Intro a R (Studio)

Jorge Loría

Sept 4, 2017

R Studio :)

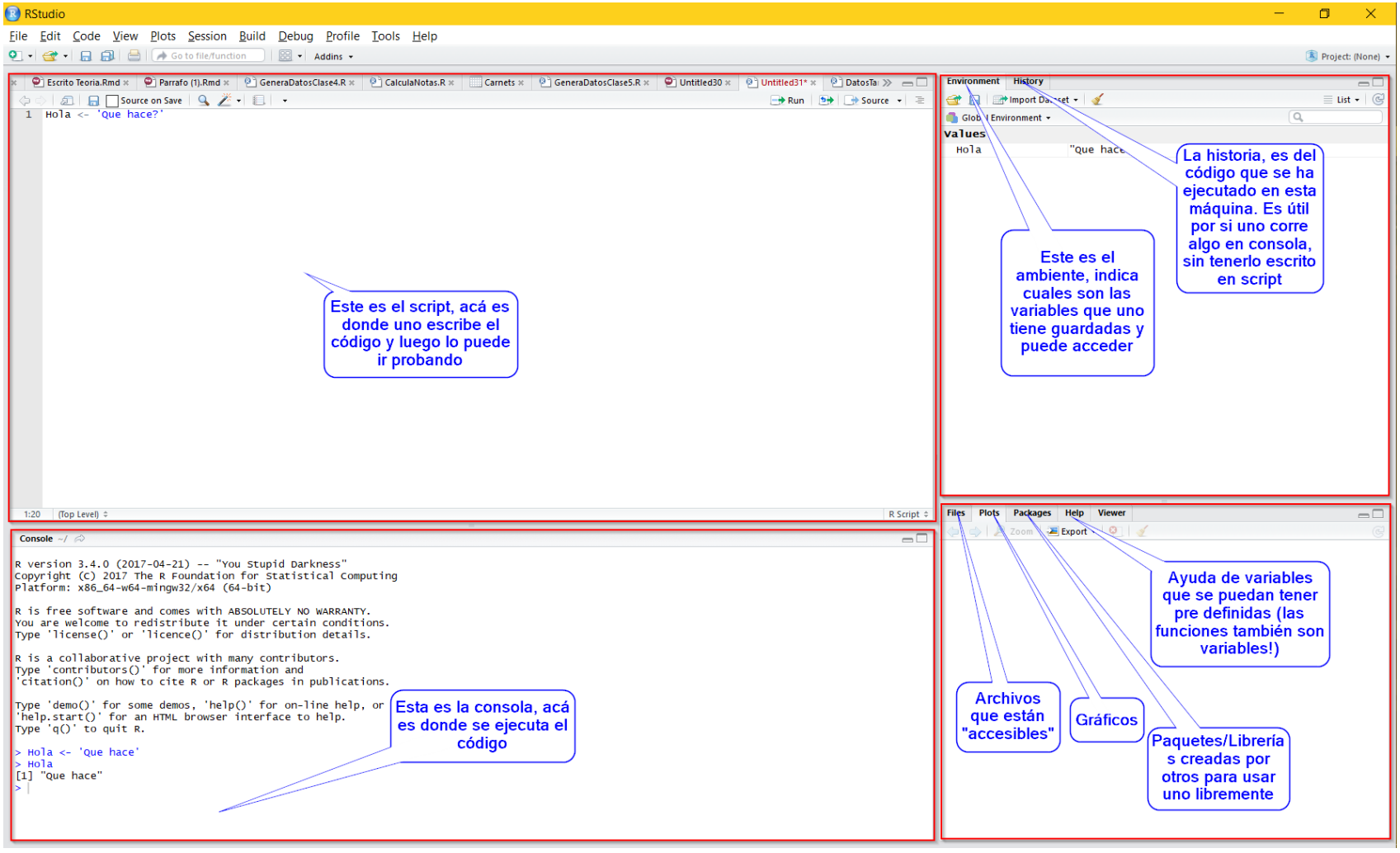

Figure 1: Interfaz de R Studio:

