a 126 - d
  else{print("prediabetes")} # los demás son prediabetes.
}

## [1] "prediabetes"
## [1] "prediabetes"
## [1] "normal"
## [1] "normal"
```

Estructuras de Control

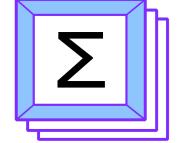
Estructuras de selección: **if()**

Ejercicio

Tomando como entrada la altura y el sexo (codificados de la siguiente manera: 1 = mujer 2 = hombre) de una persona, calcule e imprima su peso ideal, usando las siguientes fórmulas:

- para mujeres: $(62.1 * \text{altura}) - 44.7$
- para hombres: $(72.7 * \text{altura}) - 58$

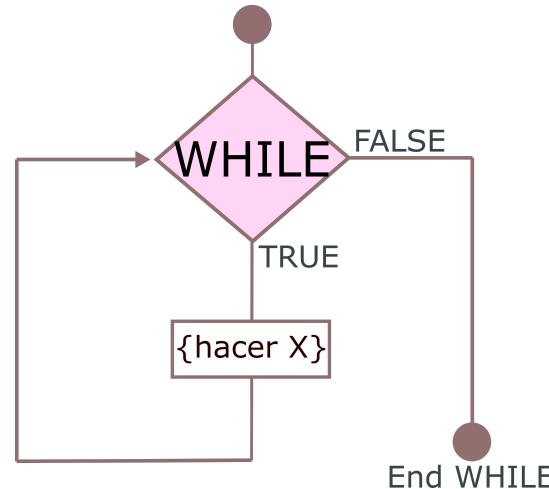
cod	altura
1	1.65
1	1.72
2	1.78
2	1.81



Estructuras de Control

Estructura de repetición: `while()`

- Un ciclo `while` en la programación R es una función diseñada para ejecutar algún código hasta que se cumpla una condición.



- Si bien la condición lógica es VERDADERA, el código no dejará de ejecutarse.

El loop `while` es muy similar al loop `for`, pero en el segundo definirás el número de iteraciones a ejecutar.

Estructuras de Control

Estructura de repetición: `while()`

- Para un loop `while` necesitas usar la función con la siguiente sintaxis:

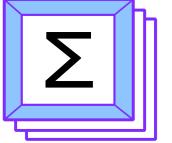
```
while(condición lógica) {code}
```

Diagrama que explica la sintaxis del bucle while. Los términos están etiquetados con flechas apuntando a las partes correspondientes del código:

- "...ejecutar el código!" apunta a la palabra "code".
- "Mientras que..." apunta a la palabra "condición lógica".
- "...la condición no es TRUE..." apunta al operador lógico entre paréntesis.

- En :

```
while(condición lógica){  
  # Code  
}
```



@somaquadrados

Estructuras de Control

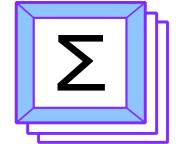
Estructura de repetición: **while()**

- Por ejemplo, dado que n = 5, siempre que no sea igual a 0, R no detendrá la ejecución del ciclo.

```
n = 5

while(n > 0){ # siempre que el valor de n sea mayor que 0 ...
  print("R está trabajando")
  print(n)
  n = n - 1
}
```

