Introducción a R: Ejercicios Día 1

Juan Ramón Gómez Berzosa

15/10/2018

Ejercicio 1: R Interactivo

*** Crea una secuencia de números interactivos**

```
#Primera Opción
a = c(1:20)
ej1.1 = a[a %% 2 != 0]
ej1.1
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

```
#Segunda Opción

b = seq(1,20,by=2)
b
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

*** Crea números del 1 al 30**

```
#Primera Opción

ej1.2 = seq(1,30)
ej1.2
```

```
##  [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] 24 25 26 27 28 29 30
```

```
#Segunda Opción

b = c(1:30)
b
```

```
##  [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] 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.**

La función "seq" genera una secuencia regular de números enteros o flotantes. Se le pueden poner algunas restricciones como dónde empezar la secuencia (from) y donde terminarla (to) o de cuanto en cuanto hacer el incremento (by). Existen otros argumentos como "length.out" que permiten indicar la longitud del vector de salida que queremos, divide la secuencia de modo que existan tantos valores como hemos indicado en la secuencia entre to y from, se redondea para que no queden números fraccionales (también se puede utilizar seq_len que es una forma más rápida de usar la función seq con dicho argumento). El argumento "along.with"

hace la misma función que el argumento anterior pero cogiendo la longitud del vector que le indiquemos a este argumento (también se puede utilizar la función seq_along que haría la misma función).

```
#Secuencia del 1 al 30 con un incremento del 0.5

ej1.3.1 = seq(1,30, by= 0.5)
ej1.3.1
```

```
##  [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
## [15]  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5
## [29] 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
## [43] 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5 26.0 26.5 27.0 27.5 28.0 28.5
## [57] 29.0 29.5 30.0
```

```
#Ejemplo parámetro length.out

ej1.3.2 = seq(1,30, length.out = 10)
#Como podemos observar la secuencia contiene 10 elementos en total.
ej1.3.2
```

```
##  [1]  1.000000  4.222222  7.444444 10.666667 13.888889 17.111111 20.333333
##  [8] 23.555556 26.777778 30.000000
```

```
#Ejemplo parámetro along.with

ej1.3.3 = seq(1,30, along.with = month.abb)
#month.abb es un vector que contiene los meses del año abreviados, luego la secuencia tendrá una
#longitud de 12 elementos en total.
ej1.3.3
```

```
##  [1]  1.000000  3.636364  6.272727  8.909091 11.545455 14.181818 16.818182
##  [8] 19.454545 22.090909 24.727273 27.363636 30.000000
```

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

```
ej1.4 = seq(1, length.out = 30)
ej1.4
```

```
##  [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] 24 25 26 27 28 29 30
```

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

```
#Letras minúsculas
ej1.5.1 = letters
ej1.5.1
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
#Letras mayúsculas
ej1.5.2 = LETTERS
ej1.5.2
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
#Meses del año
ej1.5.3 = month.name
ej1.5.3
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```

```
#Meses del año abreviados
ej1.5.4 = month.abb
ej1.5.4
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
## [12] "Dec"
```

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

```
ej1.6 = rep(seq(1,8),5)
ej1.6
```

```
##  [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
## [36] 4 5 6 7 8
```

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

```
ej1.7 = ej1.5.2[1:8]
ej1.7
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
```

Ejercicio 2: Vectores

*** Crea los siguientes vectores:**

- un vector del 1 al 20

- un vector del 20 al 1

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

```
#Vector del 1 al 20
ej2.1 = c(1:20)
ej2.1
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
#Vector del 20 al 1
ej2.2 = c(20:1)
ej2.2
```

```
##  [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

```
#Vector con el patron 1,2,3,4,5...20,19,18,17....1
#Opción 1
ej2.3.1 = c(c(1:20),c(19:1))
ej2.3.1
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18 17
## [24] 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

```
#Opción 2 aunque usando otros comandos además de c()
ej2.3.2 = as.integer(paste(c(1:20, 19:1)))
ej2.3.2
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18 17
## [24] 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  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.**

```
x = 1:30
x
```

```
##  [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] 24 25 26 27 28 29 30
```

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

```
# Opción 1
x1 = 1:9
x1
```

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

```
# Opción 2
x2 = seq(1,9)
x2
```

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

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

```
x =
x = seq(1,5, length.out = 9)
x
```

```
## [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 una secuencia para cada elemento del vector que le pasamos como argumento, siendo números enteros positivos. Por ejemplo: sequence(c(3,2)) creará una secuencia del 1 al 3 y después

otra del 1 al 2 y las concatenará. Sin embargo, seq crea una secuencia de números enteros positivos, negativos, enteros o flotantes y además se puede indicar con que número iniciar la secuencia. Por defecto si nosotros usamos sequence(15) nos creará una secuencia del 1 a 15, del mismo modo que si usamos seq(15).