Tipos básicos de objetos en R

- Strings

```
String1 <- 'Los strings van entre comillas simples'
String2 <- "O entre comillas dobles"
String3 <- '4.4'
```

Tipos básicos de objetos en R

- Strings

```
String1 <- 'Los strings van entre comillas simples'
String2 <- "O entre comillas dobles"
String3 <- '4.4'
```

- Lógicos

```
Logico1 <- TRUE # También se puede declarar como T
Logico2 <- FALSE # También se puede declarar como F

Logico1 == Logico2
```

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

Tipos básicos de objetos en R

Tipos básicos de objetos en R

- Enteros

```
Entero1 <- 1L
Entero2 <- 20000L
Entero3 <- -6540L
Entero2 > Entero1 # Comparación de números
```

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

Tipos básicos de objetos en R

- Enteros

```
Entero1 <- 1L
Entero2 <- 20000L
Entero3 <- -6540L
Entero2 > Entero1 # Comparación de números
```

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

- Numéricos

```
numerico1 <- 3.14589
numerico2 <- -794.5135
numerico3 <- 1
Entero2 > numerico1
```

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

Revisar tipo, primeras funciones

Existe una serie de funciones en R para identificar el tipo del objeto que se tiene

Revisar tipo, primeras funciones

Existe una serie de funciones en R para identificar el tipo del objeto que se tiene

```
typeof(Entero1)
```

```
## [1] "integer"
```

```
typeof(numerico3)
```

```
## [1] "double"
```

```
Entero1 == numerico3
```

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

Revisar tipo, primeras funciones

```
is.numeric(String1)
```

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

```
is.character(String1)
```

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

```
is.logical(Logico1)
```

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

```
is.integer(Entero2)
```

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

Estructuras de datos

Hay 5 (ish) estructuras de datos básicos (como listas, pilas y objetos en *java*), estas son:

Estructuras de datos

Hay 5 (ish) estructuras de datos básicos (como listas, pilas y objetos en *java*), estas son:

	Homogeneas	No.Homogeneas
1-dim	Vectores	Listas
2-dim	Matriz	Data Frames
n-dim	Array	Listas*

Estructuras de datos

Hay 5 (ish) estructuras de datos básicos (como listas, pilas y objetos en *java*), estas son:

	Homogeneas	No.Homogeneas
1-dim	Vectores	Listas
2-dim	Matriz	Data Frames
n-dim	Array	Listas*

Intro a vectores

Para definir un vector, se usan los símbolos: c(), y adentro se ponen sus elementos separados por comas:

Intro a vectores

Para definir un vector, se usan los símbolos: c(), y adentro se ponen sus elementos separados por comas:

```
vector1 <- c(1L,2,79.97,987.4)
vector2 <- 1:15
vector1
```

```
## [1]   1.00   2.00  79.97 987.40
```

```
vector2
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
c(1,c(2,3)) == c(1,2,3)
```

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

```
c('Hola','Clase de intro','1.7')
```

Intro a vectores

Note que la comparación se hace elemento por elemento.

Una de las características principales de R (y de *MatLab* también) es que es un lenguaje vectorial. Por lo que hay funciones que funcionan sobre **todo** el vector, entrada por entrada:

Intro a vectores

Note que la comparación se hace elemento por elemento.

Una de las características principales de R (y de *MatLab* también) es que es un lenguaje vectorial. Por lo que hay funciones que funcionan sobre **todo** el vector, entrada por entrada:

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

```
## [1]  1  9 16 25
```

Intro a vectores

Note que la comparación se hace elemento por elemento.

Una de las características principales de R (y de *MatLab* también) es que es un lenguaje vectorial. Por lo que hay funciones que funcionan sobre **todo** el vector, entrada por entrada:

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

```
## [1]  1  9 16 25
```

Igual pasa con la suma, resta, y multiplicación:

```
v1 <- c(72.45, 45.86, -7.4)
v2 <- c(54.23, 6.1, 0.3246)
v1 + v2
v2 * v2
v1 - v2
```

¿Qué pasa si sumamos un vector de 2 entradas con uno de 3?

