Agenda

- Operators
- Built-in Functons
- Data Structures In R
- Data Types - Vectors
- R Data Structures - Matrices

Operators

What Is Assignment In R?

Objects obtain values in R by assignment **('x gets a value')**.

This is done either by <- or =,

Thus to create a scalar constant x with value 6,
we type
x <- 6 or x = 6
y <- "a" or y = "a"

What Are Operators In R?

Operator	Function	Example
+ - */%% ^	arithmetic	1+1; 2*9; 3-7
>= <= == !=	relational	
! &	logical	
~	model formulae	
<- ->	assignment	a<-5
$	list indexing (the 'element name' operator)	
:	create a sequence	10:20

What Is Screen Prompt In R?

- The screen prompt > is an invitation to put R to work.

```
> log (14/3)
[1] 1.540445
```

- Each line can have at most 128 characters.
- Two or more expressions can be on a single line so long as they are separated by semi-colons:

```
> 9+7; 2*4; 15-12
[1] 16
[1] 8
[1] 3
```

Built-in Functions

What Are Built In Functions In R ?

The log function gives logs to the base **e (e = 2718)**, for which the antilog function is **exp**

Function	Meaning
log (x):	log to base e of x
exp (x):	antilog of x (e^x)
log(x,n):	log to base n of x
log10(x):	log to base 10 of x
sqrt(x):	square root of x
factorial(x):	x!
floor(x):	greatest integer <x
ceiling(x):	smallest integer > x
trunc(x):	closest integer to x between x and 0
round(x, digits=0):	round the value of x to an integer
signif(x, digits=6):	give x to 6 digits in scientific notation
runif(n):	generates n random numbers between 0 and 1 from a uniform distribution
cos(x),sine(x),tan(x):	cosine of x (in radians), sine of x (in radians), tangent of x (in radians)
abs(x):	the absolute value of x, ignoring the minus sign if there is one

log(10) [1] 2.302585 exp(1) [1] 2.718282

Numbers With Exponents

For very big numbers or very small numbers R uses the following scheme:

Scheme	Meaning
1.9e3	1900 i.e. 1.9 multiplied by 1000
1.9e-2	0.019 i.e. 1.9 multiplied by 1/100

Modulo And Integer Quotients

To know the integer part of a division: say, how many 2s are there in 50:

```
> 50%/%2
[1] 25
```

To know the remainder (what is left over when 50 is divided by 2): in maths this is known as modulo:

```
> 50%%2
[1] 0
```

Rounding

There are multiple types of rounding :

rounding up rounding down rounding to the nearest integer

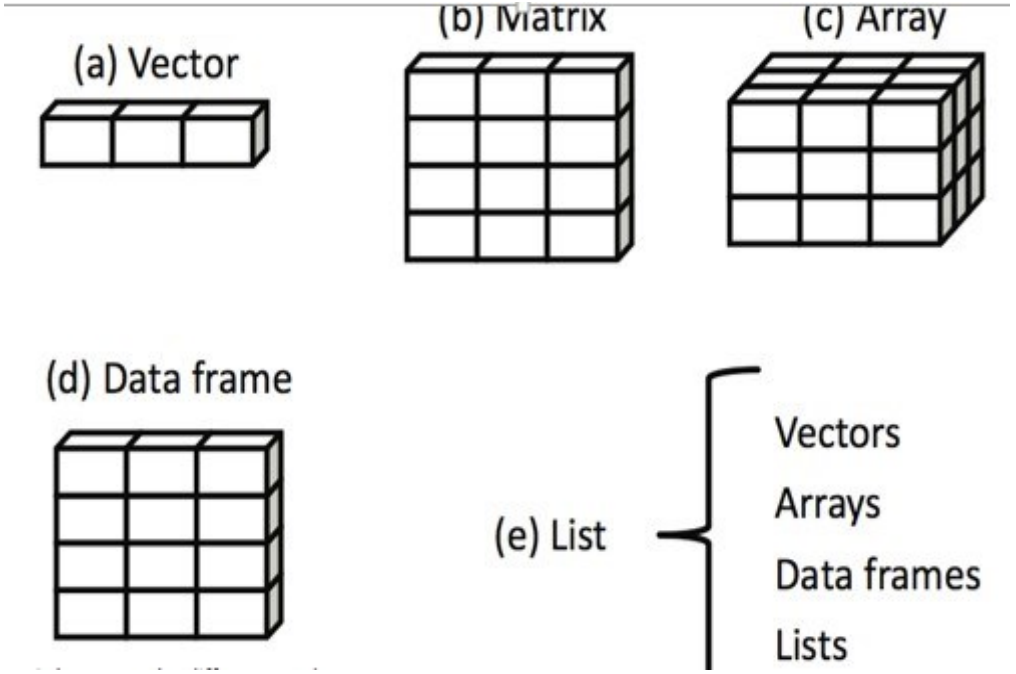
These can be done in R using

```
floor() :
 floor(5.9)
[1] 5
```

```
And the 'next integer' function is ceiling
ceiling (2.9)
[1] 3
```

