Introducción a R

Email: delval@decsai.ugr.es

Carmen Biedma Rodriguez

1. R Interactivo

* Crea una secuencia de números impares

Basta con crear una secuencia con el comando seq() poniendo como primer elemento un número impar y poner el incremento de 2 en 2 en el último argumento de la llamada.

* Crea números del 1 al 30

Seq (1,30)

* 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.

Es una función que genera secuencias de números de acuerdo al criterio que se le pase como argumento. La regla para crear la secuencia más básica que podemos proporcionarle a este método es un comienzo y un final, de forma que creará una secuencia de números enteros desde el primero que indicamos hasta el último. Ej: seq(1,5) nos generará una secuencia 1,2,3,4,5.

Además tenemos muchas posibilidades con dicha función, por ejemplo podemos decir el incremento que queremos en nuestra secuencia, al igual que se pide en el enunciado. Para ello utilizaremos 3 argumentos separados por comas: el primero será el 1 (comienzo), el segundo será 30 (final) y el tercero es el que determina el incremento en la secuencia, por lo que pondremos un 0.5. La llamada a la función quedaría de la siguiente manera: seq(1,30,by=0.5).

Otra función interesante que tiene seq() es que podemos establecer un com ienzo, un final y una longitud. Esta longitud determinará el tamaño de la sec uencia, y será R el que decidirá el incremento que tiene que utilizar en la se cuencia para que sea del tamaño preestablecido. Para ver mejor éste funcio namiento usaremos un ejemplo: seq(0,1,length.out=5) nos devolverá la sigu iente salida 0.00 0.25 0.50 0.75 1.00.

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

Está contestada en la pregunta anterior.

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

Para crear las letras del abecedario en minúscula basta con utilizar el comando letters. Para que las letras salgan en mayúscula pondremos la palabra LETTERS en mayúscula.

Para el caso de los meses también tenemos un comando reservado, en éste caso se llama month. Para que salga el nombre completo pondremos month.name y para que salgan las abreviaturas el comando es month.abb.

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

Con la función rep() podemos repetir cualquier objeto las veces que queramos. La sintaxis de uso es la siguiente: seq(objeto a repetir, nveces). En primer lugar diremos qué queremos repetir (un número, un carácter, una secuencia…) y en segundo el número de veces que repetiremos dicho objeto.

Para repetir una secuencia del 1 al 8 repetiremos una secuencia del 1 al 8 de la siguiente manera: rep(seq(1,8),5).

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

En este caso no podemos realizar la secuencia con la función seq ya que es un conjunto de letras y no una secuencia numérica. Para realizar las repeticiones entonces lo que tendremos que hacer es crear un array con las 8 letras mayúsculas y repetirlas. De ésta manera el comando que utilizaremos será el siguiente:

Rep(c("A","B","C","D","E","F","G","H"),5)

2. Vectores

* Crea los siguientes vectores:

- un vector del 1 al 20
seq(1,20))

- un vector del 20 al 1

seq(20,1)

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

c(seq(1,20),seq(20,1))

- 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.

Con el comando x <- 1:30 generamos un vector x con la secuencia especificada.

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

  x <- seq(1,9)
  x <- 1:9

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

```
> x <- c(seq(from = 1 , to = 5, length.out = 9))
```

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

La función seq() crea una secuencia dados un inicio y un final y sequence() crea una secuencia de secuencias desde el 1 hasta los valores indicados.

```
> sequence(c(2,5))
[1] 1 2 1 2 3 4 5
> seq(2,5)
[1] 2 3 4 5
```

* Crea un vector numérico utilizando la función c()

```
> vector <- c(1,2,3,4,5)
> vector
[1] 1 2 3 4 5
```

* Accede al segundo elemento del vector

```
> vector[2]
[1] 2
```

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

```
> z <- seq(1,10)
> z
 [1]  1  2  3  4  5  6  7  8  9 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”.

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

La función sample hace exactamente lo que pide el ejercicio. Para utilizarla pasaremos como primer argumento el vector del que queremos sacar la muestra y a continuación el tamaño que queremos que tenga la muestra. La función sample sacará automáticamente una muestra aleatoria del tamaño que hayamos establecido.

```
> x <- c(seq(1,100))
> sample(x,5)
[1] 67 21 23 11 74
```

* 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 sea de 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()?

En este caso utilizaremos la función runif() para generar la secuencia aleatoria de valores (por defecto serán números reales). A continuación lo convertiremos a un vector de enteros. Cabe destacar que los argumentos que hay que pasar a la función runif() son: primero el número de elementos que queremos obtener, segundo el límite inferior del intervalo en el que se generarán los datos y tercero el límite superior (no incluido).

```
> x <- as.integer(c(runif(100,1,4)))
```

Para ordenar el vector haremos uso de la función sort(), a la que le pasaremos como argumento el vector que queremos que ordene. Al igual que en el resto de funciones ésta no modifica el vector que se le pasa como argumento, por lo que para ordenarlo tendremos que reasignar su valor al que nos proporcione la función sort.