¿Qué pasa si sumamos un vector de 2 entradas con uno de 3? Note que R da una advertencia, pero igual realiza la operación. Esto puede ser o muy bueno, o muy malo. Por lo que hay que tener cuidado:

```
c(12.5,-1.4) + c(2,4,6)
```

```
## Warning in c(12.5, -1.4) + c(2, 4, 6): longer object len
## multiple of shorter object length
```

```
## [1] 14.5  2.6 18.5
```

¿Qué pasa si sumamos un vector de 2 entradas con uno de 3? Note que R da una advertencia, pero igual realiza la operación. Esto puede ser o muy bueno, o muy malo. Por lo que hay que tener cuidado:

```
c(12.5,-1.4) + c(2,4,6)
```

```
## Warning in c(12.5, -1.4) + c(2, 4, 6): longer object len
## multiple of shorter object length
```

```
## [1] 14.5  2.6 18.5
```

Los vectores también tienen tipo. Usando la misma función que antes, identifique el tipo de los vectores v1, v2 y vector1.

¿Qué pasa si sumamos un vector de 2 entradas con uno de 3? Note que R da una advertencia, pero igual realiza la operación. Esto puede ser o muy bueno, o muy malo. Por lo que hay que tener cuidado:

```
c(12.5,-1.4) + c(2,4,6)
```

```
## Warning in c(12.5, -1.4) + c(2, 4, 6): longer object len
## multiple of shorter object length
```

```
## [1] 14.5  2.6 18.5
```

Los vectores también tienen tipo. Usando la misma función que antes, identifique el tipo de los vectores v1, v2 y vector1.

Si definimos `s1 = c(String1,String2,String3)`, ¿qué tipo tiene?

¿Qué pasa si sumamos un vector de 2 entradas con uno de 3? Note que R da una advertencia, pero igual realiza la operación. Esto puede ser o muy bueno, o muy malo. Por lo que hay que tener cuidado:

```
c(12.5,-1.4) + c(2,4,6)
```

```
## Warning in c(12.5, -1.4) + c(2, 4, 6): longer object len
## multiple of shorter object length
```

```
## [1] 14.5  2.6 18.5
```

Los vectores también tienen tipo. Usando la misma función que antes, identifique el tipo de los vectores v1, v2 y vector1.

Si definimos `s1 = c(String1,String2,String3)`, ¿qué tipo tiene? Si lo hacemos con un vector de lógicos?

¿Qué pasa si sumamos un vector de 2 entradas con uno de 3? Note que R da una advertencia, pero igual realiza la operación. Esto puede ser o muy bueno, o muy malo. Por lo que hay que tener cuidado:

```
c(12.5,-1.4) + c(2,4,6)
```

```
## Warning in c(12.5, -1.4) + c(2, 4, 6): longer object len
## multiple of shorter object length
```

```
## [1] 14.5  2.6 18.5
```

Los vectores también tienen tipo. Usando la misma función que antes, identifique el tipo de los vectores v1, v2 y vector1.

Si definimos `s1 = c(String1,String2,String3)`, ¿qué tipo tiene? Si lo hacemos con un vector de lógicos? y con enteros?

Accesar vectores

Para accesar los valores de un vector se usan los paréntesis cuadrados `[]`. Por lo que si queremos accesar el primer valor de un vector, se debe usar:

Accesar vectores

Para accesar los valores de un vector se usan los paréntesis cuadrados `[]`. Por lo que si queremos accesar el primer valor de un vector, se debe usar: `nombreVector[1]`

```
vector1[1]
```

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

Accesar vectores

Para accesar los valores de un vector se usan los paréntesis cuadrados `[]`. Por lo que si queremos accesar el primer valor de un vector, se debe usar: `nombreVector[1]`

```
vector1[1]
```

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

Se puede usar un vector lógico para accesar las entradas de otro vector:

```
vector1[c(TRUE,FALSE,FALSE,TRUE)]
```

Accesar vectores

Para accesar los valores de un vector se usan los paréntesis cuadrados `[]`. Por lo que si queremos accesar el primer valor de un vector, se debe usar: `nombreVector[1]`

```
vector1[1]
```

