**R objects**

The fundamental data type in R is the *vector*. R uses functions or operators which act on objects (vectors, matrices etc.). Important modes of vector are

* Null (empty object): Null
* Logical (Boolean): TRUE, FALSE or T, F
* Numeric (real number): 1,2,3, pi, 1e-5
* Complex(complex number): 2+i, 2i
* Character(chain of characters): “ABC”, “hi”

In order to know the mode of an object, say x, in R we can use

>mode(x)

It is also possible to test which mode a particular object, say x, belongs to.

> x=c(1,2,3,4,5,6)

> is.null(x)

[1] FALSE

> is.logical(x)

[1] FALSE

> is.numeric(x)

[1] TRUE

> is.complex(x)

[1] FALSE

> is.complex(x)

[1] FALSE

> is.character(x)

[1] FALSE

**Numeric Vectors:**

There are number of different ways of constructing a vector

1. Use ***collect function or concatenate: c***

>x=c(2, 4, 5, 6.8, 7.2) # Numeric vector with 5 entries

>y<-5 # vector of length 1

1. Construction by ***sequence*** operator

> 1:10

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

> seq(1,10, by=3) # We can simply use seq(1,10,3)

[1] 1 4 7 10

> seq(1,50,length=8)

[1] 1 8 15 22 29 36 43 50

c) Construction by ***rep function***

> rep(1,6)

[1] 1 1 1 1 1 1

> rep(c(1,2,3,4), each=3)

[1] 1 1 1 2 2 2 3 3 3 4 4 4

> rep(c(1,2,3,4), 3)

[1] 1 2 3 4 1 2 3 4 1 2 3 4

1. Construction by ***scan function***

R will ask you to enter the data one at a time and enter

> x=scan()

1: 1

2: 2

3: 4

4:

Read 3 items

> x

[1] 1 2 4

**Character Vectors**

It is possible to create character vectors in the same way as the numeric vectors with functions ***c or rep***

> x<-c("A", "BB", "CCC", "Example")

> x

[1] "A" "BB" "CCC" "Example"

> y<-rep("A", 10)

> y

[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"

> z<-rep('STAT',10)

> z

[1] "STAT" "STAT" "STAT" "STAT" "STAT" "STAT" "STAT" "STAT" "STAT" "STAT"

Note that in R ” and ’ are essentially the same.

**Logical Vectors**

Boolean vectors are usually generated with logical operations:

“>”, “>=”, “<”, “<=”, “= =”, “! =” etc.

> 1>0

[1] TRUE

> 1<5

[1] TRUE

> 2==2

[1] TRUE

> 2<=2

[1] TRUE

> 3<=6

[1] TRUE

> 3!=6

[1] TRUE

**Using all() and any()**

The any() and all() functions are handy shortcuts. They report whether any or all of their arguments are TRUE.

> x <- 1:10

> any(x > 8)

[1] TRUE

> any(x > 88)

[1] FALSE

> all(x > 88)

[1] FALSE

> all(x > 0)

[1] TRUE

**Vector In, Vector Out**

> u<-c(5,2,8)

> v<-c(1,3,9)

> u>v

[1] TRUE FALSE FALSE

Note that here, the > function was applied to u[1] and v[1], resulting in TRUE, then to u[2] and v[2], resulting in FALSE, and so on.

A key point is that if an R function uses vectorized operations, it, too, is vectorized, thus enabling a potential speedup. Here is an example:

> u<-c(5,2,8)

> w <- function(x) return(x+1)

> w(u)

[1] 6 3 9

**Selecting Part of a vector**

Selections are made using the selection operator [ ] and a selection vector:

> x[indexvector]

**Example:**

> x=c(2,4,6,8,4,6,7,8,9,0,12,13,14,15)

> x

[1] 2 4 6 8 4 6 7 8 9 0 12 13 14 15

> x[5]

[1] 4

> x[6:9]

[1] 6 7 8 9

> x[-5]

