

Estructuras de datos R

Tipos de datos

Un vector es una secuencia ordenada de datos. R dispone de los siguientes tipos de datos:

- **logical**: (True or False)
- **integer**: \mathbb{Z}
- **numeric**: \mathbb{R}
- **complex**: \mathbb{C}

En los vectores de R, todos sus objetos han de ser del mismo tipo: todos números, todas palabras, etc. Cuando queramos usar vectores formados por objetos de diferentes tipos, tendremos que usar **listas generalizadas**, **lists** que veremos al final del tema.

Creaciones básicas

- **c()**: crear un vector
- **scan(0)**: definir un vector
- **fix(x)**: modificar visualmente un vector **x**
- **rep(a,n)**: crear un vector que contiene el dato *a* repetido *n* veces

Ejemplo

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

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

```
rep("Mates",7)
```

```
[1] "Mates" "Mates" "Mates" "Mates" "Mates" "Mates" "Mates"
```

Progresiones y secuencias

Una progresión aritmética es una sucesión de números tales que la **diferencia**, *d*, de cualquier par de términos sucesivos de la secuencia es constante.

$$a_n = a_1 + (n - 1) \cdot d$$

- **seq(a,b,by=d)**: para generar una progresión aritmética de diferencia *d* que empieza en *a* y termina en *b*

```
seq(5, 60, by=5)
```

```
[1] 5 10 15 20 25 30 35 40 45 50 55 60
```

- **seq(a,b,length.out=n)**: define una progresión aritmética de longitud n que va de a a b con diferencia $d = \frac{b-a}{n-1}$. (n es la cantidad de elementos del array)

```
seq(4, 35, length.out=7)
```

```
[1] 4.000000 9.166667 14.333333 19.500000 24.666667 29.833333 35.000000
```

- **seq(a,by=d, length.out=n)**: define la progresión aritmética de longitud n y diferencia d que empieza en a

```
seq(4, by=25, length.out=3)
```

```
[1] 4 29 54
```

- **a:b**: define la secuencia de números **enteros** (\mathbb{Z}) consecutivos entre dos números a y b

```
1:9
```

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

Funciones

Cuando queremos aplicar una función a cada uno de los elementos de un vector de datos, la función **sapply** nos ahorra tener que programar con bucles en R:

- **sapply(nombre_del_vector, FUN=nombre_de_la_función)**: para aplicar dicha función a todos los elementos del vector

```
x = 1:10  
sapply(x, FUN=function(x){sqrt(x)})
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427  
[9] 3.000000 3.162278
```

- **sqrt(x)**: calcula un nuevo vector con las raíces cuadradas de cada uno de los elementos del vector x

```
x = 1:10  
x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
x + pi
```

```
[1] 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593 10.141593
[8] 11.141593 12.141593 13.141593
```

```
x * pi
```

```
[1] 3.141593 6.283185 9.424778 12.566371 15.707963 18.849556 21.991149
[8] 25.132741 28.274334 31.415927
```

```
sqrt(x)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[9] 3.000000 3.162278
```

```
2^x
```

```
[1] 2 4 8 16 32 64 128 256 512 1024
```

```
x^2
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

- **mean(x)**: calcula la media aritmética de las entradas del vector x

```
mean(1:10)
```

```
[1] 5.5
```

- **diff(x)**: cañcula el vector formado por las diferencias sucesivas entre entradas del vector original

```
diff(1:10)
```

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

- **cumsum(x)**: calcula el vector formado por las sumas acumuladas de las entradas del vector original x
 - Permite definir sucesiones descritas mediante sumatorios
 - Cada entrada de $\text{cumsum}(x)$ es la suma de las entradas de x hasta su posición

```
cumsum(1:10)
```

```
[1] 1 3 6 10 15 21 28 36 45 55
```

Orden

- **sort(x)**: ordena el vector en orden natural de los objetos que lo forman: el orden numérico creciente, orden alfabético. . .
- **rev(x)**: invierte el orden de los elementos del vector x

```
v = c(1,7,5,2,4,6,3)
sort(v)
```

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

```
rev(v)
```

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

Otros

- **length(x)** : longitud del vector
- **max(x)** : elemento máximo del vector
- **min(x)** : mínimo del vector
- **sum(x)** : suma de todos los elementos del vector
- **prod(x)** : producto de todos los elementos del vector

```
x = 1:10
length(x)
```

```
[1] 10
```

```
max(x)
```

```
[1] 10
```

```
min(x)
```

```
[1] 1
```

```
sum(x)
```

```
[1] 55
```

```
prod(x)
```

```
[1] 3628800
```

Subvectores

- **Vector[i]**: da la i -ésima entrada del vector
 - Los índices en R empiezan en 1
 - **vector(length(vector))** nos da la última entrada del vector
 - **vector[a:b]**: si a y b son naturales, nos da el subvector con las entradas del vector original que van de la posición a -ésima hasta la b -ésima

- **vector[-i]**: si i es un número, este subvector está formado por todas las entradas del vector original menos la entrada i -ésima. Si i resulta ser un vector, entonces es un vector de índices y crea un nuevo vector con las entradas del vector original, cuyos índices pertenecen a i
- **vector[-x]**: si x es un vector (de índices), entonces este es el complementario de **vector[x]**
- También podemos utilizar operadores lógicos:
 - **==**: =
 - **!=**: ≠
 - **>=**: ≥
 - **<=**: ≤
 - **<**: <
 - **>**: >
 - **!**: NO lógico
 - **&**: Y lógico
 - **|**: O lógico