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

Se puede usar un vector lógico para accesar las entradas de otro vector:

```
vector1[c(TRUE,FALSE,FALSE,TRUE)]
```

También se puede usar un vector de ubicaciones para que las accese:

```
vector1[1:3]
```

Vectores y escalares

Una de las mayores fortalezas de los lenguajes vectoriales es que las operaciones con escalares son muy *lindas*. Por lo que, lo da resultados entrada por entrada:

```
700*c(1,2,5,8)
2^c(-2,7,3,10)
(1/2)^c(-2,7,3,10)
```

Vectores y escalares

Una de las mayores fortalezas de los lenguajes vectoriales es que las operaciones con escalares son muy *lindas*. Por lo que, lo da resultados entrada por entrada:

```
700*c(1,2,5,8)
2^c(-2,7,3,10)
(1/2)^c(-2,7,3,10)
```

```
## [1]  700 1400 3500 5600
## [1]    0.25  128.00    8.00 1024.00
## [1] 4.0000000000 0.0078125000 0.1250000000 0.0009765625
```

Ejercicio:

Existe una función que se llama `sum` que recibe un vector como parámetro y suma todas sus entradas. Sabiendo esto, calcule lo siguiente:

$$\sum_{j=1}^{20} v^j$$

Con $i = 0.05$, y (como siempre) $v = \frac{1}{1+i}$.

Ejercicio:

Existe una función que se llama `sum` que recibe un vector como parámetro y suma todas sus entradas. Sabiendo esto, calcule lo siguiente:

$$\sum_{j=1}^{20} v^j$$

Con $i = 0.05$, y (como siempre) $v = \frac{1}{1+i}$.

Esto tiene varias partes: Primero:

```
i <- 0.05
v <- 1/(1+i)
```

Ejercicio:

Existe una función que se llama `sum` que recibe un vector como parámetro y suma todas sus entradas. Sabiendo esto, calcule lo siguiente:

$$\sum_{j=1}^{20} v^j$$

Con $i = 0.05$, y (como siempre) $v = \frac{1}{1+i}$.

Esto tiene varias partes: Primero:

```
i <- 0.05
v <- 1/(1+i)
```

Y luego para elevar el v a esas potencias se usa: `v^(1:20)`.

Ejercicio:

Existe una función que se llama `sum` que recibe un vector como parámetro y suma todas sus entradas. Sabiendo esto, calcule lo siguiente:

$$\sum_{j=1}^{20} v^j$$

Con $i = 0.05$, y (como siempre) $v = \frac{1}{1+i}$.

Esto tiene varias partes: Primero:

```
i <- 0.05
v <- 1/(1+i)
```

Y luego para elevar el v a esas potencias se usa: `v^(1:20)`. Para sumar, se usa `sum(v^(1:20))`.

Ejercicio:

Existe una función que se llama `sum` que recibe un vector como parámetro y suma todas sus entradas. Sabiendo esto, calcule lo siguiente:

$$\sum_{j=1}^{20} v^j$$

Con $i = 0.05$, y (como siempre) $v = \frac{1}{1+i}$.

Esto tiene varias partes: Primero:

```
i <- 0.05
v <- 1/(1+i)
```

Y luego para elevar el v a esas potencias se usa: `v^(1:20)`. Para sumar, se usa `sum(v^(1:20))`.Si se quiere hacer en una linea:

```
sum((1/(1+0.05))^(1:20))
```

```
## [1] 12.46221
```

Funciones de utilidad:

Función	Resultado
`prod`	producto
`sqrt`	raiz cuadrada
`mean`	media
`median`	mediana
`var`	varianza
`length`	longitud del vector
`min`	mínimo
`max`	máximo
`summary`	resumen del vector
`sort`	ordena
`range`	calcula el mínimo y máximo
`floor`	piso, entrada por entrada
`which`	cuales son las entradas ciertas
`exp`	exponencial
`log`	logaritmo
`cumsum`	suma acumulada

