Vectores, matrices y listas

Igual que ocurre con otros lenguajes como Python, R no contiene arrays propiamente dichos, en el sentido de estructuras de datos de tamaño prefijado e inmutable. Lo que podremos tener alternativamente son *vectores* o *listas* de tamaño variable, así como matrices bidimensionales. Veremos cómo definirlos y algunas operaciones que se pueden hacer con ellos.

1. Vectores

Un vector es una colección de elementos del mismo tipo (aunque en algunos lenguajes se admite que sea de tipos distintos). Existen varias formas de definirlos en R:

- Usando la función `vector`. Podemos inicializarlo vacío o con un tamaño inicial definido, indicando el tipo de dato que contendrá. Los elementos se inicializan con valores por defecto (cero para números, *FALSE* para booleanos y cadenas vacías para textos)
- Alternativemente, podemos crear vectores de un tipo determinado usando sus funciones específicas, como `character` o `numeric`, por ejemplo.
- Si queremos especificar qué elementos van a formar parte del vector inicialmente, usaremos la función `c`. Es una forma bastante habitual de crear vectores.
- Podemos usar también el operador `:` y definir un rango de valores, que automáticamente se convierten en una secuencia o vector de valores. Alternativamente, la instrucción `seq` permite generar una secuencia desde un valor inicial (*from*) a uno final (*to*) con un incremento determinado (*by*)
- La instrucción `rep` genera un vector repitiendo un elemento una determinada cantidad de veces

Aquí vemos algunos ejemplos:

```
# Vector vacío
v1 <- vector()
# Vector con tamaño inicial de 10 para strings (se inicializan vacías)
v2 <- vector(mode="character", length=10)
# Vector de 5 números
v3 <- numeric(5)
# Vector con 3 datos numéricos: 1, 5 y 8
v4 <- c(1, 5, 8)
# Vector con los números del 1 al 5
v5 <- 1:5
# Vector con los números pares del 0 al 10 (ambos inclusive)
v6 <- seq(from=0, to=10, by=2)
# Vector con el número 1 repetido 5 veces
v7 <- rep(1, 5)
```

Comentábamos antes que, en R, los elementos de un vector deben ser del mismo tipo. Si intentamos definir un vector con elementos de distintos tipos R intentará convertirlos a un tipo común. Por ejemplo, en el siguiente caso intentamos crear un vector de textos y números, con lo que R convierte todo a texto:

```
v <- c("Hola", 23)  # "Hola" "23"
```

1.1. Operaciones sencillas con vectores

Podemos obtener el **número de elementos** de un vector con la función `length`. De hecho, podemos comprobar cómo las variables simples son intrínsecamente vectores de tamaño 1:

```
v <- c(1, 5, 8)
tam <- length(v)  # 3
tam2 <- length(8) # 1 (vector de tamaño 1)
```

La instrucción `str` nos da un **resumen de la estructura** de un vector: tipo de dato, dimensiones y contenido.

```
v <- c(1, 5, 8)
str(v)  # num [1:3] 1 5 8
```

Las **operaciones aritméticas** básicas trabajan elemento a elemento; podemos usarlas para sumar/restar/etc un valor a cada elemento del vector, o para operar elemento a elemento los componentes de un vector con los de otro. Ocurre algo similar con las **operaciones relacionales**

```
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
v3 <- v1 + 5     # v3 = 6, 7, 8
v4 <- v1 + v2    # v4 = 5, 7, 9
v1 > 2           # FALSE FALSE TRUE (se comparan todos con 2)
v1 > v2          # FALSE FALSE FALSE (se comparan uno a uno)
```

Para **elegir una subsecuencia** dentro de un vector, entre corchetes, indicamos el rango o rangos de casillas que nos interesan (en el caso de que haya varios rangos, los agrupamos en un vector con `c`):

```
v1 <- c(10, 20, 30, 40, 50, 60, 70, 80, 90)
v2 <- v1[1:3]            # 10 20 30
v3 <- v1[c(1:3, 5, 7:9)] # 10 20 30 50 70 80 90
```

Para **comprobar si un valor está contenido en el vector** usamos el operador `%in%`:

```
if (20 %in% v1) { ... }
```

A la hora de **añadir elementos al vector**, podemos usar la misma función `c`, indicando primero el vector existente y luego el nuevo elemento a añadir (añadir al final), o al revés (añadir al principio):

```
v <- c(1, 2, 3)
v2 <- c(v, 4)    # 1 2 3 4
```

Para **eliminar elementos del vector** indicamos entre corchetes el índice del número a eliminar, en negativo.

```
v <- c(1, 7, 3)
v2 <- v[-2]      # 1 3
```

> **NOTA**: observa que, a diferencia de otros lenguajes, en R las posiciones de los vectores comienzan a numerarse desde 1, en lugar de desde 0.