[1] 2 4 6 8 6 7 8 9 0 12 13 14 15

> v=1:15

> v

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

> v[v<5]

[1] 1 2 3 4

> v[(v<4)|(v>12)] # | means OR

[1] 1 2 3 13 14 15

> v[(v<4)&(v>12)] # & means AND

integer(0)

**which() and match()**

***which()*** Indices of a logical vector where the condition is TRUE

***which.max()*** Location of the (first/second ) maximum element of a numeric vector

***which.min()*** Location of the (first/second ) minimum element of a numeric vector

***match()*** First position of an element in a vector

> x<-c(4,7,2,12,9,0)

> y <- rep(1:5, times=5:1)

> x

[1] 4 7 2 12 9 0

> y

[1] 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5

> which(x>4)

[1] 2 4 5

> which.max(x) # Location of the maximum x value

[1] 4

> which.min(x)# Location of the minimum x value

[1] 6

> x[which.max(x)] # Exact value of the maximum value

[1] 12

>

> y

[1] 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5

> match(1:5,y)

[1] 1 6 10 13 15

**Vector Operations**

When vectors are used in math expressions the operations are performed element by element

> x <- c(4,7,2,10,1,0)

> y <- x^2 + 1

> y

[1] 17 50 5 101 2 1

> x\*y

[1] 68 350 10 1010 2 0

**Testing Vector Equality**

Suppose we wish to test whether two vectors are equal. The naive approach, using = =, won’t work.

> x <- 1:3

> y <- c(1,3,4)

> x = = y

[1] TRUE FALSE FALSE

> identical(x,y)

[1] FALSE

Be careful, though because the word *identical* really means what it says.

Consider this little R session:

> x <- 1:2

> y <- c(1,2)

> x

[1] 1 2

> y

[1] 1 2

> identical(x,y)

[1] FALSE

> typeof(x)

[1] "integer"

> typeof(y)

[1] "double"

So, : produces integers while c() produces floating-point numbers.

**Missing Data in R**

For a number of reasons, certain elements of data may not be collected during and experiment or study. In **R**, missing values are represented by the symbol **NA** (not available) but impossible values (e.g., dividing by zero) are represented by the symbol ***NaN*** (not a number) and ***Inf*** for infinity.

Example:

> var(5) # Variance of a single observation

[1] NA

> log(-2)

[1] NaN

Warning message:

In log(-2) : NaNs produced

> exp(1e10)

[1] Inf

We can use is.na(x) to check whether there is missing value.

> x<-c(1,2,3,4,5,6,7,8,4,5,7)

> is.na(x)

[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

> y<-c(1,4,NA,6,9,NA,7)

> is.na(y)

[1] FALSE FALSE TRUE FALSE FALSE TRUE FALSE

**Detecting NA**

>y<-c(1,4,NA,6,9,NA,7)

>any(is.na(y)) # checking if there is any missing value

>which (is.na(y)) # Which one is NA?

[1] 3 6 # Answer: the third one and sixth one

**Ordering the missing Value**

The user can specify if NA should be last or first in a sorted order by indicating TRUE or FALSE for the **na.last** argument.

>y<-c(1,4,NA,6,9,NA,7)

> sort(y, na.last = TRUE)

[1] 1 4 6 7 9 NA NA

**Excluding Missing Values from Analysis**

**na.omit** and **na.exclude** returns the object with observations removed if they contain any missing values.

> x <- c(88,NA,12,168,13)

> x

[1] 88 NA 12 168 13

> mean(x)

[1] NA

> mean(x, na.rm=T)

[1] 70.25

> mean(na.omit(x))

[1] 70.25

> mean(na.exclude(x))

[1] 70.25

> > x <- c(88,NULL,12,168,13) # R has skipped the 2nd observation

> mean(x)

[1] 70.25

NULL values really are counted as nonexistent, as you can see here:

> u <- NULL

> length(u)

[1] 0

> v <- NA

> length(v)

[1] 1