Listas

Las listas son uno de los objetos que más se utilizan en R, por su gran flexibilidad de tipos de datos que pueden contener. Inclusive una lista puede contener otra lista.

```
l1 <- list(1:3,'a',
           c(TRUE,FALSE,TRUE),
           c(13.731,67.89,-0.675),1)
```

Listas

Las listas son uno de los objetos que más se utilizan en R, por su gran flexibilidad de tipos de datos que pueden contener. Inclusive una lista puede contener otra lista.

```
l1 <- list(1:3,'a',
           c(TRUE,FALSE,TRUE),
           c(13.731,67.89,-0.675),1)
```

Las listas pueden tener nombres para sus elementos:

```
l2<- list(Nombre = c('Jorge','Loria'),
          Cedula = '1-1624-0508',
          Casillero = 20,
          lista_anterior = l1)
```

Accesar listas

Para accesar los objetos dentro de una lista se usan dos paréntesis cuadrados `[[]]`

```
l1[[1]]
l2[[1]]
l2[['Nombre']]
l2$Casillero
```

Accesar listas

Para accesar los objetos dentro de una lista se usan dos paréntesis cuadrados `[[]]`

```
l1[[1]]
l2[[1]]
l2[['Nombre']]
l2$Casillero
```

¿Cual es el output si llamamos `l1[1]`?

Accesar listas

Para accesar los objetos dentro de una lista se usan dos paréntesis cuadrados `[[]]`

```
l1[[1]]
l2[[1]]
l2[['Nombre']]
l2$Casillero
```

¿Cual es el output si llamamos `l1[1]`? ¿Qué tipo de objeto es?

Accesar listas

Para accesar los objetos dentro de una lista se usan dos paréntesis cuadrados `[[]]`

```
l1[[1]]
l2[[1]]
l2[['Nombre']]
l2$Casillero
```

¿Cual es el output si llamamos `l1[1]`? ¿Qué tipo de objeto es? Si hacemos la comparación == entre `l1[[1]]` y `l1[1]`, ¿qué se obtiene?

Accesar listas

Para accesar los objetos dentro de una lista se usan dos paréntesis cuadrados `[[]]`

```
l1[[1]]
l2[[1]]
l2[['Nombre']]
l2$Casillero
```

¿Cual es el output si llamamos `l1[1]`? ¿Qué tipo de objeto es? Si hacemos la comparación == entre `l1[[1]]` y `l1[1]`, ¿qué se obtiene? ¿Es igual llamar un objeto en `l2` usando `$` y usando `[[]]`?

Descripción de listas

Algunas veces los contenidos de las estructuras multidimensionales pueden ser un poco complicadas. Para esto existe una función que se llama `str`.

Descripción de listas

Algunas veces los contenidos de las estructuras multidimensionales pueden ser un poco complicadas. Para esto existe una función que se llama `str`. Intente llamar `str(l1)` y `str(l2)`.

Descripción de listas

Algunas veces los contenidos de las estructuras multidimensionales pueden ser un poco complicadas. Para esto existe una función que se llama `str`. Intente llamar `str(l1)` y `str(l2)`. ¿Qué pasa si llama `str(l2,1)`?

Descripción de listas

Algunas veces los contenidos de las estructuras multidimensionales pueden ser un poco complicadas. Para esto existe una función que se llama `str`. Intente llamar `str(l1)` y `str(l2)`. ¿Qué pasa si llama `str(l2,1)`?

También se pueden asignar nuevos elementos a una lista:

```
l2$Horario <- list(Lunes = 9:11,
                   Miercoles = 1:4)
l2$Edad <- 2017 - 1995
```

Descripción de listas

Algunas veces los contenidos de las estructuras multidimensionales pueden ser un poco complicadas. Para esto existe una función que se llama `str`. Intente llamar `str(l1)` y `str(l2)`. ¿Qué pasa si llama `str(l2,1)`?