Data Structures In R

R data structures



Data Type - Vectors

Creating A Vector

Vectors are variables with one or more than one values of the Same type: logical, integer, real, complex, string. Vectors could also have length 0

```
a<-1.3
a <- 5:10
[1]  5  6  7  8  9 10
```

To know the length of a vector :

```
length(a)
[1] 6
```

Values can be assigned to vectors in many different ways.
One can type the values into the command line, using the concatenation function c,

```
a <- c(5  6  7  8  9 10)
```

The length of the longest vector is assigned to a derived vector (created by calculation), here A is of length 5 and B is of length 2:

```
A<-1:5
B<-c(6,7)
A*B
[1]  6 14 18 28 30
```

Types Of Vectors

- Numeric vector
  - a <- c(4,3,6.3,6,-8,9)

    > typeof(a)

    [1] "double"
- Character vector
  - b <- c(“nine","two","eight")

- Logical vector
  - c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)

How to refer these elements !

- a[c(1,3)] # refers to 1st and 3rd elements of vector a

Vector Functions

Important vector functions are listed in Vector functions used in R.

Operation	Meaning
max(x):	maximum value in x
min(x):	minimum value in x
sum(x):	total of all the values in x
mean(x):	arithmetic average of the values in x
median(x):	median value in x
range(x):	vector of (minx) and(max)
var(x):	Sample variance of x
cor(x,y):	correlation between vectors x and y
sort(x) a:	sorted version of x
rank(x):	vector of the ranks of the values in x
order(x):	an integer vector containing the permutation to sort x into ascending

Identifying Missing Values

Missing values are a cause of concern and dealt accordingly:

```
> a<-c(NA,11:15,NA,NA)
[1] NA 11 12 13 14 15 NA NA
mean(a)
[1] NA
```

To handle missing values, using the na.rm=TRUE argument:

```
mean (a,na.rm=T)
[1] 13
```

To check for the location of missing values within a vector, use the function is.na(x)

```
a<-c(11:15,NA,NA,12:20)
 a
 [1] 11 12 13 14 15 NA NA 12 13 14 15 16 17 18 19 20
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE
```

```
which(is.na(a))
[1] 6 7
To convert the NA to 0, use the ifelse functon :
ifelse(is.na(a),0,a)
 [1] 11 12 13 14 15  0  0 12 13 14 15 16 17 18 19 20
```

To only display the position of NA in the data :

How To Work With Vectors & Logical Subscripts

Take the example of a vector containing the 8 numbers 0 to 7:

```
y<-0:7
```

To add up all the values:

```
sum(y)
[1] 28
```

To know how many of the values less than 3 were:

```
sum(y<3)
[1] 3
```

To find the sum of the values of x that are less than 3, we write:

```
sum(y[y<5])
[1] 10
```

To find out the logical condition x<3 is either true or false:

```
y<3
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
FALSE
```

How To Work With Vectors & Logical Subscripts

To find out the sum of the two largest values in a vector. First sort the vector into descending order, then add up the values of the last two elements of the sorted array. Let's do this in stages.