*** Crea un vector numérico utilizando la función c()**

```
vector = c(1:15)
vector
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

*** 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.**

```
#Con el operador 1:10 creamos un vector númerico y posteriormente con la función
# as.character convertimos el vector en uno de caractéres
z = as.character(1:10)
z
```

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

*** Ahora cambia el vector z a numérico de nuevo**

```
z = as.integer(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".**

Creamos un archivo con el texto "Hola mundo", con nombre prueba.txt.

```
#en primer lugar establezco el directorio de trabajo con la función setwd()
setwd("~/Master/IntroduccionR/Ejercicios1")

#Uso scan para leer el fichero usando el parámetro what para indicarle que tipo
#de datos leer, ya que por defecto es double
z = scan(file="prueba.txt", what="string")
z
```

```
## [1] "Hola"  "mundo"
```

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

```
ejercicio2.11 = 1:100
# Con el argumento size indicamos el número de elementos de la muestra
# Exixte un parámetro replace que por defecto está en false y permite indicar si
# queremos o no que exista reemplazo entre las muestras aleatorias
muestra = sample(ejercicio2.11, size= 5)
muestra
```

```
## [1] 12 62 10 93 45
```

*** 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()?**

La función sort devuelve un vector ordenado de forma creciente o decreciente y la función order devuelve las posiciones en las que se encuentran los elementos de menor a mayor por defecto, aunque se puede establecer el orden decreciente.

```
# Generamos un vector de números reales comprendidos entre 1 y 4 y lo forzamos a que
# sea de números enteros
ejercicio2.12 = as.integer(runif(n = 100, min = 1, max = 4))

# Ordenamos el vector con la función sort
ejercicio2.12.sort.crec = sort(ejercicio2.12)
ejercicio2.12.sort.crec
```

```
##   [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
##  [36] 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 3 3 3 3 3 3 3
##  [71] 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 3 3
```

```
# Para invertir el orden usamos la función sort con el argumento decreasing = true,
# está en false por defecto
ejercicio2.12.sort.decr = sort(ejercicio2.12,decreasing = TRUE)
ejercicio2.12.sort.decr
```

```
##   [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 3 3 3
##  [36] 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
##  [71] 2 2 2 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
```

```
ejercicio2.12.order = order(ejercicio2.12)
ejercicio2.12
```

```
##   [1] 2 1 2 1 1 2 3 2 3 3 2 1 3 3 2 2 3 2 3 3 2 1 3 2 3 1 2 2 2 3 1 2 3 3 1
##  [36] 2 2 1 1 2 2 2 2 2 2 1 1 1 1 2 3 2 2 3 1 3 3 3 3 1 2 3 3 2 2 3 3 3 2 3
##  [71] 3 1 3 3 1 2 2 3 2 1 2 2 3 1 1 3 1 3 1 2 3 1 1 2 3 3 3 2 3 2
```

*** 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()**

Realmente si tú deseas crear un vector del 1 al 10 con los números ordenados de forma consecutiva no necesitas usar la función rep, así que lo que he hecho es usar la función rep con el argumento each = 3 para que me repita cada valor del vector x que de forma consecutiva 3 veces. Así, después aplico la función unique() para eliminar los repetidos y dejar el vector como el original.

```
x = 1:10
x = rep(x,each=3)
x
```

```
##  [1]  1  1  1  2  2  2  3  3  3  4  4  4  5  5  5  6  6  6  7  7  7  8  8
## [24]  8  9  9  9 10 10 10
```

```
# Eliminamos los repetidos
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 = 1:10
x1 = sample(x, size = 5)
x1
```

```
## [1] 5 7 2 8 4
```

```
# Con reemplazo
x2 = sample(x, size=5, replace = TRUE)
x2
```

```
## [1]  1  5 10  9 10
```

Como podemos observar, en el primer caso al usar el método por defecto el argumento replace está establecido en falso, por lo que no hay reemplazo y por tanto no existe repetición en la muestra aleatoria. En el segundo caso, si que hay reemplazo pues lo hemos establecido así, por lo que si puede haber números repetidos.

