Unidad 2. Conceptos básicos de programación en R

Del Curso introductorio al lenguaje de programación R orientado al análisis cuantitativo en Ciencias Sociales por Sarahí Aguilar González

Objetivo: Que el estudiante reconozca los conceptos básicos de programación, cómo programarlos en R y sus casos de uso prácticos básicos.

Agenda

- Objetos
- Vectores
- Condicionales
- Iteraciones
- Funciones

Agenda

Objetos

Vectores

Condicionales

Iteraciones

Funciones

Una línea de código es efímera.
Para almacenar y manipular información, necesitamos **instanciar objetos***.

*Para los fines de este curso, un objeto también puede ser conocido como *una variable*.

Objetos

En R, **un objeto es un nombre específico que almacena un dato o un conjunto datos** y...

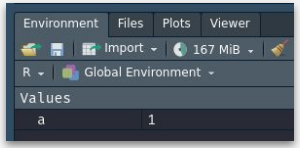
...se puede nombrar comenzando con una letra, y utilizando únicamente letras, números, guiones bajos (_) y puntos (.)

```
x
y
bdd
segmento_a
censo_2010
```

...se instancia utilizando un símbolo de menor que (<) seguido de un guión corto (-).

```
a <- 1
```

...después de ser instanciado, puedes observarlo en la lista de variables en la ventana de Ambiente en RStudio.



...se puede utilizar en nuevas líneas de código y su nombre será sustituído por el dato o conjunto de datos que almacenan.

```
b <- a + 4
```

b almacena el valor 5

...se sobrescribe sin aviso previo si son instanciados con el mismo nombre de variable.

```
b <- 42
```

b ahora almacena el valor 42

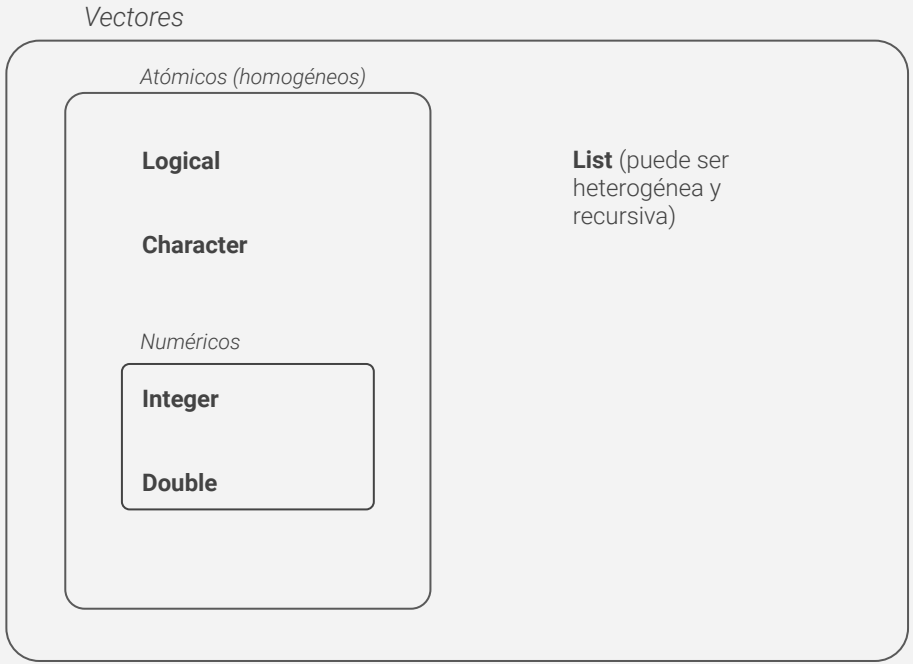
Agenda

- Objetos
- **Vectores**
- Condicionales
- Iteraciones
- Funciones

En R, los **bloques de construcción más básicos** son los **vectores**.

`c(` 1 `,` 2 `,` 3 `,` 4 `,` 5 `)`

Todos los **tipos de objetos en R** están construidos a partir de **vectores**.



NULL *(Ausencia de vector)*

Las 2 **propiedades** de un vector son:

Tipo

Tamaño

Además, pueden también contener **atributos adicionales** que los convierten en **vectores aumentados**.

Los **factores** están construidos *encima* de los vectores numéricos de tipo entero.
Las **fechas** están construidas *encima* de los vectores numéricos.
Los **data frames** están construidos *encima* de los vectores de tipo lista.

