Introducción a R

Email: delval@decsai.ugr.es

Alberto Armijo Ruiz.

1. R Interactivo

* Crea una secuencia de números impares

```
> seq(from=1,to=30,by=2)
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29
```

* Crea números del 1 al 30

```
> seq(from=1, to=30)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
```

* Busca en la ayuda que hace la función seq(). Describe que hace. Utilízala para crear números del 1 al 30 con un incremento de 0.5. ¿Qué otros parámetros te ofrece la función seq()? Utilízalos en un ejemplo.

Seq() se trata de una función que crea secuencias de números. Tiene varios argumentos, como el número de comienzo, el número final, el incremento, la longitud total de la secuencia creada, etc. Para crear una secuencia del 1 al 30 con 0.5 de incremento habría que escribir lo siguiente:

```
> seq(from=1,to=30, by=0.5)
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5
[27] 14.0 14.5 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0 20.5 21.0
21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5 26.0 26.5
[53] 27.0 27.5 28.0 28.5 29.0 29.5 30.0

Otros parámetros que ofrece la función son:
- lengt.out:
> seq(from=1,to=20, length.out = 11)
 [1]  1.0  2.9  4.8  6.7  8.6 10.5 12.4 14.3 16.2 18.1 20.0
- along.with:
> seq(from=1,to=20, along.with = rep(1,11))
 [1]  1.0  2.9  4.8  6.7  8.6 10.5 12.4 14.3 16.2 18.1 20.0
```

* Crea una secuencia de números indicando el principio y la longitud de la secuencia de números

```
> seq(from=1,length.out = 10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

*Crea letras minúsculas, mayúsculas, nombre de los meses del año y nombre de los meses del año abreviado

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
"s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R"
"S" "T" "U" "V" "W" "X" "Y" "Z"
month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"
"July"      "August"    "September" "October"
[11] "November"  "December"
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

* Investiga la función rep(). Repite un vector del 1 al 8 cinco veces.

La función rep() se utiliza para replicar el valor especificado un número de veces; puede usarse para repetir una secuencia de números o para repetir dichos números dentro de la secuencia. Tiene varios argumentos que permiten definir la longitud del nuevo vector, el número de veces que se repite cada letra, el número de veces que se repite el vector original. Para repetir un vector del 1 al 8 cinco veces:

```
> rep.int(1:8,5)
 [1] 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5
6 7 8
```

* Haz lo mismo con las primeras ocho letras del abecedario en mayúsculas

```
> rep.int(LETTERS[1:8],5)
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "A" "B" "C" "D" "E" "F" "G" "H" "A" "B"
"C" "D" "E" "F" "G" "H" "A" "B" "C" "D" "E" "F" "G" "H"
[33] "A" "B" "C" "D" "E" "F" "G" "H"
```

2. Vectores

* Crea los siguientes vectores:

- un vector del 1 al 20

```
x=1:20
```

- un vector del 20 al 1

```
y=20:1
```

- Utilizando el comando c() crea un vector que tenga el siguiente patrón 1,2,3,4,5…20,19,18,17….1

```
z=c(x,y[2:length(y)])
```

- Genera una secuencia de números del 1 al 30 utilizando el operador : y asígnalo al vector x. El vector resultante x tiene 30 elementos. Recuerda que el operador ':' tiene prioridad sobre otros operadores aritméticos en una expresión.

```
X=1:30
```

- Genera un vector x que contenga números del 1 al 9. Utilizando el operador ':' . y utilizando otras opciones. PISTA: seq()

```
> seq_along(1:9)
[1] 1 2 3 4 5 6 7 8 9
```

- Genera un vector x que contenga 9 números comprendidos entre 1 y 5

```
> seq(from=1,to=5,length.out = 9)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

* Busca que hace la función sequence(). ¿Cual es la diferencia con la función seq()

La función sequence() crea un secuencia concatenando secuencias, para utilizar esta función debemos especificar con el elementos final de cada secuencia. La diferencia con seq() es que crea una secuencia de elementos a partir de la concatenación de secuencias de elementos, en vez de crear una sola secuencia.

