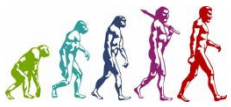
R Data Structure



Defining Data Structures in R

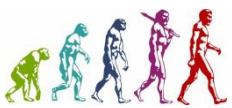
The entities that R creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components.

A Data Structure is a particular way of organizing and storing data in a computer so that it can be accessed and modified efficiently. They are a type of object.

R operates on named Data Structures

R's data structures include vectors, matrices, arrays, data frames (similar to tables in a relational database), factors and lists.

The scalar data type was never a data structure of R. Instead, a scalar is represented as a vector with length one

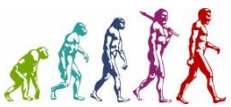


Types of Data Structure

Mode: All objects have a certain mode. Some objects can deal with only one mode at a time, others can store elements of multiple modes. Mode types: Integer, Numeric, Complex, Character, Logical

R's data structures include the following:

- Vectors – Basic Data Structure, one dimensional and all elements of same mode
- Matrices – 2 dimensional rectangular objects of same mode
- Arrays – Higher (>2) dimensional rectangular object of same mode
- Data Frames – Special Matrices, Two dimensional containers with rows and columns corresponding to observations and variables respectively
- Lists – Like vectors but do not have to contain elements of same mode
- Factors – Vector to classify Categorical data



Numeric Vectors

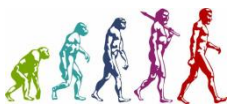
- The simplest structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers

- To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

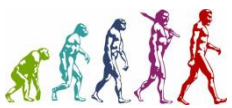
- This is an assignment statement using the function c()

- A number occurring by itself in an expression (like scalar) is taken as a vector of length one



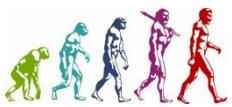
Vector Arithmetic

- Vectors can be used in Arithmetic operations
- Operations are performed element by element
- Multiple vectors in the same expression may not be of same length
- Shorter vectors are recycled as need be
- Value of expression has the same length as longest vector
- All vector operations and common arithmetic functions are available for Numeric Vectors



Vector Arithmetic

- Some more functions can be applied to Vectors:
  - length(x) #Number of elements
  - min(x)
  - max(x)
  - range(x) # returns min and max
  - var(x) # Variance
  - mean(x)
  - median(x)
  - sort(x) # Ascending order
  - sort(x, decreasing = TRUE) #Descending order
  - sum(x)
  - prod(x) # Product



Colon Operator

- Using c() function to create a vector is tedious so colon operator can be used to create an integer vector

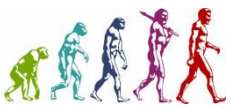
```
> 1:20
```

- Similarly seq() function can be used which is more general

```
> seq(from = 5, to = 10)    # 5 6 7 8 9 10
> seq(5,10)                 # 5 6 7 8 9 10
> seq(0,10,by =2)           # 0 2 4 6 8 10
> seq(0,10,length.out = 11)  # 0 1 2 3 4 5 6 7 8 9 10, Number of
                              #output elements provided in length.out
```

- rep() allows to repeat things

```
> rep(0, times = 10)
> rep(1:5, times = 2)
> rep(1:5, each = 2)
```

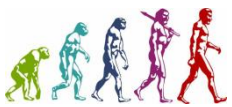


Logical Vectors

- As well as numerical vectors, R allows manipulation of logical quantities
- The elements of a logical vector can have the values TRUE, FALSE, and NA
- Logical vectors are generated by conditions. For example

```
> temp <- x > 13
```

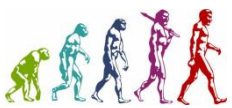
sets temp as a vector of the same length as x with values FALSE corresponding to elements of x where the condition is not met and TRUE where it is



Missing Values (NA)

- In some cases the components of a vector may not be completely known.
- When an element or value is “not available” or a “missing value” in the statistical sense, a place within a vector may be reserved for it by assigning it the special value NA.
- In general any operation on an NA becomes an NA.
- The motivation for this rule is simply that if the specification of an operation is incomplete, the result cannot be known and hence is not available.
- The function is.na(x) gives a logical vector of the same size as x with value TRUE if and only if the corresponding element in x is NA.

```
> z <- c(1:3,NA);        ind <- is.na(z)
```

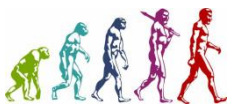


Character Vector

- Character quantities and character vectors are used frequently in R, for example as plot labels.