Los 4 tipos de **vectores atómicos**

Logical

Solo pueden tomar tres valores posibles: `FALSE`, `TRUE` y `NA`.

Se construyen con operadores lógicos o de comparación.

```
b <- TRUE

c <- c(FALSE, TRUE)
```

Character

Se componen por “strings”, es decir, cadenas de caracteres.

Se utilizan comillas para instanciarlos.

```
yo <- “Sarahí”

amigos <- c(“Ana”, “Pepe”)
```

Integer

Se componen de un único valor numérico.

Tienen un valor especial: `NA`

```
edad <- 24L

edades <- c(24L, NA)
```

Double

Se componen de la aproximación de un valor numérico.

Tienen varios valores especiales: `NA`, `NaN`, `Inf` and `-Inf`

```
radio <- sqrt(2) ^ 2

radios <- c(3/3, -Inf)
```

Las **listas**

Las listas son un paso adelante en la complejidad de los vectores atómicos, porque las listas pueden contener otras listas.

Esto las hace adecuadas para representar estructuras jerárquicas o en forma de árbol.

Las listas se crean con la función `list()`.

```
x1 <- list(c(1, 2), c(3, 4))
x2 <- list(list(1, 2), list(3, 4))
x3 <- list(1, list(2, list(3)))
```

Completa el siguiente diccionario de datos.

Campo	**Tipo de dato**	**Ejemplo**
Vivienda	Entero	15124
Hogar	Entero	3
Encuestado	Entero	2
UPM	Caracter	“124-2150-3”
Factor de expansión	Double	3.25
¿La semana pasada trabajó por lo menos una hora?	Logical	TRUE
¿Cuántas horas dedica a su trabajo principal a la semana?	Entero	40
¿Cuál es su ingreso mensual neto?	Caracter	“20-25”
¿Cuáles son las funciones que desempeñó en su trabajo anterior y cuáles son las funciones que desempeña en su trabajo actual?	Lista	list(list(“venta de frutas”), list(“venta de ropa”))
¿En qué fecha comenzó a buscar trabajo?	Entero	23456

Operaciones entre vectores

Ejecución por elemento.

`c(1, 2, 4, 6) * 2`

1 * 2

2 * 2

4 * 2

6 * 2

`c(1, 2, 4, 6) * c(2, 4)`

1 * 2

2 * 4

4 * 2

6 * 4

`c(1, 2, 4, 6) * c(3, 5, 8, 4)`

1 * 3

2 * 5

4 * 8

6 * 4

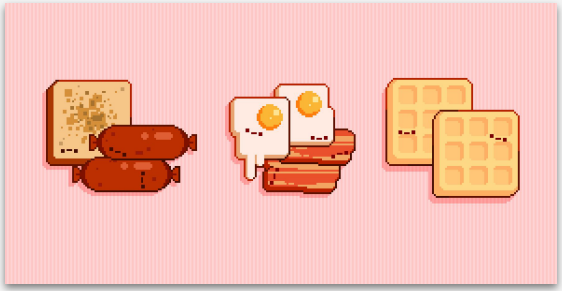
`c(1, 2, 4, 6) * c(3, 5, 8)`

```
Warning message:
In c(1, 2, 4, 6) * c(3, 5,
8) :
  longitud de objeto mayor
no es múltiplo de la
longitud de uno menor
```

Agenda

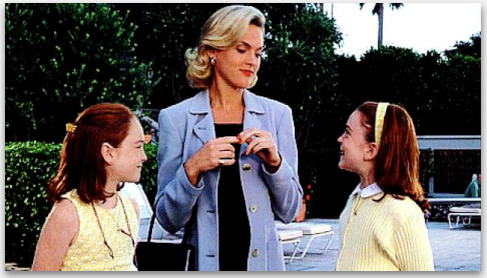
- Objetos
- Vectores
- **Condicionales**
- Iteraciones
- Funciones

¿Cómo decidieron qué desayunar hoy?



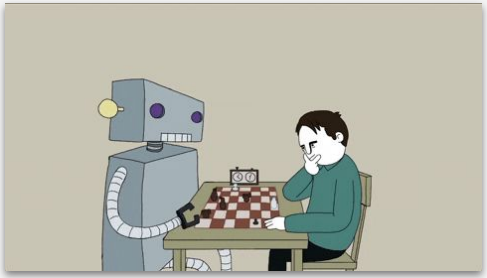
La toma de decisiones se basa en condiciones.