* Crea un vector numérico utilizando la función c()

```
> t=c(1,2,3,4)
> t
[1] 1 2 3 4
```

* Accede al segundo elemento del vector

```
> t[2]
[1] 2
```

* Crea un vector numérico "z" que contenga del 1 al 10. Cambia el modo del vector a carácter.

```
> z=1:10
> z=as.character(z)
> z
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

* Ahora cambia el vector z a numérico de nuevo

```
> z=as.numeric(z)
> z
 [1]  1  2  3  4  5  6  7  8  9 10
```

* Busca en la ayuda que hace la función scan(). Utilízala para leer un fichero cualquiera y asigna la lectura a un vector “z”.

La función scan() se utiliza para leer e> table(

l contenido de ficheros o de urls, para ello se debe especificar el nombre del fichero, el tipo de separador, número de líneas, número de líneas a saltar hasta encontrar los datos, etc.

```
z=scan(file="ejemplo_texto.txt",quiet = T, sep='',what='numeric')
> z
[1] "1" "2" "3" "4"
```

* Crea un vector x con 100 elementos, selecciona de ese vector una muestra al azar de tamaño 5. Busca que hace la función sample().

```
> s=sample(x,5)
> s
[1] 91 70  2 19 83
```

* Genera un vector de tipo con 100 números entre el 1 y el 4 de forma random. Para ello mira en la ayuda la función runif(). Obliga a que el vector resultante se ade tipo integer. Ordena el vector por tamaño usando la función sort(). ¿Qué devuelve la función sort?. SI quisieras invertir el orden de los elementos del vector que función utilizarías. Utiliza la función order() sobre x. ¿Cuál es la diferencia con la función sort()?

```
> x=as.integer(runif(100,1,4))
> x
  [1] 2 2 1 3 3 3 2 1 3 2 1 1 1 3 2 2 2 2 3 2 1 1 2 1 1 3 2 3 1 2 1 3 2 2 2 2
3 2 1 1 3 1 2 3 2 3 1 3 3 1 1 1 1 1 3 1 1 3 1 3 2 3 3 1
 [65] 2 2 3 2 1 1 2 2 3 1 1 1 2 3 1 2 3 2 2 1 3 2 2 2 1 3 3 1 2 2 1 3 2 3 3 2
> y=sort(x)
> y
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [65] 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

La función sort() ordena los elementos de un vector. Se puede expecificar si debe ser orden creciente o decreciente.

```
> y=sort(x,decreasing = T)
> y
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [65] 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

> order(x)
  [1]   3   8  11  12  13  21  22  24  25  29  31  39  40  42  47  50  51  52
53  54  56  57  59  64  69  70  74  75  76  79  84  89
 [33]  92  95   1   2   7  10  15  16  17  18  20  23  27  30  33  34  35  36
38  43  45  61  65  66  68  71  72  77  80  82  83  86
 [65]  87  88  93  94  97 100   4   5   6   9  14  19  26  28  32  37  41  44
46  48  49  55  58  60  62  63  67  73  78  81  85  90
 [97]  91  96  98  99
```

La función order() devuelve la permutación de los valores del vector especificado para que este esté ordenado, ya sea ascendente o decrecientemente; en cambio, la función sort() devuelve los elementos del vector ordenados en vez de las posiciones.

* Crea un vector x que contenga los números ordenados del 1 al 10 de forma consecutiva. Investiga la función rep(). Una vez comprobado que funciona, elimina las entradas repetidas del vector, para ello consulta la función unique().

```
> x=rep(1:10,times=10)
> x
  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4
5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3
 [44]  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7
8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6
 [87]  7  8  9 10  1  2  3  4  5  6  7  8  9 10

> x=unique(x)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

* Crea un vector cualquiera. Devuelve de ese vector una muestra cinco números seleccionada al azar. Usa la función sample(), prueba a determinar los valores que quieres extraer con y sin remplazo.

