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 y sus casos de uso prácticos básicos.
Duración: 1 sesión (2 horas)

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**.

Objetos

En R, **un objeto es un nombre específico que almacena uno 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

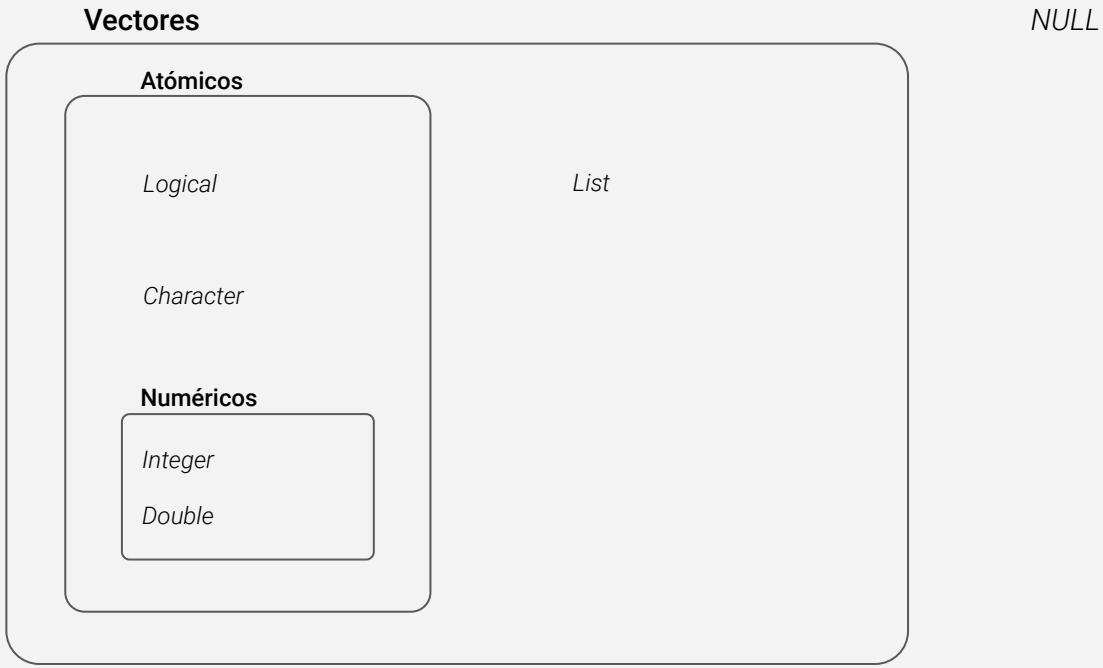
Agenda

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

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

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

Tipos de datos en R



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.

```
a <- "a" == "b"
```
a es igual a TRUE

```
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.

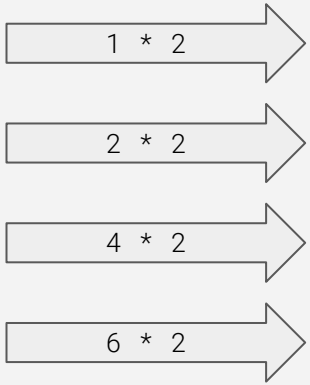
Las listas se crean con la función list().

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

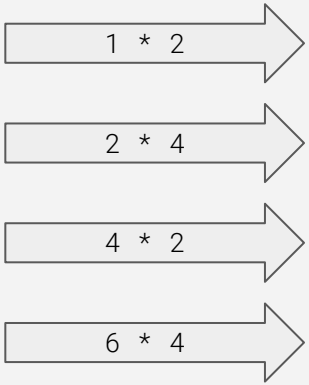
Operaciones entre vectores

Ejecución por elemento.

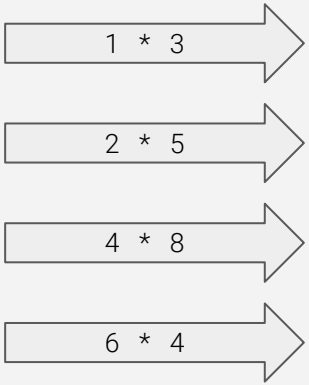
`c(1, 2, 4, 6) * 2`



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



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

Describan detalladamente su método de lavado de dientes.

La toma decisiones se basa en condiciones.

Si su primer apellido empieza con una vocal, reaccionen con el emoji de 👍.

Si su primer apellido empieza con una vocal, reaccionen con el emoji de 👍,
si no, reaccionen con el emoji de 👋.

Las condiciones se basan en comparaciones.

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	**|**

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 este es TRUE, se ejecutará el bloque de código dentro de las llaves (*Plan A*).

Ejemplo con código de R:

```
if (num < 0) {
  num <- num * -1
}
```

Instrucciones `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 este es `TRUE`, se ejecutará el bloque de código dentro de las primeras llaves (*Plan A*).
Si este es `FALSE`, se ejecutará el 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
}
```

Agenda

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

Describan detalladamente su método de lavado de dientes.

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.

Ventajas de las iteraciones

- 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**.

Instrucción `for`

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

Ejemplo figurativo:

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

Se recorrerán todos los valores del vector.
En cada ciclo, se sobrescribe una variable (valor) con cada uno de los valores del vector.
"PARA cada elemento EN un objeto, haz la siguiente operación."

Ejemplo con código de R:

```
a <- 1
for (i in c(1, 2, 3)) {
     a <- a + i
}
```

Ejemplo con código de R:

```
a <- 1
for (i in c(1, 2, 3)) {
     a <- a + 1
}
```

Instrucción `for`

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

⚠️ Dentro de una función, existe un ambiente cerrado.

Ejemplo figurativo:

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

Se recorrerán todos los valores del vector.
En cada ciclo, se sobrescribe una variable (valor) con cada uno de los valores del vector.
"PARA cada elemento EN un objeto, haz la siguiente operación."

Ejemplo con código de R:

```
a <- 1
for (i in c(1, 2, 3)) {
     a <- a + i
}
```

Ejemplo con código de R:

```
a <- 1
for (i in c(1, 2, 3)) {
     a <- a + 1
}
```

Agenda

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

Describan detalladamente su método de lavado de dientes.

Cepillar molar superior derecho con movimientos circulares.
Cepillar molar inferior derecho con movimientos circulares.
Cepillar molar superior izquierdo con movimientos circulares.
Cepillar molar inferior izquierdo con movimientos circulares.
Cepillar incisivo frontal superior derecho con movimientos verticales.
Cepillar incisivo frontal superior izquierdo con movimientos verticales.
Cepillar incisivo frontal inferior derecho con movimientos verticales.
Cepillar incisivo frontal inferior izquierdo con movimientos verticales.

Describan detalladamente su método de lavado de dientes.

Si el diente es un molar, cepillar con movimientos circulares.
Si el diente es un incisivo, cepillar con movimientos verticales.

Describan detalladamente su método de lavado de dientes.

Cepillar con base en tipo de diente y tipo de movimiento.

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 dependiendo de uno o varios parámetros.

Anatomía de una función

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

1. Su nombre.
2. Su entrada (argumentos o parámetros).
3. Su cuerpo (bloque de código a ejecutar).
4. Su salida.

⚠️ Dentro de una función, existe un ambiente cerrado.

Ejemplo figurativo:

```
nombre <- function(entrada) {
  Tareas a ejecutar
}

nombre(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.

Ejemplo con código de R:

```
a <- 1

sumar_2 <- function(x) {
  x + 2
}

sumar_2(a)
sumar_2(5)
```

CÓDIGO MODULAR