La toma de decisiones se basa en condiciones.
Y las condiciones se basan en comparaciones.



En “términos de programación”...

La toma de decisiones se basa en **instrucciones `if` y `else`**.
Y las condiciones se basan en **operadores de comparación y lógicos**.



Instrucción `if`

Indica a R **ejecutar** un bloque de código **si** se cumple una condición o un conjunto de condiciones.

Ejemplo figurativo:

```
if (this) {
  Plan A
}
```

La condición (`this`) será evaluada y devolverá un valor lógico.
Si la condición es verdadera (valor lógico `TRUE`), se ejecutará el bloque de código dentro de las llaves (*Plan A*).

Ejemplo con código de R:

```
if (TRUE) {
  num <- 1
}
```

Instrucción `if` y `else`

Indica a R ejecutar un bloque de código **si** se cumple una condición o un conjunto de condiciones y otro bloque de código distinto en el caso de que **no** se cumpla.

Ejemplo figurativo:

```
if (this) {
  Plan A
} else {
  Plan B
}
```

La condición (`this`) será evaluada y devolverá un valor lógico.
Si la condición es verdadera (valor lógico **`TRUE`**), se ejecutará el bloque de código dentro de las llaves (Plan A), pero si la condición es falsa (valor lógico **`FALSE`**), se ejecutará el bloque de código dentro de las segundas llaves (Plan B).

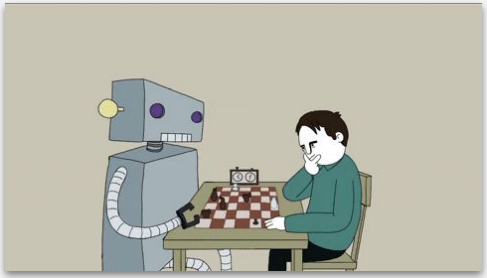
Ejemplo con código de R:

```
a <- 1
b <- 1

if (a > b) {
  a <- b
} else {
  b <- b + 1
}
```

En “términos de programación”...

La toma de decisiones se basa en **instrucciones `if` y `else`**.
Y las condiciones se basan en **operadores de comparación y lógicos**.



Operadores de comparación

funcionan entre valores

Descripción	Operador
más pequeño que	`<`
mayor que	`>`
más pequeño o igual que	`<=`
más grande o igual que	`>=`
igual que	`==`
diferente que	`!=`

Operadores lógicos

funcionan entre condiciones

Descripción	Operador
no es	`!`
y	`&`
o	`|`
en	`%in%`

Condicionales

¿Cuál es el valor de `a` después de ejecutar las siguientes líneas de código?

```
a <- 101

if (a <= 101) {
  a <- 100
}
```

¿Cuál es el valor de `a` después de ejecutar las siguientes líneas de código?

```
a <- 4
b <- 5

if (a != b) {
  c <- 6
}
```

¿Cuál es el valor de `b` después de ejecutar las siguientes líneas de código?

```
a <- 1
b <- 2

if (a > b) {
  a <- b
} else {
  b <- b + 1
}
```

¿Cuál es el valor de `a` después de ejecutar las siguientes líneas de código?

```
a <- "fem"

if (!a == "mas") {
  a <- "femenino"
}
```

¿Cuál es el valor de `a` después de ejecutar las siguientes líneas de código?

```
a <- 2
b <- 3

if (a+4 == 6 & a+b != 100) {
  a <- 7
}
```

¿Cuál es el valor de `a` después de ejecutar las siguientes líneas de código?

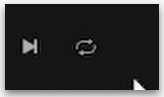
```
a <- c(5, 6, 8)

if (2 %in% a | 4  %in% a) {
  a <- 2
} else {
  b <- "other"
}
```

Agenda

- Objetos
- Vectores
- Condicionales
- **Iteraciones**
- Funciones

El verdadero poder de la programación es
ser capaz de realizar una misma tarea repetidas veces
sin necesidad de escribir la misma instrucción repetidas veces.



Las iteraciones te permiten repetir el mismo bloque de código un número determinado de veces.

Instrucción `for`

Indica a R **ejecutar** un bloque de código **n** ciclos, donde n es el tamaño de un vector.