```
> x=as.integer(runif(100,1,11))
> x
  [1]  7  5  7  8  1  8  6  2  2  5  8  1  9  7  2  2  7  2  6  6  4  6  3  3
5  9  5  7  3  6  8  8  7  2  8 10  7  4  2  2  2  3  9
 [44]  3  5  3  6 10  6  1  7  3  9 10  9  1  4  7  6  8  4  6 10  6  7  5  3
2  1  6  2 10  5  4  1  2  4  2  9  8  6 10 10  5  4  2
 [87]  2  3 10  6  8 10  2  6 10  6  9  9  8  7

> sample(x,5)
[1]  6  8 10  6  3

> sample(x,5,replace = T)
[1]  5  5 10  2  2
```

* Comprueba que objetos tienes ahora en tu espacio de trabajo. Prueba con la función ls() y la función objects()

```
> ls()
[1] "s" "t" "x" "y" "z"
> objects()
[1] "s" "t" "x" "y" "z"
```

3. Explora el indexado de Vectores

- Ejecuta los siguientes comandos y comprueba su resultado

```
x <- 1:10
names(x) <- letters[x]
```

```
x[1:3]
x[c(1,10)]
x[c(-1,-2)]
x[ x > 5]
x[c("a","d")]
x[]
x <- 1:10; y <- c(x[1:5],99,x[6:10]); y
```

```
> x <- 1:10
> names(x) <- letters[x]
>
> x[1:3]
a b c
1 2 3
> x[c(1,10)]
 a  j
 1 10
> x[c(-1,-2)]
 c  d  e  f  g  h  i  j
 3  4  5  6  7  8  9 10
> x[ x > 5]
 f  g  h  i  j
 6  7  8  9 10
> x[c("a","d")]
a d
1 4
> x[]
 a  b  c  d  e  f  g  h  i  j
 1  2  3  4  5  6  7  8  9 10
> x <- 1:10; y <- c(x[1:5],99,x[6:10]); y
 [1]  1  2  3  4  5 99  6  7  8  9 10
```

La función names() asigna un nombre a cada uno de los elementos del vector x, por ello cuando se muestran los elementos del vector se muestra además su nombre, esto permite acceder al vector a través de los nombres en vez de por el índice.

* Crea un vector con números del 1 al 100 y extrae los valores del 2 al 23.

```
> x=1:100
> x[2:23]
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

* Del mismo vector x extrae ahora todos los valores menos del 2:23

```
> x[-2:-23]
 [1]   1  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
41  42  43  44  45  46  47  48  49  50  51  52  53  54
[33]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
73  74  75  76  77  78  79  80  81  82  83  84  85  86
[65]  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

* Cambia el número en la posición 5 por el valor 99

```
> x[5]=99
> x
  [1]   1   2   3   4  99   6   7   8   9  10  11  12  13  14  15  16  17  18
19  20  21  22  23  24  25  26  27  28  29  30  31  32
```

```
 [33]  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50
51  52  53  54  55  56  57  58  59  60  61  62  63  64
 [65]  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82
83  84  85  86  87  88  89  90  91  92  93  94  95  96
 [97]  97  98  99 100
```

* Crea un vector lógico del vector letters, (e.g. comprobando si existe c en el vector letters).

```
> is_c=letters=='c'
> is_c
 [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[22] FALSE FALSE FALSE FALSE FALSE
```

* ¿Qué devuelve el siguiente comando? `which(rep(letters,2)=="c")`

Devuelve las posiciones en las que la letra es la c.

* ¿Qué devuelve el siguiente comando? `match(c("c","g"), letters)`

Devuelve la posiciones de las letras c y g dentro del vector letters.

* Crea un vector x de elementos -5 -1, 0, 1, . . . , 6. Escribe un código en R del tipo x['something'], para extraer

elementos de x menores que 0,

```
> x[x < 0]
[1] -5 -1
```

elementos de x menores o igual que 0,

```
> x[x <= 0]
[1] -5 -1  0
```

elementos of x mayor o igual que 3,

```
> x[x >= 3]
[1] 3 4 5 6
```

elementos de x menor que 0 o mayor que 4

```
> x[x < 0 | x > 4]
[1] -5 -1  5  6
```

elementos de x mayor que 0 y menor que 4

```
> x[x > 0 & x < 4]
[1] 1 2 3
```

elementos de x distintos de 0

```
> x[ x != 0]
[1] -5 -1  1  2  3  4  5  6
```

* El código is.na se usa para identificar valores ausentes (NA). Crea el vector `x<- c(1,2,NA)` y averigua que pasa cuando escribes `is.na(x).` Prueba

con `x[ x!=NA ]` ¿obtienes con este comando los missing values de x?. ¿cuál es tu explicación?