First, the values of a:

```
a<-c(2,9,9,9,2)
```

Now if you apply "sort" to this, the numbers will be sorted by ascending sequence,

```
sort(a)
[1] 2 2 9 9 9
sum(sort(a)[2:3])
[1] 11
```

Logical Arithmetic

Arithmetic involving TRUE or FALSE can be done in R . R can coerce TRUE or FALSE into numerical values: 1 for TRUE and 0 for FALSE.

```
a<-0:6
```

Is a less than 4?

```
 a<4
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

```
all(a>0)
[1] FALSE
```

```
 all(a>-1)
[1] TRUE
```

```
any(a<0)
[1] FALSE
```

Sum of values less than 2 in vector

a

Logical Operations

Symbol	Meaning
!	logical NOT
&	logical AND
|	logical OR
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	logical equals (double =)
!=	not equal
&&	AND with IF
||	OR with IF
xor(x,y)	exclusive OR
isTRUE(x)	an abbreviation of identical(TRUE,x)

Generating Regular Sequences Of

For regularly spaced sequences, often involving integers, it is simplest to use the colon operator. This can produce ascending or descending sequences:

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

Use the seq function to go from 0 up to 5 in steps of 0.5:

```
seq(0,5,0.5)
[1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Sequencing downwards

```
seq(5,0,-.5)
 [1] 5.0 4.5 4.0 3.5 3.0 2.5 2.0 1.5 1.0 0.5 0.0
```

Generating Repeated Sequence

- The rep() (or repeat) functon puts the Same constant into long vectors. The call form is rep(x,tmes).

```
> x <- rep(6,4)
> x
[1] 6 6 6 6
> rep(c(11,12,13),3)
[1] 11 12 13 11 12 13 11 12 13
> rep(1:2,3)
[1] 1 2 1 2 1 2
> rep(c(11,12,13),each=2)
[1] 11 11 12 12 13 13
```

Sorting, Ranking & Ordering

For example:

```
sales<-c(100,50,75,150,200,25)
```

Now apply the three different functions to the vector called sales

```
rank<-rank(sales)
sorted<-sort(sales)
ordered<-order(sales)
```

Make a dataframe using the four vectors :

```
view <-Data.frame(Sales,Ranks,Sorted,Ordered)
```

Sales	Ranks	Sorted	Ordered
100	4	25	6
50	2	50	2
75	3	75	3
150	5	100	1
200	6	150	4
25	1	200	5

Filtering Vectors

- ***Filtering with the subset() Functon***

```
> x <- c(6,1:3,NA,12)
> x
[1] 6 1 2 3 NA 12
> x[x > 5]
[1] 6 NA 12
```

- Subset functon handles NA :

```
> subset(x,x > 5)
[1] 6 12
```

Filtering Vectors

The Selecton Functon which()

In some cases, we may only want to find the positons within z at which the conditon occurs. It can be done by using which().

For example:

```
> z <- c(6,5,-3,7)
> which(z*z > 9)
[1] 1 2 4
```

The result says that elements 1, 2, and 4 of z have squares greater than 9.

However , the expression :

```
> z*z > 9
[1] TRUE TRUE FALSE TRUE
```

Else , to know the values , this expression will help :

- > B = z[z*z > 9]
- > B
- [1] 6 5 7

Filtering Vectors

- R also includes a vectorized version, the ifelse() functon. The form is : ifelse(b,u,v), where b is a Boolean vector, and u and v are vectors.
- The return value is itself a vector; element i is u[i] if b[i] is true

For example :

```
> x <- 1:10
> y <- ifelse(x %% 3 == 0, " Y ", " N ") # %% is the mod operator
> y
```

```
[1] "N" "N" "Y" "N" "N" "Y" "N" "N" "Y" "N"
```

Filtering Vectors

- Nested ifelse()

```
> h = c("A" "F" "F" "I" "M" "M" "F)
[1] "M" "F" "F" "I" "M" "M" "F"
> ifelse(h == "M",1,ifelse(h == "F",2,3))

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

Character Strings

In R, character strings are defined in double quotations:

```
a<-"hello"
b<-"55"
```

Numbers can be characters (as in b, above), but characters cannot be numbers.

```
as.numeric(a)
[1] NA
```

```
NAs introduced by coercion
Warning message: as.numeric(b)
[1] 55
```

One of the confusing things about character strings is the distinction between the length of a character object (a vector) and the numbers of characters in the strings comprising that object. Lets see an example to make the distinction clear:

```
sports<-
c(“badminton",“tabletennis",“cricket","basetball")
```

Character Strings

Here, pets is a vector comprising four character strings:44

```
length(sports)
[1] 4
```

and the individual character strings have 9,11,7 and 9 characters, respectively:

```
nchar(sports)
[1]  9 11  7  9
```

When first defined, character strings are not factors:

```
class(sports)
[1] "character"
is.factor(sports)
[1] FALSE
```

Character Strings

There are built-in vectors in R that contain the 26 letters of the alphabet in lower case (letters) and in upper case (LETTERS):

```
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
[17] "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"
[17] "Q" "R" "S" "T" "U" "V""W" "X" "Y" "Z"
```

To discover which number in the alphabet the letter k is, use the which function

```
which(letters=="k")
[1] 11
```

To suppress the quotes that appear around character strings by default use noquote:

```
noquote(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
```

To amalgamate strings into vectors of character information:

```
c(a,b)
[1] "hello" "55"
```

Using %in%

To know all of the matches between one character vector and another:

```
sports<-c('football','cricket','basketball','baseball')
popularsports<-
c('rugby','cricket','badminton','football','baseball','tennis','basketball')
```

To find the locations in the first-named vector of any and all of the entries in the second-named vector:

```
which(sports %in% popularsports)
[1]  1 2 3 4
```

To know what the matches are as well as where they are:

```
sports [which(sports %in% popularsports)]
[1] "football"   "cricket"    "basketball" "baseball"
```

Using paste

The R function to concatenate several strings into one string is paste:

```
paste(a,b,sep="")
[1] "hello55"
```

The third argument, sep="", means that the two character strings are to be pasted together without any separator between them: the default for paste is to insert a single blank space, like this:

```
paste(a,b)
[1] "hello 55"
```

```
paste(a,b,"a longer phrase containing blanks",sep="")
[1] " "hello55a longer phrase containing blanks"
```

If one of the arguments to paste is a vector, each of the elements of the vector is pasted to the specified character string to produce an object of the Same length as the vector:

```
d<-c(a,b,"new")
e<-paste(d,"a longer phrase containing blanks")
e
[1] "hello a longer phrase containing blanks"
[2] "55 a longer phrase containing blanks"
[3] "new a longer phrase containing blanks"
```

Using grep Function (Pattern matching)

- This functon is used to search for matches of patern within each element of a character vector and returns an integer vector of the elements of the vector that matches (if value is set to FALSE, which is the default).

- If value is set to TRUE, the contents of the matching elements of the character vector are returned.

```
> X = c(“apple","“potato","grape","10","blue.flower")
> grep("a",X)
[1] 1 2 3
> grep("a",X,value=TRUE)
[1] “apple” “potato” “grape”
> grep("[[:digit:]]",X,value=TRUE)
 [1] “10”
```

Using substr

- This functon can be used to index a porton of a string.

> word=”apples”
> substr(word,start=2,stop=5)
[1] “pple”

- The assignment operator (->) can be used to replace a porton of a string with another string

> substr(word,start=1,stop=1)<-”ban”
> word
[1] “bpple”

Using chartr

- Translates a list of old characters to the corresponding new characters. This is a vectorized functon and therefore can act on a character vector of any length

> word=”Apple”
> chartr(old=”A”,new=”a”,word)
[1] “apple”
> chartr(old=”Ae”,new=”aE”,word)
[1] “applE”

Using strsplit

- This functon is used to splits a string into a list containing multple strings, based upon a defined delimiter.
- The delimiter can be defined as a fixed character string or as a regular expression.

```
> word=”apple|lime|orange”
> v=strsplit(word,split=”|”,fixed=TRUE)
> v
[[1]]
[1] “apple” “lime” “orange”
> v[[1]][1]
[1] “apple”
```

Using sub and gsub

- sub() functon finds paterns within strings in a similar manner to grep(), but then substtutes the first instance of a match with a specified string.

```
> x=c(”apple”,”banana”,”grape”)
> sub(“a”,”$”,x)
[1] “$pple” “b$nana” “gr$pe”
```

- gsub() functon works in exactly the Same manner as sub(), but replaces all matches to patern rather than replacing only the first match.

```
 x=c(”apple”,”banana”,”grape”)
> gsub(“a”,”$”,x)
[1] “$pple” “b$n$n$” “gr$pe”
```

Using regexpr, gregexpr &

• regexpr() functon reports the character positon in the provided string(s) where the start of the match with patern occurs. The functon also returns the length of the match.

> x=c(”apple”,”grape”,”banana”)
> r=regexpr(“a”,x)
> r
[1] 1 3 2

• gregexpr(patern,text) is the Same as regexpr(), but it finds all instances of patern. Here’s an example:

> gregexpr(“ss",“Assessed")
[[1]]
[1] 2 5

• regmatches() functon can retrieve the matching components of a string vector for a provided *match object* produced by regxpr() > x=c(”apple”,”grape”,”banana”)

> r=regexpr(“a”,x)
> regmatches(x,r)
[1] “a” “a” “a”

Using sprintf

- This functon assembles a string from parts in a formated manner.