⚠️ Dentro de un for, existe un ambiente cerrado.

Ejemplo figurativo:

```
for (valor in vector){
  Tareas a repetir
}
```

Se iterará (recorrerá por) cada uno de los valores del vector.
En cada ciclo de ejecución, se sobrescribe una variable (valor) con cada uno de los valores del vector.
“POR cada elemento EN un objeto, ejecuta el siguiente bloque de código.”

Ejemplo con código de R:

```
vec <- c(1, 2, 3)
a <- 2

for (i in vec) {
     a <- a + 2
}
```

Ejemplo con código de R:

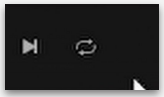
```
vec <- c(1, 2, 3)
a <- 2

for (i in vec) {
     a <- a + i
}
```

Agenda

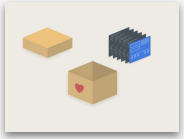
- Objetos
- Vectores
- Condicionales
- Iteraciones
- **Funciones** 

El verdadero poder de la programación es
ser capaz de realizar una misma tarea repetidas veces
sin necesidad de escribir la misma instrucción repetidas veces.



Las **iteraciones** te permiten repetir el mismo bloque de código **un número determinado de veces**.

El verdadero poder de la programación es
ser capaz de realizar una misma tarea repetidas veces
sin necesidad de escribir la misma instrucción repetidas veces.

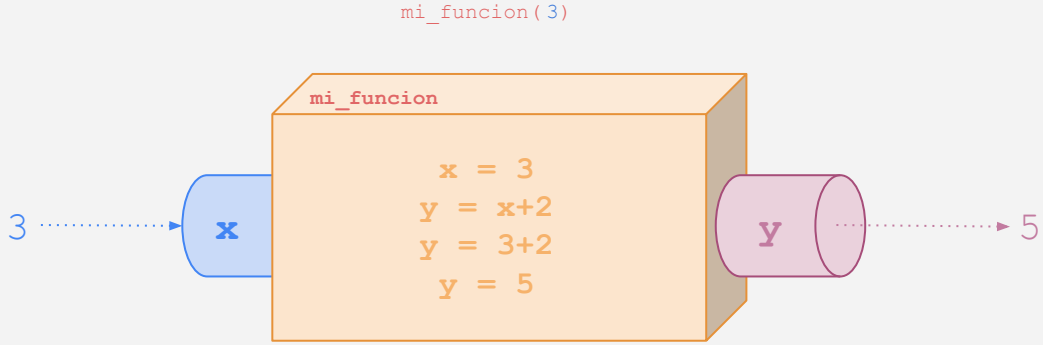


Las **funciones** te permiten repetir el mismo bloque de código **con uno o varios parámetros variables de entrada**.

Anatomía de una función

Las funciones en R se componen de **3 elementos principales**:

- Su **nombre**.
- Su **entrada** (parámetros o argumentos).
- Su **cuerpo** (bloque de código a ejecutar).
- Su **salida**.



Anatomía de una función

Las funciones en R se componen de **3 elementos principales**:

- Su **nombre**.
- Su **entrada** (parámetros o argumentos).
- Su **cuerpo** (bloque de código a ejecutar).
- Su **salida**.

⚠️ Dentro de una función, existe un ambiente cerrado.

Ejemplo figurativo:

Definición:

```
mi_func <- function(entrada) {
  Tareas a ejecutar
  return valor a devolver
}
```

Llamada:

```
mi_func(entrada)
```

Cada vez que se llame a la función, esta deberá estar seguida de paréntesis, y dentro de ellos, contener el valor de entrada.
El código dentro de las llaves se ejecutará entonces utilizando los parámetros de entrada.

Ejemplo con código de R:

Definición:

```
mi_func <- function(x) {
  y <- x + 2
  return y
}
```

Llamada:

```
mi_func(3)
```

(Devolverá el valor 5.)

Ventajas de las iteraciones y de las funciones

- Es más **fácil ver la intención de un bloque de código**, porque tus ojos se sienten atraídos por lo que es diferente, no por lo que permanece igual.
- Es más fácil responder a los cambios en los requisitos. A medida que cambian tus necesidades, **solo necesitas realizar cambios en un solo lugar**, en vez de recordar cambiar cada lugar donde copió y pegó el código.
- Es probable que tengas **menos errores**.

CÓDIGO MODULAR