```
## [1] "R está trabajando"
## [1] 5
## [1] "R está trabajando"
## [1] 4
## [1] "R está trabajando"
## [1] 3
## [1] "R está trabajando"
## [1] 2
## [1] "R está trabajando"
## [1] 1
```



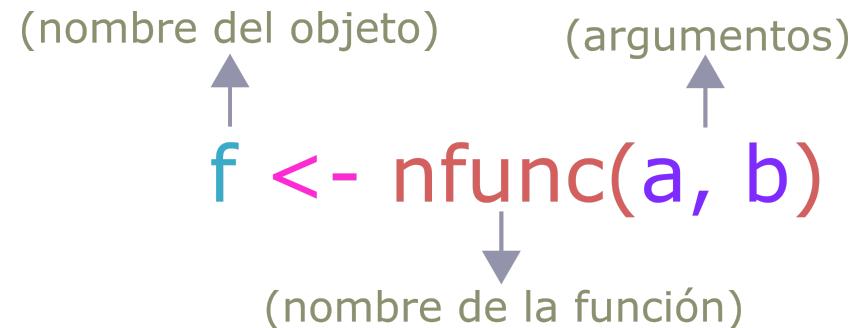
@somaquadrados

Funciones

Funciones



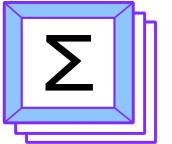
- Mientras que los **objetos** son *nombres que contienen valores*, las **funciones** son *nombres que contienen un código R*.



- La idea básica de una función es encapsular un código que se pueda invocar en cualquier momento en R.

nfunc(a,b) → {code} → resultado
(ejecutar código)

Usamos algunas funciones hasta ahora: `c()`, `rep()`, `data.frame()`, `class()`, otros.



@somaquadrados

Funciones

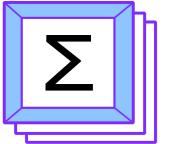
Argumentos

- Las funciones toman **argumentos**.
- Los argumentos son los valores u objetos que ponemos entre paréntesis y que las funciones necesitan un par funcional (calculando un resultado).
- Por ejemplo, la función `class()` necesita recibir un objeto para investigar la clase y devolverlo:

```
a <- 3  
class(a)
```

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

| En este caso, "a" es el argumento que incluimos en la función `class()`.



@somaquadrados

Funciones

Argumentos

- Para las funciones que toman más de un argumento, tenemos que separar los argumentos con comas.
- Por ejemplo, cuando usamos la concatenación (`c()`) para crear un **vector**.

```
ve <- c(1, 2, 3, 4)
```

Importante:

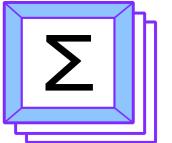
!! Observe cómo debe ser la entrada de valores para que funcione la función.

```
class(1, 2, 3, 4)
```

```
# simpleError in class(1, 2, 3, 4): 4 argumentos passados para 'class', que requer 1
```

```
class(ve)
```

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



@somaquadrados

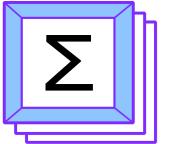
Funciones

Argumentos

Los argumentos de las funciones también tienen **nombre**, que pueden o no ser usando en la función. Por ejemplo a función `rep()`.

```
rep(x,  
     times = 1,  
     length.out = NA,  
     each = 1)
```

- **x**: valores que se repetirán.
- **times**: Un vector de valor entero que da el número (no negativo) de veces que se repite cada elemento si tiene una longitud (x), o que se repite todo el vector si tiene una longitud 1.
- **length.out**: O comprimento desejado do vetor de saída.
- **each**: Cada elemento de x é repetido todas as vezes.



@somaquadrados

Funciones

Argumentos

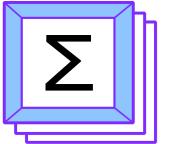
¿Cuál es la diferencia entre las salidas?

```
rep(x = 1:3,  
     times = 3,  
     length.out = NA,  
     each = 2)
```

```
## [1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
```

```
rep(x = 1:3,  
     times = NA,  
     length.out = 10,  
     each = 2)
```

```
## [1] 1 1 2 2 3 3 1 1 2 2
```



@somaquadrados

Funciones

Argumentos

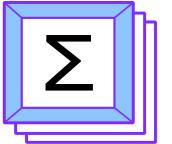
También podemos usar la función sin incluir los nombres de los argumentos:

```
rep(x = 1:3,  
     times = 2,  
     length.out = NA,  
     each = 1)
```

```
## [1] 1 2 3 1 2 3
```

```
rep(1:3,  
    2,  
    NA,  
    1)
```

```
## [1] 1 2 3 1 2 3
```



Funciones

Creación

- Las funciones en **R** son muy similares al de las funciones matemáticas, es decir, hay un *nombre*, una *definición* y posterior *invocación de la función*.
- Siempre que ejecute una función, el código que almacena se ejecutará y se devolverá el resultado.
- Además de usar las funciones listas, puede crear su propia función. La sintaxis es la siguiente:

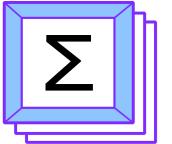
```
f <- function(a, b) {code}
```

(nombre de la función)

(argumentos)

(comando para crear función)

(código ejecutará la función)



@somaquadrados

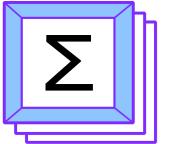
Funciones

Creación

- En R:

```
# creando una función llamada 'f':  
f <- function(a, b) {  
  code  
}  
  
# invocando la función:  
f(a, b)
```

Tenga en cuenta que `function` es un nombre reservado en R, es decir, no podrá crear un objeto con ese nombre.



@somaquadrados

Funciones

Creación

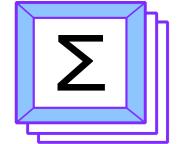
Creemos una función para calcular el peso ideal de las mujeres en función de la altura:

```
peso <- function(altura){  
  p <- (62.1 * altura) - 44.7  
  p  
}
```

- Nuestra función tiene los siguientes argumentos:
 - **peso**: nombre de la función
 - **altura**: argumento de la función
 - **p <- (62.1 * altura) - 44.7**: operación que realizará la función
 - **p**: valor devuelto por la función

```
peso(1.7) # resultado para 1.70m
```

```
## [1] 60.87
```



@somaquadrados

Funciones

Argumentos

- Nuestra función también toma un vector como argumentos:

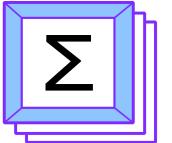
```
v <- c(1.7, 1.5, 1.65)
peso(v)
```

```
## [1] 60.870 48.450 57.765
```

- Y también podemos usar con conjuntos de control:

```
# Usando nuestra función en un loop 'for'.
for(i in v){
  print(peso(i))
}
```

```
## [1] 60.87
## [1] 48.45
## [1] 57.765
```



@somaquadrados

Funciones

!! Observaciones importantes:

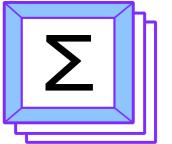
- De forma predeterminada, las funciones siempre devuelven la última línea de código como resultado de la función. En nuestro caso, es el valor contenido en '**p**'.

```
peso <- function(altura){  
  p <- (62.1 * altura) - 44.7  
  p # nuestra función devolverá p  
}
```

- ¿Y qué pasa si eliminamos **p** de nuestra función?...

```
peso <- function(altura){  
  p <- (62.1 * altura) - 44.7  
}  
  
peso(1.65)
```

- ... ¡La R no devuelve nada!



@somaquadrados

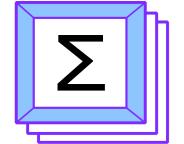
Funciones

!! Observaciones importantes:

- En el caso comentado, la última línea de código es la función matemática que será el 'valor' devuelto.
- En este caso, para ver el resultado de la función debemos hacer:

```
peso <- function(altura){  
  p <- (62.1 * altura) - 44.7  
}  
  
p1 <- peso(1.65)  
p1  
  
## [1] 57.765
```

- En nuestro caso, es como si lo estuviéramos haciendo directamente: `p1 <- (62.1 * 1.65) - 44.7.`

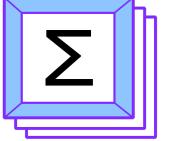


@somaquadrados

Funciones

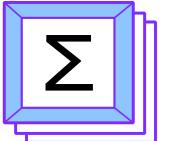
- ¿Existe una función lista para mi problema?
- ¿Cómo averiguar el nombre de esta función?





@somaquadrados

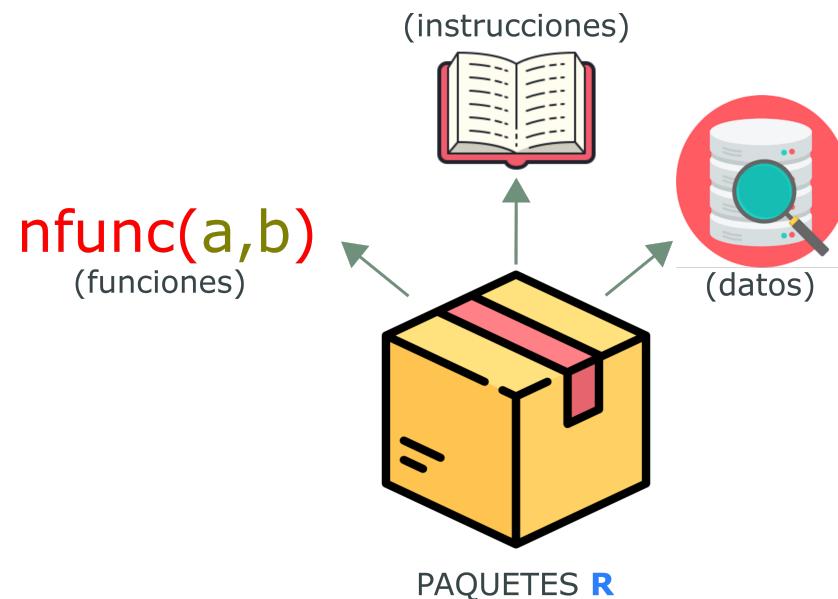
Paquetes



@somaquadrados

Paquetes

- Las funciones provienen de dos fuentes:
 1. paquetes **R** estándar que se cargan siempre que trabajamos con el lenguaje
 2. paquetes que instalamos y cargamos por comandos.
- Básicamente, un paquete es una convención para organizar y estandarizar la distribución de funciones **R**.



Paquetes

- La principal motivación de crear un paquete **R** es de organizar y compartir funciones de nuevos métodos y/o implementaciones creadas y que son útiles para otras personas.
- En general, descargamos paquetes de dos fuentes: **CRAN** y **GitHub**.

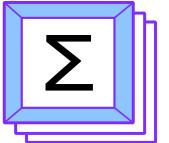


(paquetes listos
para usar)



(paquetes en
desarrollo)

Paquetes



@somaquadrados

Instalación

- Para instalar paquetes desde **CRAN** usamos el comando `install.packages("nombre_paquete")`.

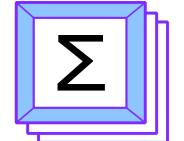
```
install.packages("ggplot2") # Para instalar el paquete "ggplot2"
```

* Tenga en cuenta que el nombre del paquete siempre debe ir entre comillas para la instalación.

- Compruebe si el paquete se ha instalado:

```
library()
```

abre una nueva pestaña en R escrita
"Paquetes R disponibles".

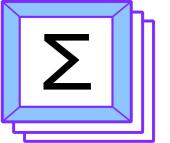


Paquetes

Instalación

- Para instalar paquetes de **Github**, usamos el paquete `devtools`:
`install_github("direccion/nombre_paquete")`.
- Para hacer esto, necesitaremos la dirección y el nombre del paquete de un repositorio de GitHub (<https://github.com/tidyverse/dplyr>)

```
# Instalar el paquete 'devtools'  
install.packages("devtools")  
  
# Cargar el paquete para su uso  
library(devtools)  
  
# Incluir la dirección de descarga  
# del paquete do github en install_github()  
install_github("tidyverse/dplyr")
```



Paquetes

Instalación

- Para instalar paquetes de **Github**, usamos el paquete **devtools**:

tidyverse/ggplot2: An implement x +

github.com/tidyverse/ggplot2

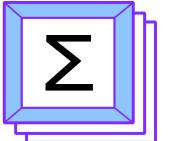
Library Genesis Localizaço lonomia Banco de dados Epidemiologia casamento Modelagem de nicho Outline - Read & a... GLMM_Lonomia

☰ README.md

ggplot2 is a system for declaratively creating graphics, based on [The Grammar of Graphics](#). You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:  
install.packages("tidyverse")  
  
# Alternatively, install just ggplot2:  
install.packages("ggplot2")  
  
# Or the development version from GitHub:  
# install.packages("devtools")  
devtools::install_github("tidyverse/ggplot2")
```



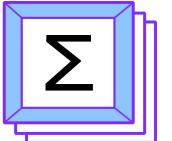
@somaquadrados

Paquetes

Instalación

- Solo instalamos los paquetes una vez.
- Los paquetes se descargan a través de la internet.
- El nombre del paquete debe estar entre comillas ("paquete_nombre"), independientemente de si lo vamos a descargar de [CRAN](#) o [GitHub](#).
- Para cargar paquetes en R usamos la función `library(paquete_nombre)`.
 - En este caso no es necesario incluir comillas.
 - Cargamos paquetes para usar sus funciones.

```
library(ggplot2)
library(dplyr)
```