```
> is.na(x)
[1] FALSE FALSE  TRUE
```

Comprueba cada uno de los valores del vector y devuelve si es NA o no.

```
> x[x!=NA]
[1] NA NA NA
```

No, devuelve todos los valores del vector como NA, esto es porque es incapaz de comparar un valor con NA.

4. Búsqueda de valores idénticos y distintos en Vectores

* Haz la intersección de dos v> table(
ectores `month.name[1:4]` y `month.name[3:7]` usando la función `intersect()`.

```
> intersect(month.name[1:4],month.name[3:7])
[1] "March" "April"
```

* Recupera los valores idénticos entre dos vectores usando %in%. Esta función devuelve un vector lógico de los elementos idénticos. Utiliza esa función para poder extraer ese subset del vector original.

```
> month.name[1:4][month.name[1:4]%in%month.name[3:7]]
[1] "March" "April"
```

* Si x=month.name[1:4] e y= month.name[3:7]
recupera los valores únicos en el primer vector. Para ello investiga la función setdiff().¿Puedes usarlo con caracteres?. Busca una alternativa.

```
> setdiff(x,y)
[1] "January"  "February"
```

La función setdiff() calcula los elementos que están en el primer vector pero no en el primero.

```
> setdiff("xy","y")
[1] "xy"
```

Con caracteres sí funciona. La función que no acepta caracteres es diff().

* Une dos vectores sin duplicar las entradas repetidas. Investiga la función unión().

```
> x=1:10
> y=5:12
> union(x,y)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

La función union(x,y) hace la unión de dos conjuntos

* Recupera las entradas duplicadas que existen entre el vector x y el vector y

```
> intersect(x,y)
[1] "March" "April"
```

5. Filtrado de Vectores, subset(), which(), ifelse()

* R permite extraer elementos de un vector que satisfacen determinadas condiciones. Es una de las operaciones mas comunes. Dado el vector z obtén las posiciones donde el cuadrado de z sea mayor que 8 sin utilizar ninguna función, con filtrado normal

- `z <- c(5,2,-3,8)`

```
> z=c(5,2,-3,8)
> z2=z**2
> z2[z2>8]
[1] 25  9 64
```

* R permite extraer elementos de un vector que satisfacen determinadas condiciones usando la función subset(), la diferencia entre esta función y el filtrado normal es como funciona con NA, subset(9 los elimina automáticamente del cálculo. Para el vector `x <- c(6,1:3,NA,12)` calcula los elementos mayores que 5 usando primero el filtrado normal
y luego la función subset()

```
> x=c(6,1:3,NA,12)
> subset(x,x>5)
[1]  6 12
```

* R permite extraer encontrar las posiciones en las que se encuentran los elementos que cumplen una determinada condición con which(). Utiliza esta función para encontrar dado el vector z, las posiciones donde el cuadrado de z sea mayor que 8

- `z <- c(5,2,-3,8)`

```
> z=c(5,2,-3,8)
> z2=z^2
> which(z2>8)
[1] 1 3 4
```

* En R aparte de encontrarse los típicos bucles if-then-else existe la función ifelse(). Ifelse funciona de la siguiente manera (ver ejemplo). Para un vector x devuelve 5 para aquellos números que sean pares (módulo igual a 0) y 12 para los números impares.

```
> y=as.integer(runif(100,1,100))
> ifelse(y%%2==0,5,12)
  [1] 12 12 12  5 12  5 12  5 12  5 12 12  5  5  5  5  5  5 12 12  5
5 12 12 12  5  5 12  5  5  5  5  5 12 12 12  5 12  5  5
 [44]  5  5 12  5  5  5 12  5  5 12  5 12 12  5  5 12 12  5  5  5  5  5 12  5
5  5 12 12 12 12  5  5  5  5 12 12 12  5 12 12  5 12  5
 [87]  5  5  5  5 12  5  5  5 12 12 12  5 12 12
```

- Práctica ahora para el vector x <- c(5,2,9,12)devuelve el doble de x si el valor de x es mayor que 6 y el triple si no lo es.

```
> x <- c(5,2,9,12)
> ifelse(x>6,x*2,x*3)
[1] 15  6 18 24
```