

Vectores y tipos de datos

Imanol

10/2/2021

Estructuras de datos

Basicos

Vectores

```
c(1,2,3) # Para definir un vector
```

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

```
scan() # Para definir un vector por consola a mano
```

```
## numeric(0)
```

```
##fix(c): # Para modificar visualmente un vector x
```

```
rep(1,5) # Para definir un vector que tiene el mismo dato a (1) repetido n (5) veces
```

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

Para especificar como colocar los decimales -> scan(dec = “,”) Para especificar que tipo de datos seran -> scan(what = “character”)

Ejercicio

Repetir mi año de nacimiento 10 veces

```
anno <- 1993
```

```
n <- 10
```

```
rep(anno,n)
```

```
## [1] 1993 1993 1993 1993 1993 1993 1993 1993 1993 1993
```

Crear un vector que tenga como entradas 16,0,1,20,1,7,88,5,1,9, llamalo **vec** y modifica la cuarta entrada con la funcion **fix()**.

```
vec <- c(16, 0, 1, 20, 1, 7, 88, 5, 1, 9)
vec
```

```
## [1] 16 0 1 20 1 7 88 5 1 9
```

```
fix(vec)
vec
```

```
## [1] 16 0 1 20 1 7 88 5 1 9
```

Mas operaciones con vectores

Operaciones directas aplicadas a un vector

```
x <- 1:10
x + pi # Suma pi a todos
```

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

```
pi*x # Multiplica a todos por 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) # Raiz cuadrada de cada valor
```

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

```
2^x # 2 elevado a cada valor
```

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

```
x^2 # cada valor elevado a dos
```

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

```
1:10 + 1:10 #Se pueden sumar vectores de la misma longitud
```

```
## [1] 2 4 6 8 10 12 14 16 18 20
```

```
1:10 * 1:10 #Se pueden multiplicar vectores de la misma longitud
```

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

```
n = 1:100
x = 2*3^(n/2)-15 # Esta formula calcularia los n elemntos de esta sucesion
x
```

```
## [1] -1.153590e+01 -9.000000e+00 -4.607695e+00 3.000000e+00 1.617691e+01
## [6] 3.900000e+01 7.853074e+01 1.470000e+02 2.655922e+02 4.710000e+02
## [11] 8.267767e+02 1.443000e+03 2.510330e+03 4.359000e+03 7.560990e+03
## [16] 1.310700e+04 2.271297e+04 3.935100e+04 6.816891e+04 1.180830e+05
## [21] 2.045367e+05 3.542790e+05 6.136402e+05 1.062867e+06 1.840951e+06
## [26] 3.188631e+06 5.522882e+06 9.565923e+06 1.656868e+07 2.869780e+07
## [31] 4.970606e+07 8.609343e+07 1.491182e+08 2.582803e+08 4.473546e+08
## [36] 7.748410e+08 1.342064e+09 2.324523e+09 4.026192e+09 6.973569e+09
## [41] 1.207858e+10 2.092071e+10 3.623573e+10 6.276212e+10 1.087072e+11
## [46] 1.882864e+11 3.261215e+11 5.648591e+11 9.783646e+11 1.694577e+12
## [51] 2.935094e+12 5.083732e+12 8.805282e+12 1.525119e+13 2.641584e+13
## [56] 4.575358e+13 7.924753e+13 1.372608e+14 2.377426e+14 4.117823e+14
## [61] 7.132278e+14 1.235347e+15 2.139683e+15 3.706040e+15 6.419050e+15
## [66] 1.111812e+16 1.925715e+16 3.335436e+16 5.777145e+16 1.000631e+17
## [71] 1.733144e+17 3.001893e+17 5.199431e+17 9.005678e+17 1.559829e+18
## [76] 2.701703e+18 4.679488e+18 8.105110e+18 1.403846e+19 2.431533e+19
## [81] 4.211539e+19 7.294599e+19 1.263462e+20 2.188380e+20 3.790385e+20
## [86] 6.565139e+20 1.137115e+21 1.969542e+21 3.411346e+21 5.908625e+21
## [91] 1.023404e+22 1.772588e+22 3.070212e+22 5.317763e+22 9.210635e+22
## [96] 1.595329e+23 2.763191e+23 4.785987e+23 8.289572e+23 1.435796e+24
```

Operaciones que no se pueden hacer directamente a un vector (uso de sapply)