También se pueden asignar nuevos elementos a una lista:

```
l2$Horario <- list(Lunes = 9:11,
                   Miercoles = 1:4)
l2$Edad <- 2017 - 1995
```

Ahora, llame de nuevo `str(l2)`

Descripción de listas

Algunas veces los contenidos de las estructuras multidimensionales pueden ser un poco complicadas. Para esto existe una función que se llama `str`. Intente llamar `str(l1)` y `str(l2)`. ¿Qué pasa si llama `str(l2,1)`?

También se pueden asignar nuevos elementos a una lista:

```
l2$Horario <- list(Lunes = 9:11,
                   Miercoles = 1:4)
l2$Edad <- 2017 - 1995
```

Ahora, llame de nuevo `str(l2)`

Si cambia elementos de una lista, se pierde el valor anterior

```
l2$Edad <- 2017-1995 - 1
```

purrr

Hay una librería dedicada al manejo *bonito* de las listas, que posiblemente vamos a ver más adelante en el curso.

Matrices

Matrices

Para implementar una matriz, se usa la función `matrix`

```
matrix(1:9,nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Matrices

Para implementar una matriz, se usa la función `matrix`

```
matrix(1:9,nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Para crear la matriz, se pueden utilizar varias entradas opcionales. Que son: *byrow*, *data*, *nrow*, *ncol* y *dimnames*.

Matrices

Para implementar una matriz, se usa la función `matrix`

```
matrix(1:9,nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Para crear la matriz, se pueden utilizar varias entradas opcionales. Que son: *byrow*, *data*, *nrow*, *ncol* y *dimnames*.

Sabiendo esto, cree la siguiente matrix:

```
##      [,1] [,2] [,3]  [,4]   [,5]
## [1,]  1.7    6   12  6.00 781.05
## [2,] 15.0  101 -140 -0.05 750.00
```

Matrices

Para implementar una matriz, se usa la función `matrix`

```
matrix(1:9,nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Para crear la matriz, se pueden utilizar varias entradas opcionales. Que son: *byrow*, *data*, *nrow*, *ncol* y *dimnames*.

Sabiendo esto, cree la siguiente matrix:

```
##      [,1] [,2] [,3]  [,4]   [,5]
## [1,]  1.7    6   12  6.00 781.05
## [2,] 15.0  101 -140 -0.05 750.00
```

Repita lo anterior, pero aumente ncol/nrow por un número.

También se pueden hacer matrices de strings, de elementos lógicos o de números:

```
mi_matriz1 <- matrix(c('Hola','Que','HAce','Hoy'),
                     nrow = 2,byrow= T)
mi_matriz2 <- matrix(c(TRUE,FALSE,TRUE,TRUE,TRUE,FALSE,
          FALSE,TRUE,FALSE,FALSE,TRUE,TRUE),
          ncol = 2)
mi_matriz3 <- matrix(1/c(1:20),nrow = 5)
```

También se pueden hacer matrices de strings, de elementos lógicos o de números:

```
mi_matriz1 <- matrix(c('Hola','Que','HAce','Hoy'),
                     nrow = 2,byrow= T)
mi_matriz2 <- matrix(c(TRUE,FALSE,TRUE,TRUE,TRUE,FALSE,
          FALSE,TRUE,FALSE,FALSE,TRUE,TRUE),
          ncol = 2)
mi_matriz3 <- matrix(1/c(1:20),nrow = 5)
```

Se accesan los elementos de una matriz usando dos coordenadas, separadas por una comma:

```
mi_matriz2[3,2] # Una entrada
mi_matriz2[1,1:2] # Para accesar un segmento de la matriz
mi_matriz2[1,] # Una fila
```

También se puede usar la función `str` para obtener información sobre una matriz. Sin embargo, muchas veces es más útil la función `dim`, para conocer las dimensiones de la matriz.

Arrays

La idea de un array es poder guardar –en una estructura– datos de dimensiones arbitrarias:

```
array(data = 1:24,dim = c(2,3,4))
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
```

Arrays

No cabe en la diapositiva anterior, y se pueden hacer del tamaño que se quiera:

```
a1<- array(data= 1:240,dim = c(2,3,4,5,2))
```

Al igual que en las otras estructuras se puede revisar si es de ese tipo usando la función `is.array`.