```
> i <- 2
> s <- sprint("the cube of %d is %d",i,i^3)
> s
[1] "the square of 2 is 8"
```

R Data Structures - Matrices

Matrices

- A *matrix* is a vector with two atributes
  - number of rows.
  - number of columns.
- Matrices can be defined as 2-dimensional arrays
- *All columns in a matrix must have the Same mode(numeric, character, etc.) and the Same length.*
- **To create a matrix is by using the matrix() functon:**
  - *myymatrix* <- matrix(*vector,* nrow=*number_of_rows*, ncol=*number_of_columns*, byrow=*logical_value*, dimnames=list(
  - *char_vector_rownames, char_vector_colnames*))

```
y <- matrix(c(1,2,3,4),nrow=2,ncol=2)
> y
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Matrix

```
M<-matrix(c(2,3,8,1,1,5,7,9,1),nrow=3)
print(M)
     [,1] [,2] [,3]
[1,]    2    1    7
[2,]    3    1    9
[3,]    8    5    1
```

where, by default, the numbers are entered column wise. The class and attributes of M indicate that it is a matrix of three rows and three columns (these are its dim attributes)

```
class(M)
[1] "matrix"
attributes(M)
$dim
[1]    3      3
```

For transpose of a Matrix:

```
Minv = t(M)
print(Minv)
     [,1]  [,2]  [,3]
[1,]    2     3     8
[2,]    1     1     5
[3,]    7     9     1
```

Matrix

Naming the rows and columns of matrices

At first, matrices have numbers naming their rows and columns (see above). Here is a 3×4 matrix of marks obtained in each quarterly exams for 4 different subjects

```
X<-matrix(c(20,10,80,50,60,100,90,30,40,70,50,60),nrow=3)
X
     [,1] [,2] [,3] [,4]
[1,]   20   50   90   70
[2,]   10   60   30   50
[3,]   80  100   40   60
```

We employ the function rownames() to change names of rows

```
rownames(X)<-rownames(X
,do.NULL=FALSE,prefix="Test.")
X
       [,1]  [,2]  [,3]  [,4]
Test.1   20    50    90    70
Test.2   10    60    30    50
Test.3   80   100    40    60
```

Matrix

For the columns we want to supply a vector of different names for the four subs involved in the test, and use this to specify the colnames(X):

```
subs<-c("Maths", "English", "Science", "History")
colnames(X)<-subs
X
        Maths English Science History
Test.1    20      50      90      70
Test.2    10      60      30      50
Test.3    80     100      40      60
```

Calculations On Rows Or Columns Of The Matrix

We can use subscripts to select parts of the matrix, with a blank meaning 'all of the rows' or 'all of the columns'. Here is the mean of the rightmost column (number 4)

```
X<-matrix(c(50,60,40,90,100, 80,50, 90,10, 80,30, 70),nrow=3)
 mean(X[,4])
[1] 60
```

calculated over all the rows (blank then comma), and the standard deviation of the bottom row,

```
sd(X[3,])
[1] 31.62278
```

There are some special functions for calculating summary statistics on matrices:

```
rowSums(X)
[1] 270 280 200
colSums(X)
[1] 150 270 150 180
 rowMeans(X)
[1] 67.5 70.0 50.0
colMeans(X)
[1] 50 90 50 60
```

Calculations On Rows Or Columns Of The Matrix

In this particular case we have been asked to add a row at the bottom showing the column means, and a column at the right showing the row variances:

```
X<-rbind(X,apply(X,2,mean))
X<-cbind(X,apply(X,1,var))
X
      [,1] [,2] [,3] [,4]     [,5]
[1,]   50   90    50   80    425.0000
[2,]   60  100   90   30    1000.0000
[3,]   40   80   10   70    1000.0000
[4,]   50   90   50   60     358.3333
>
```

Note : that the number of decimal places may vary across columns, with fifth columns The default in R is to print the minimum number of decimal places consistent with the contents of the column as a whole.