Ejemplo básico:

```
sapply(x, FUN = function(elemento){sqrt(elemento)}) # A x le aplicamos la función FUN
```

```
## Warning in sqrt(elemento): Se han producido NaNs
```

```
## Warning in sqrt(elemento): Se han producido NaNs
```

```
## Warning in sqrt(elemento): Se han producido NaNs
```

```
## [1] NaN NaN NaN 1.732051e+00 4.022054e+00
## [6] 6.244998e+00 8.861757e+00 1.212436e+01 1.629700e+01 2.170253e+01
## [11] 2.875372e+01 3.798684e+01 5.010319e+01 6.602272e+01 8.695395e+01
## [16] 1.144858e+02 1.507082e+02 1.983709e+02 2.610918e+02 3.436321e+02
## [21] 4.522574e+02 5.952134e+02 7.833519e+02 1.030954e+03 1.356816e+03
## [26] 1.785674e+03 2.350081e+03 3.092883e+03 4.070464e+03 5.357033e+03
## [31] 7.050252e+03 9.278654e+03 1.221140e+04 1.607110e+04 2.115076e+04
## [36] 2.783597e+04 3.663419e+04 4.821331e+04 6.345228e+04 8.350790e+04
## [41] 1.099026e+05 1.446399e+05 1.903568e+05 2.505237e+05 3.297077e+05
## [46] 4.339198e+05 5.710705e+05 7.515711e+05 9.891232e+05 1.301759e+06
## [51] 1.713212e+06 2.254713e+06 2.967369e+06 3.905278e+06 5.139635e+06
## [56] 6.764140e+06 8.902108e+06 1.171583e+07 1.541890e+07 2.029242e+07
## [61] 2.670633e+07 3.514750e+07 4.625671e+07 6.087726e+07 8.011898e+07
## [66] 1.054425e+08 1.387701e+08 1.826318e+08 2.403569e+08 3.163275e+08
## [71] 4.163104e+08 5.478953e+08 7.210708e+08 9.489825e+08 1.248931e+09
```

```
## [76] 1.643686e+09 2.163212e+09 2.846948e+09 3.746794e+09 4.931058e+09
## [81] 6.489637e+09 8.540843e+09 1.124038e+10 1.479317e+10 1.946891e+10
## [86] 2.562253e+10 3.372114e+10 4.437952e+10 5.840673e+10 7.686758e+10
## [91] 1.011634e+11 1.331386e+11 1.752202e+11 2.306028e+11 3.034903e+11
## [96] 3.994157e+11 5.256606e+11 6.918083e+11 9.104709e+11 1.198247e+12
```

Ejemplo más complicado:

```
cd = function(x){summary(lm((1:4)~c(1:3,x))}$r.squared # Coeficiente de determinación, muy avanzado por
cd(5)
```

```
## [1] 0.9657143
```

```
cd(6)
```

```
## [1] 0.9142857
```

```
cd(7)
```

```
## [1] 0.8698795
```

Más funciones

```
v = c(1,2,3,4,5,6)
mean(v) # media
```

```
## [1] 3.5
```

```
cumsum(v) # suma acumulada
```

```
## [1] 1 3 6 10 15 21
```

```
sort(v) # Ordena el vector de forma creciente
```

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

```
rev(v) # Invierte el vector
```

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

```
length(v) # Longitud del vector
```

```
## [1] 6
```

```
max(v) # EL mas grande
```

```
## [1] 6
```

```
min(v) # El mas pequeño
```

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

```
sum(v) # La suma de todos
```

```
## [1] 21
```

```
prod(v) # El producto de todos
```

```
## [1] 720
```

```
cumsum(v) # La suma acumulada
```

```
## [1] 1 3 6 10 15 21
```

```
cummax(v) # EL valor mas grande encontrado hasta el momento
```

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

```
cummin(v) # El mas pequeño encontrado hasta el momento
```

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

```
cumprod(v) # El producto acumulado
```

```
## [1] 1 2 6 24 120 720
```

```
diff(v) # La diferencia de uno a otro numero
```

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

Ejercicio

Combinad las dos funciones, **sort** y **rev** para crear una funcion que dado un vector x os lo devuelva ordenado en orden decreciente.

```
x <- scan()  
x <- sort(x)  
x <- rev(x)  
x
```

```
## numeric(0)
```

```
# Otro metodo más corto
y <- c(2,4,3,1,7,8,5,9,6,0)
y <- sort(y, decreasing = TRUE) # Al decirle decreasing TRUE hago que lo ordene en orden decreciente
y
```

```
## [1] 9 8 7 6 5 4 3 2 1 0
```