```
v = c(14,5,6,19,32,0,8)
v[2]
```

```
[1] 5
```

```
v[-c(3,5)] # no aparecen los elementos de las posiciones 3 y 5
```

```
[1] 14 5 19 0 8
```

```
v[v != 19 & v > 15] # elementos que no son 19 y mayores a 15
```

```
[1] 32
```

Condicionales

- **which(x cumple condición)**: para obtener los índices de las entradas del vector x que satisfacen la condición dada
- **which.min(x)**: nos da la primera posición en la que el vector x toma su valor mínimo
- **which(x==min(x))**: da todas las posiciones en las que el vector x toma sus valores mínimos
- **which.max(x)**: nos da la primera posición en la que el vector x toma su valor máximo
- **which(x==max(x))**: da todas las posiciones en las que el vector x toma sus valores máximos

Ejemplos

```
x = seq(3, 50, by=3.5)
x
```

```
[1] 3.0 6.5 10.0 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0 48.5
```

```
x[3]
```

```
[1] 10
```

```
x[length(x)]
```

```
[1] 48.5
```

```
x[length(x)-2]
```

```
[1] 41.5
```

```
x[-3] # Se quita la tercer entrada
```

```
[1] 3.0 6.5 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0 48.5
```

```
x[8:4] # da de la posición 8 a la 4
```

```
[1] 27.5 24.0 20.5 17.0 13.5
```

```
x[seq(2,length(x), by=2)] # los de posición par
```

```
[1] 6.5 13.5 20.5 27.5 34.5 41.5 48.5
```

```
x[-seq(2,length(x), by=2)] # quitar los de posición par
```

```
[1] 3 10 17 24 31 38 45
```

```
x[c(1,5,6)] #Da los de posición 1, 5 y 6
```

```
[1] 3.0 17.0 20.5
```

```
# Da los elementos que son mayores a 5 (x>5 da una lista booleana de los elementos de x que  
# son mayores 5)  
x[x>5]
```

```
[1] 6.5 10.0 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0 48.5
```

```
x>5
```

```
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[13] TRUE TRUE
```

```
x = c(6,7,4,2,4,8,9,2,9)  
y = c(5,2,-3,5,-1,4-2,7,1)  
# y%%2==0 da las posiciones de y cuyos valores son pares y tomamos esas posiciones y  
x[y%%2==0]
```

```
[1] 7 8
```

```
which(x>4) # la las posiciones de los elementos de x que son mayores que 4
```

```
[1] 1 2 6 7 9
```

```
which.max(x) # Da la primera posición del elemento más grande en x
```

```
[1] 7
```

```
which(x == max(x)) # Da todas las posiciones del elemento máximo de x
```

```
[1] 7 9
```

```
x[x<0] # es vacío
```

```
numeric(0)
```

```
which(x<0) #En posiciones
```

```
integer(0)
```

Valores NA

```
x = seq(1, 10, by=1)
```

```
x[11] # no existe elemento en la posición 11
```

```
[1] NA
```

```
x[11]=11 # se le agrega un elemento en la posición 11
```

```
x
```

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

```
x[15]=15 # Se le agrega un elemento en la posición 15
```

```
x # se rellena con NA
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 NA NA NA 15
```

```
mean(x) # no puede operar porque tiene valores NA
```

```
[1] NA
```

```
mean(x, na.rm = TRUE) # quita los elementos NA
```

```
[1] 6.75
```

```
which(is.na(x)) # Da las posiciones de los elementos NA
```

```
[1] 12 13 14
```

```
which(x==NA) # Da vacío
```

```
integer(0)
```

```
y=x  
y[is.na(y)] = mean(y,na.rm = TRUE) # Cambiamos los NA por la media  
y
```

```
[1] 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00 9.00 10.00 11.00 6.75  
[13] 6.75 6.75 15.00
```

```
x[!is.na(x)] # da los elementos que no son NA
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 15
```

```
cumsum(x[!is.na(x)]) #Para funciones que no admiten na.rm
```

```
[1] 1 3 6 10 15 21 28 36 45 55 66 81
```

```
y=na.omit(x) # Se quitan los NA  
cumsum(y)
```

```
[1] 1 3 6 10 15 21 28 36 45 55 66 81
```

```
y # tiene atributos "na.action" y "class"
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 15  
attr("na.action")  
[1] 12 13 14  
attr("class")  
[1] "omit"
```

```
attr(y,"na.action")= NULL # se quita el atributo  
attr(y,"class")= NULL # se quita el atributo  
y
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 15
```

Factor

Factor: es como un vector, pero con una estructura interna más rica que permite usarlo para clasificar observaciones

- **levels:** atributo del factor. Cada elemento del factor es igual a un nivel. Los niveles clasifican las entradas del factor. Se ordenan por orden alfabético
- Para definir un factor, primero hemos de definir un vector y trasformarlo por medio de una de las funciones `factor()` o `as.factor()`.

La función factor()

- `factor(vector, levels=...)`: define un factor a partir del vector y dispone de algunos parámetros que permiten modificar el factor que se crea:
 - `levels`: permite especificar los niveles e incluso añadir niveles que no aparecen en el vector
 - `labels`: permite cambiar los nombres de los niveles
- `levels(factor)`: para obtener los niveles del factor

```
nombres = c("Juan", "Antonio", "Ricardo", "Juan", "Juan", "María", "María")
nombres
```

```
[1] "Juan"      "Antonio" "Ricardo" "Juan"      "Juan"      "María"     "María"
```

```
nombres.factor = factor(nombres)
nombres.factor
```

```
[1] Juan      Antonio Ricardo Juan      Juan      María     María
Levels: Antonio Juan María Ricardo
```

Factor ordenado

Factor ordenado. Es un factor donde los niveles siguen un orden

- `ordered(vector, levels=...)`: función que define un factor ordenado y tiene los mismos parámetros que factor