Para **ordenar** los elementos de un vector podemos usar la instrucción `sort`:

```
sort(v)
```

2. Matrices

Para **crear matrices bidimensionales** con sus filas y columnas podemos usar la función `matrix` en R. Disponemos de parámetros para indicar el número de filas y de columnas, además del conjunto de datos de entrada (un vector). Podemos también indicar el sentido en que se deben rellenar los datos de la matriz (por filas o por columnas). Veamos algunos ejemplos:

```
matriz <- matrix(data = c(1, 2, 3, 4, 5, 6),
    nrow=2, ncol=3, byrow=TRUE)
```

Esta instrucción genera una matriz 2x3 con esta distribución:

```
1 2 3
4 5 6
```

En cambio, si ponemos el último parámetro a *FALSE*, o lo omitimos, obtendríamos esta otra:

```
1 3 5
2 4 6
```

Para **acceder a los elementos** de la matriz usaremos entre la misma pareja de corchetes el indicador de fila y el de columna, separados por comas.

```
matriz <- matrix(data = c(1, 2, 3, 4, 5, 6),
    nrow=2, ncol=3, byrow=TRUE)
matriz[2, 2]   # Fila 2, columna 2 = 5
```

La instrucción `dim` obtiene las **dimensiones de la matriz**, en forma de vector:

```
dimensiones <- dim(matriz) # 2 3
dimensiones[1]  # 2
```

Esto podemos usarlo para redimensionar la matriz:

```
matriz <- matrix(data = c(1, 2, 3, 4, 5, 6),
    nrow=2, ncol=3, byrow=TRUE)
dim(matriz) <- c(3, 2)
```

Esto dejaría la matriz con 3 filas y 2 columnas, reajustando los elementos dentro.

```
1 5
4 3
2 6
```

La instrucción `cbind` permite **crear matrices combinando vectores** en columnas, de forma que el nombre de cada vector se pone como nombre de su propia columna. Alternativamente, la instrucción `rbind` hace lo mismo, pero distribuyéndolos en filas.

```
v1 <- 1:3
v2 <- 4:6
matriz1 <- cbind(v1, v2)
matriz2 <- rbind(v1, v2)
# También podemos combinar matrices con vectores
matriz3 <- cbind(matriz1, c(10:12))
```

Esto deja dos matrices *matriz1* y *matriz2* como estas:

```
     v1 v2
[1,]  1  4
[2,]  2  5
[3,]  3  6

   [1,] [2,] [3,]
v1    1    2    3
v2    4    5    6
```

Podemos **seleccionar un subrango** de una matriz indicando el rango de filas y columnas, separados por coma:

```
# Dos primeras filas y columnas
matriz[1:2, 1:2]
# Segunda columna, todas las filas
matriz[,2]
# Segunda fila, todas las columnas
matriz[2,]
# Primera y segunda filas, todas las columnas
matriz[1:2,]
```

3. Listas

Las listas son un tipo de colección en R distintas de los vectores. A diferencia de éstos, una lista puede tener elementos de distinto tipo. Podemos crearlas con la función `list`, indicando también los datos de inicio separados por comas:

```
l1 <- list(1, "a", TRUE)
```

Cada elemento dentro de la lista es en sí mismo un vector de tamaño 1 (comentábamos antes esta característica en R). También podemos tener datos complejos como componentes de la lista:

```
l2<- list(c(1, 2), "a", list(2, 4, 6))
```

Si usamos una pareja de corchetes para **acceder a un elemento de la lista**, lo que obtendremos es otra lista (sublista con el elemento seleccionado). Tendremos que indicar una segunda pareja de corchetes anidada para acceder al elemento en cuestión.

```
l1 <- list(1, "a", TRUE)
l1[2]    # Lista con el elemento "a"
l1[[2]]  # "a"
```

Alternativamente podemos asignar un nombre a cada dato, y acceder a ellos por una notación más compacta:

```
l1 <- list(uno=1, dos="a", tres=TRUE)
l1[["uno"]] # 1
l1$uno      # 1
```

Para **obtener el tamaño de la lista** podemos emplear la función `length` que ya hemos visto para los vectores:

```
l1 <- list(1, "a", TRUE)
tam <- length(l1) # 3
```

Para **aplicar una función a todos los elementos de una lista** usamos `lapply`, indicando la lista y la función a aplicar. Devuelve una lista con el resultado de aplicar la función a cada elemento de la lista original (una lista del mismo tamaño que la original).

```
# Obtiene la clase de cada elemento de l1
tipos <- lapply(l1, class)
```

Podemos también **convertir un dato a lista** con la instrucción `as.list`:

```
v <- c(1, 2, 3, 4)
l1 <- as.list(v)
```

> **Ejercicio 1**:

> Crea un programa llamado `buscaFrases.r` que le pida al usuario una secuencia de frases, hasta que escriba una cadena vacía. Las almacenará en un vector y luego le pedirá que busque un texto en el vector. El programa deberá mostrar las frases que contengan el texto escrito por el usuario, extraídas del vector de frases.
>
> **AYUDA:** vídeo con solución del ejercicio