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

```
#ls

ls()
```

```
##  [1] "a"                     "b"
##  [3] "ej1.1"                 "ej1.2"
##  [5] "ej1.3.1"               "ej1.3.2"
##  [7] "ej1.3.3"               "ej1.4"
##  [9] "ej1.5.1"               "ej1.5.2"
## [11] "ej1.5.3"               "ej1.5.4"
## [13] "ej1.6"                 "ej1.7"
```

```
## [15] "ej2.1"                    "ej2.2"
## [17] "ej2.3.1"                  "ej2.3.2"
## [19] "ejercicio2.11"            "ejercicio2.12"
## [21] "ejercicio2.12.order"      "ejercicio2.12.sort.crec"
## [23] "ejercicio2.12.sort.decr" "muestra"
## [25] "vector"                   "x"
## [27] "x1"                       "x2"
## [29] "z"
```

```
#objects

objects()
```

```
##  [1] "a"                     "b"
##  [3] "ej1.1"                 "ej1.2"
##  [5] "ej1.3.1"               "ej1.3.2"
##  [7] "ej1.3.3"               "ej1.4"
##  [9] "ej1.5.1"               "ej1.5.2"
## [11] "ej1.5.3"               "ej1.5.4"
## [13] "ej1.6"                 "ej1.7"
## [15] "ej2.1"                 "ej2.2"
## [17] "ej2.3.1"               "ej2.3.2"
## [19] "ejercicio2.11"         "ejercicio2.12"
## [21] "ejercicio2.12.order"   "ejercicio2.12.sort.crec"
## [23] "ejercicio2.12.sort.decr" "muestra"
## [25] "vector"                "x"
## [27] "x1"                    "x2"
## [29] "z"
```

La función ls y objects devuelven un vector de string con los nombres de todas las variables que hemos definido en nuestro espacio de trabajo. La única diferencia es que si ls se utiliza dentro de una función devuelve el nombre de las variables locales que se han definido en ella.

Ejercicio 3: Explora el indexado de vectores

*** 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[c(-2:-23)]
```

```
##  [1]   1  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39
## [18]  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
## [35]  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73
## [52]  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
## [69]  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]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

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

```
x = letters == "c"
x
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE
```

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

```
which(rep(letters,2)=="c")
```

```
## [1]  3 29
```

Devuelve la posición dónde se encuentran los dos elementos del vector que cumplen la condición true, en este caso la posición de las 2 letras c en el vector lógico.

*** ¿Qué devuelve el siguiente comando? match(c(“c”,“g”), letters)?**

```
match(c("c","g"), letters)
```

```
## [1] 3 7
```

Devuelve la primera ocurrencia dentro del vector letters donde se encuentran las letras c y g.

*** 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,

-elementos de x menores o igual que 0,

-elementos of x mayor o igual que 3,

-elementos de x menor que 0 o mayor que 4

-elementos de x mayor que 0 y menor que 4

-elementos de x distintos de 0

```
x = -5:6
x
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5  6
```

```
#elementos de x menores que 0
x[x<0]
```

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

```
#elementos de x menores o igual que 0
x[x<=0]
```

```
## [1] -5 -4 -3 -2 -1  0
```

```
#elementos de 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 -4 -3 -2 -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 -4 -3 -2 -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?**

```
x = c(1,2,NA)
is.na(x)
```

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

Nos devuelve un vector lógico indicándonos true si encuentra un missing value.

```
x[x != NA]
```

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

Se obtiene un vector con las mismas posiciones que nuestro vector original pero con valor NA. Esto se debe a que NA es una palabra reservada en R que se utiliza para dar nombre a los valores desconocidos, por lo tanto al ser desconocidos no podemos compararlos ya que no sabemos que son en realidad. Por tanto, cuando comprobamos si una posición de x, es distinta de NA nos va a devolver NA ya que realmente no conocemos el valor de NA para poder compararlos.

Ejercicio 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 = 1:40
y = 30:60

x[x%in%y]
```

```
##  [1] 30 31 32 33 34 35 36 37 38 39 40
```

*** 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.

La función setdiff() te dice los elementos en los que difiere el primer vector del segundo. Esta función si se puede usar con caracteres, pero con la función diff no se puede.

```
setdiff(month.name[1:4], month.name[3:7])
```

```
## [1] "January"  "February"
```

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

```
vector.x = c(1:20)
vector.y = c(10:30)
union(vector.x,vector.y)
```

```
##  [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] 24 25 26 27 28 29 30
```

La función union() elimina las entradas duplicadas.

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

```
intersect(vector.x,vector.y)
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

Ejercicio 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 los valores donde el cuadrado de z sea mayor que 8 sin utilizar ninguna función, con filtrado normal**

```
z = c(1:100)
z[ (z**2) > 8]
```

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

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

```
## [1]  6 NA 12
```

```
subset(x, x>5)
```

```
## [1]  6 12
```

La diferencia está en que el filtrado normal no trata los valores NA.

*** 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.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, 2*x, 3*x)
```

```
## [1] 15  6 18 24
```