```
> x <- sort(x)
```

Si queremos invertir el orden de la secuencia tendremos que establecer a TRUE el parámetro "decreasing" de la función sort de la siguiente manera:

```
> x <- sort(x,decreasing = TRUE)
```

La función sort() ordena los valores y la función order() devuelve la permutación de los elementos del vector para que esté ordenado.

* 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().

Con los siguientes comandos crearemos un vector con los números consecutivos del 1 al 10 y los repetiremos 2 veces.Para eliminar los repetidos usaremos la función unique() pasándole como argumento el vector sobre el que queremos que actúe. Para obtener los resultados en el mismo vector que teníamos originalmente lo reasignaremos con el resultado de ésta operación.

```
> x <- c(1:10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- rep(x,2)
> x
 [1]  1  2  3  4  5  6  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.

Para sacar la muestra usaremos la función sample pasando como argumento el vector que hemos creado previamente y asignaremos el resultado a un vector llamado muestra. Por defecto la función extrae la muestra sin reemplazo. Si quisiésemos que se hiciese con reemplazo tendríamos que poner la variable replace a TRUE como se muestra en la última línea de ejecución de la siguiente prueba:

```
> x <- c(seq(1,50))
> muestra <- c(sample(x,5))
> muestra
[1]  9 20  6 19 46
> muestra <- c(sample(x,5,replace = TRUE))
> muestra
[1] 13 16 12 23 21
```

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

```
> ls()
[1] "muestra" "vector"  "x"       "y"       "z"
> objects()
[1] "muestra" "vector"  "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
```

En primer lugar se crea un vector de números consecutivos del 1 al 10. La función letters devuelve tantas letras como elementos tengamos en el vector que le pasamos como argumento, por orden alfabético, en este caso nos generará las letras de la a "a" hasta la "j". Con el comando names() le pondremos "nombre" a cada uno de los elementos de la secuencia que le pasamos como argumento. En este caso creará unos índices con una letra que identificará cada uno de los elementos: el 1 será la letra "a", el 2 la letra "b" y así sucesivamente.

Con el comando x[1:3] mostrará los elementos en las posiciones 1,2 y 3 con sus respectivos nombres. De la misma manera, el comando x[c(1,10)] mostrará los elementos en las posiciones 1 y 10 con sus respectivos índices. Éste último comando también se puede usar poniendo posiciones "negativas", lo que significará que mostrará todas menos las que especifiquemos con el signo - . En el caso del ejemplo se muestran los elementos desde la posición 3 hasta la 10, quitando la 1 y la 2 (también con sus correspondientes índices).

Además, como vemos en el comando x[x>5] , podemos filtrar los elementos que queremos mostrar del vector que pasamos. En éste caso se mostrarán las posiciones del vector cuyo contenido sean mayores que 5.

Una vez que tenemos nombres para cada uno de los elementos de nuestro vector, podemos acceder a sus posiciones también especificando el mismo. Como podemos ver en el ejemplo del ejercicio, con el comando x["a","d"] podemos mostrar las posiciones que tienen dicho nombre, que son la 1 y la 4.

Con el comando x[ ] mostramos todos los elementos del vector x.

La última línea de código crea en primer lugar un vector x con una secuencia del 1 al 10 consecutiva. A continuación crea otro vector y concatenando varias cadenas. Para ello utiliza el comando c() y coloca en primer lugar los elementos del 1 al 5 de la cadena x, luego un 99 y por último la parte de la cadena del 6 al 10. El resultado es el siguiente:

```
> 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
```

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

```
> x <- 1:100
> a <- x[2:23]
> a
 [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

```
> a <- c(x[1],x[24:100])
```

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

```
> x[5] <- 99
```

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

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

El comando which selecciona los elementos de un vector que cumplen la condición especificada. En este caso se ha creado un vector con todas las letras del abecedario en minúscula repetidas y se han seleccionado las posiciones que coinciden con la letra c.

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

Este comando hace algo parecido al which que hemos visto en el ejercicio anterior. En el primer argumento se pasa un vector con los elementos que queremos seleccionar del vector que le pasamos como segundo argumento. En este caso nos devuelve las posiciones de las letras c y g en el vector que devuelve el comando letters.

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

La creación del vector se hará de la siguiente manera:

```
> x <- c(-5,-1,0:6)
```

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] 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

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

En ese caso no obtenemos los missing values porque NA no tiene valor y no lo compara con nada.

4. Búsqueda de valores idénticos y distintos en Vectores

* Haz la intersección de dos vectores `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.

```
> x[x%in%y]
[1]  5  6  7  8  9 10
```

* 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(month.name[1:4],month.name[3:7])
[1] "January"  "February"
```

Diff no se puede usar con caracteres pero setdiff si.

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

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

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

```
> intersect(x,y)
[1]  5  6  7  8  9 10
```

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[z^2>8]
[1]  5 -3  8
```

* 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[x>5]
[1]  6 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)`

```
> which(z^2>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.

```
> ifelse(x%%2,5,12)
 [1]  5 12  5 12  5 12  5 12  5 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
```