SUBVECTORES Y FILTROS

```
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] # Para sacar el dato de una posicion
```

```
## [1] 10
```

```
x[length(x)] # Para ir al ultimo valor
```

```
## [1] 48.5
```

```
x[length(x)-1] # Para ir al penultimo valor
```

```
## [1] 45
```

```
x[-3] # Para quitar el valor de esa posicion
```

```
## [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[4:8] # Para quedarme solo en el rango que quiero de posiciones
```

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

```
x[8:4]
```

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

```
x[seq(2, length(x), by = 2)] # Para coger solo las posiciones pares
```

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

```
x[seq(1, length(x), by = 2)] # Para coger solo las posiciones impares
```

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

```
x[-seq(1, length(x), by = 2)] # Elimar posiciones impares
```

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

```
x[(length(x)-3):length(x)] # Para sacar solo las posiciones de antepenultima a ultima posicion
```

```
## [1] 38.0 41.5 45.0 48.5
```

```
x[c(1,2,4)]
```

```
## [1] 3.0 6.5 13.5
```

```
x[x>20 & x<40] # Escoger solo los numeros mayores de 20 y menores de 40
```

```
## [1] 20.5 24.0 27.5 31.0 34.5 38.0
```

```
x[x%%2==0] #Sacar solo numeros pares
```

```
## [1] 10 24 38
```

```
x>30 # Nos da true o false de los que cumplen o no
```

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

```
x = c(1,7,4,2,4,8,9,2,0)
```

```
y = c(5,2,-3,-7,-1,4,-2,7,1)
```

```
x[y>0] # Ambos vectores deben ser del mismo tamaño, nos saca los valores en x
```

```
## [1] 1 7 8 2 0
```

```
# de las posiciones de los numeros de y que cumplen la condición
```

```
which(x>4) # me da la posicion de los que son mayores de 4
```

```
## [1] 2 6 7
```

```
x[which(x>4)] # da los valores que cumplen la condicion
```

```
## [1] 7 8 9
```

```
which.min(x) # La posicion del valor mas pequeño
```

```
## [1] 9
```

```
which.max(x)
```

```
## [1] 7
```

```
which(x == max(x)) # Posiciones de los valores iguales al valor mas grande
```

```
## [1] 7
```

```
x = c() # Vector vacio, NULL  
x
```

```
## NULL
```

```
y = NULL  
y
```

```
## NULL
```

```
z = c(x, 2, y, 7) # los null no salen  
z
```

```
## [1] 2 7
```

LOS VALORES DE NA

PASARTE DEL RANGO DEL VECTOR

```
x <- c(1:10)  
x[length(x)+5] = 9 # Me pone un 9 en la ultima posicion mas cinco  
# pero como solo hay 10 posiciones en el resto pone NA  
x
```

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

OPERACIONES TENIENDO NA

```
sum(x) # No se puede, da NA
```

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



```
sum(x, na.rm = TRUE) # no tiene en cuenta los NA y hace la suma
```

```
## [1] 64
```

```
mean(x, na.rm = TRUE)
```

```
## [1] 5.818182
```

```
is.na(x) # Me dice si hay NA
```

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

```
which(is.na(x)) # Posiciones en las que hay NA
```

```
## [1] 11 12 13 14
```

En estadística descriptiva no se puede trabajar con NA así que se sustituyen por la media

```
y = x  
y[is.na(y)]
```

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

```
y[is.na(y)] = mean(y, na.rm = TRUE)  
y
```

```
## [1] 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000  
## [8] 8.000000 9.000000 10.000000 5.818182 5.818182 5.818182 5.818182  
## [15] 9.000000
```

Cumsum con NA

```
cumsum(x[!is.na(x)])
```

```
## [1] 1 3 6 10 15 21 28 36 45 55 64
```

Atributos y su eliminación

```
x_clean = na.omit(x) #Elimina los na  
x_clean # Se puede ver que elimina los na pero se queda con el atributo "na.action"
```

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

```
attr(x_clean, "na.action") = NULL # Asi elimino el atributo
x_clean
```

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

FACTORES

Un factor es una lista que tiene copias de las etiquetas colocadas (*Levels*)

```
nombres = c("Juan", "Antonio", "Maria", "Paco")
nombres
```

```
## [1] "Juan" "Antonio" "Maria" "Paco"
```

```
nombres.factor = factor(nombres) # Convierto el vector en un factor
nombres.factor
```

```
## [1] Juan Antonio Maria Paco
## Levels: Antonio Juan Maria Paco
```