@somaquadrados

Paquetes

Actualización

- Los paquetes no se actualizan solos.
- Es necesario actualizarlos de vez en cuando.
- ¡Es un proceso que lleva tiempo!

```
update.packages(ask = FALSE)
```

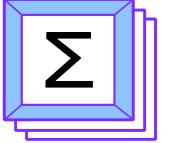
Dirección en mi compu

- ¿Dónde están los paquetes?
 - Windows: `C:/Users/nombre_del_compu/Documentación/R/win-library/versión_r`
 - Unix (Linux o MacOS): `/home/nombre_del_compu/R/tipo_compu/versión_r`

Paquetes

Dos paquetes útiles para empezar

- **tidyverse** es una colección obstinada de paquetes R diseñados para la ciencia de datos.
 - dplyr
 - ggplot2
 - forcats
 - tibble
 - readr
 - stringr
 - tidyr
 - purr
- **tidymodels** es una colección de paquetes R para modelado y aprendizaje automático utilizando principios tidyverse.
 - tidymodels
 - rsample
 - parsnip
 - recipes
 - tune
 - yardstick



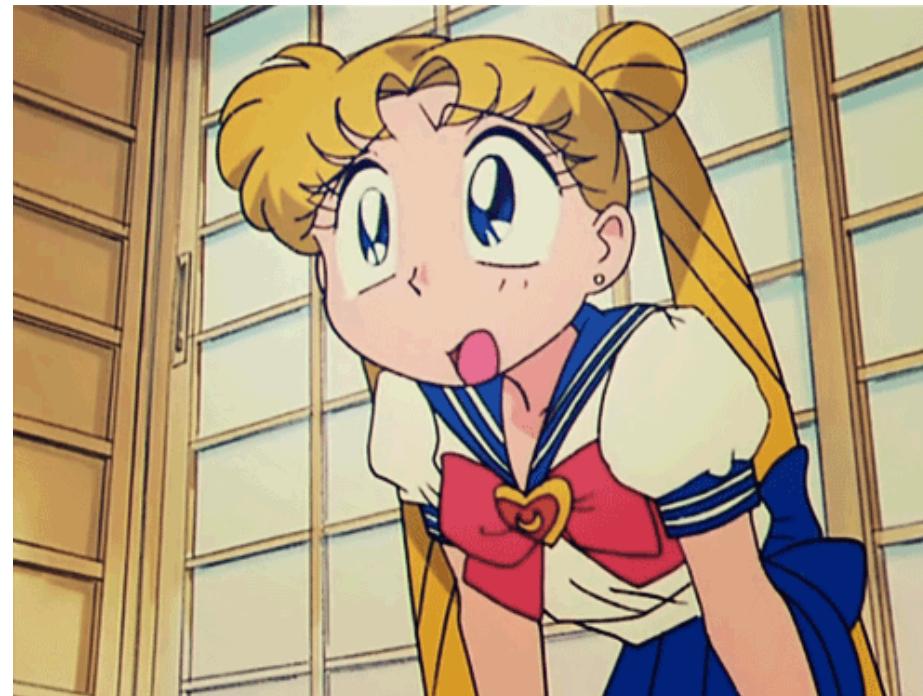
@somaquadrados

Paquetes

Cantidad de paquetes disponibles

```
nrow(available.packages(repos = "http://cran.r-project.org"))

## [1] 17779
```

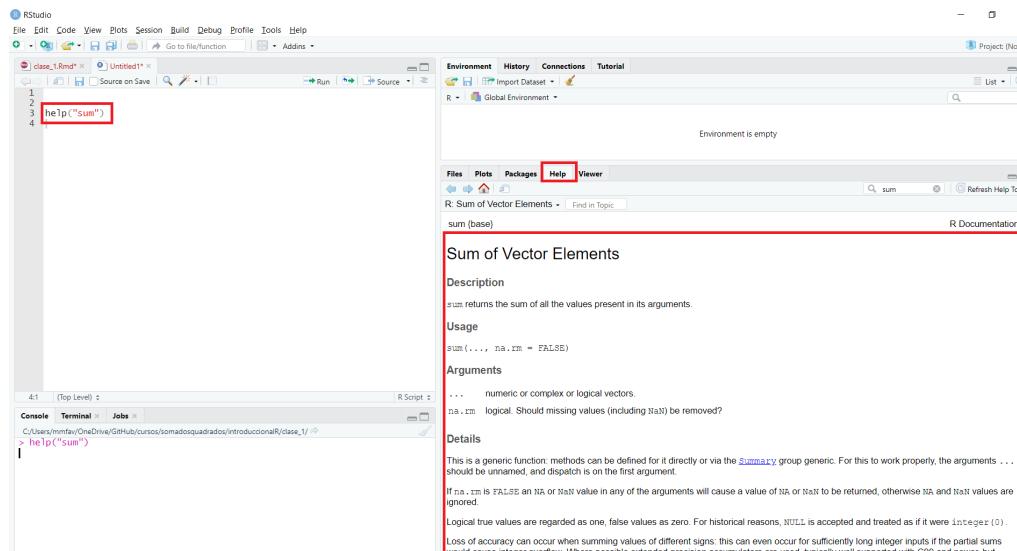


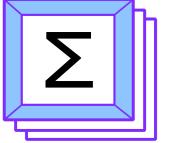
Paquetes

Help! (ayuda)

- El "help" de R es muy útil cuando necesitamos ayuda para comprender una función.

```
help("sum")
# es necesario encerrar el nombre de
# la función entre comillas.
```



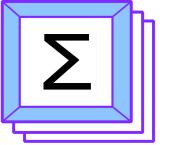


@somaquadrados

Paquetes

Help! (ayuda)

- *Description*: Una declaración sobre el propósito de la función.
- *Usage*: Muestra cómo debemos usar la función (parámetros y argumentos).
- *Arguments*: Explica lo que significa cada uno de los argumentos de la función.
- *Details*: Explica algunos detalles sobre el uso y la aplicación de la función.
- *Value*: La salida de la función (o resultados).
- *Note*: Notas de función.
- *Authors*: Los autores de la función.
- *References*: Las referencias utilizadas para desarrollar la función/método.
- *See also*: Otras funciones relacionadas que se pueden consultar en R help.
- *Examples*: Ejemplos de cómo utilizar la función

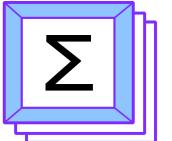


@somaquadrados

Paquetes

¿Cómo citar la R?

```
citation()  
  
##  
## To cite R in publications use:  
##  
##   R Core Team (2021). R: A language and environment for statistical computing. R  
##   Foundation for Statistical Computing, Vienna, Austria. URL  
##   https://www.R-project.org/.  
##  
## A BibTeX entry for LaTeX users is  
##  
## @Manual{,  
##   title = {R: A Language and Environment for Statistical Computing},  
##   author = {{R Core Team}},  
##   organization = {R Foundation for Statistical Computing},  
##   address = {Vienna, Austria},  
##   year = {2021},  
##   url = {https://www.R-project.org/},  
## }  
##
```



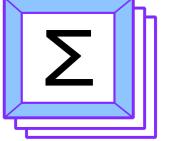
@somaquadrados

Paquetes

¿Cómo citar la R?

```
citation("ggplot2")

##
## To cite ggplot2 in publications, please use:
##
##   H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New
##   York, 2016.
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     author = {Hadley Wickham},
##     title = {ggplot2: Elegant Graphics for Data Analysis},
##     publisher = {Springer-Verlag New York},
##     year = {2016},
##     isbn = {978-3-319-24277-4},
##     url = {https://ggplot2.tidyverse.org},
##   }
```

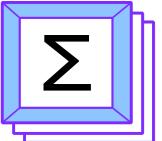


@somaquadrados

Paquetes

Ejercicios

Trabajarás con índices de disimilitud y para eso usarás el paquete "vegan" (CRAN). Instale el paquete, asegúrese de que esté instalado, cárguelo en R y obtenga su cita.



@somaquadrados

¡¡Fin de clase!!



Soma dos quadrados

-  [Soma-Dos-Quadrados/introductioR](#)
-  [/somaquadrados](#)
-  [/somadosquadrados](#)
-  [@somadosquadrados](#)

Marília Melo Favalesso

-  mariliabioufpr@gmail.com
-  [www.mmfava.com](#)
-  [/mmfava](#)