- Where needed they are denoted by a sequence of characters delimited by the double quote character, e.g., "x-values", "New iteration results".

- Character strings are entered using either matching double (") or single (’) quotes, but are printed using double quotes (or sometimes without quotes).



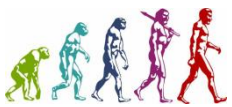
Index Vectors

- Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets. E.g. x[6] is the sixth component of x

- More generally any expression that evaluates to a vector may have subsets of its elements similarly selected by appending an index vector in square brackets immediately after the expression e.g.

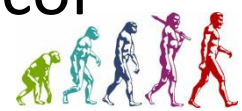
```
> x[1:10]
```

selects the first 10 elements of x (assuming length(x) is not less than 10).



Matrices

- A Matrix is an extension of vectors in two dimension
- A Matrix is used to represent two dimensional data of single type (single mode)
- A clean way to generate Matrix is to use matrix() function. It takes the following arguments:
  - data an R Object (could be vector)
  - nrow desired number of rows
  - ncol desired number of rows
  - byrow to populate either by column (default) or by row
- It is also possible to transform another data structure into matrix form using as.matrix() function
- Using [i j] will retrieve the element for jth row, jth col



Solution of system of linear equation using inverse of a matrix

$$a_1 x + b_1 y + c_1 z = d_1$$
$$a_2 x + b_2 y + c_2 z = d_2$$
$$a_3 x + b_3 y + c_3 z = d_3$$

$$\text{Let } A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_2 & b_2 & c_3 \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{and } B = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$AX = B$$

$$X = A^{-1}B$$

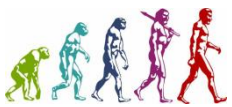
Arrays and Matrices

- An Array is an extension of vectors in more than two dimension of single type (single mode)
- A vector can be used by R as an array only if it has a dimension vector as its dim attribute.
- Suppose, for example, z is a vector of 150 elements. The assignment

```
> dim(z) <- c(3,5,10)
```

gives it the dim attribute that allows it to be treated as a 3 by 5 by 10 array.

- array() is available for simpler and more natural looking assignments
- if the dimension vector for an array, say a, is c(3,4,2) then there are 3 X 4 X 2 = 24 entries in a and the data vector holds them in the order a[1,1,1], a[2,1,1], ...,a[2,4,2], a[3,4,2].



Arrays

- As well as giving a vector structure a dim attribute, arrays can be constructed from vectors by the array function, which has the form

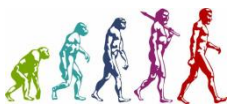
  > Z <- array(data_vector, dim_vector)

- For example, if the vector h contains 24 or fewer, numbers then the command

  > Z <- array(h, dim=c(3,4,2))

- would use h to set up 3 by 4 by 2 array in Z. If the size of h is exactly 24 the result is the same as

  > Z <- h ; dim(Z) <- c(3,4,2)



Lists

- *Lists* are a general form of vector in which the various elements need not be of the same type, and are often themselves vectors or lists.
- Lists provide a convenient way to return the results of a statistical computation.

```
> Lst <- list(name="Fred", wife="Mary",  no.children=3,
child.ages=c(4,7,9))
```

- An R list is an object consisting of an ordered collection of objects known as its *components*.
- Components are always numbered and may always be referred to as such. Thus if Lst is the name of a list with four components, these may be individually referred to as Lst[[1]], Lst[[2]], Lst[[3]] and Lst[[4]].
- If, further, Lst[[4]] is a vector subscripted array then Lst[[4]][1] is its first entry.

Factor

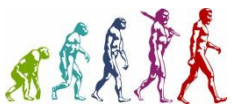
- A factor is an ordered collection of categorical items. The different values that a factor can take are called its levels.

```
> eye.colours <- factor(c("brown", "blue", "black", "blue", "brown",
                          "black", "green", "black"))
```

- levels() function shows all the levels of a factor

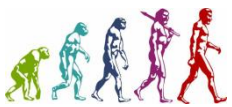
```
> levels(eye.colours)

[1] "black" "blue" "brown" "green"
```

- In above example, the levels do not have any order. We can have ordered factors where levels are ordered by setting attribute ordered=TRUE
- It is possible to take an integer array and turn it into an integer array by using function unclass()



Understanding Data Frames

- A data frame represent a table of data.
- Each column may have different data types but each row in the data frame must have same length
- Usually, each column is named. Sometimes rows are named as well using row.names
- The columns are often referred to as “variables” and are tightly coupled
- It shares many properties of matrices and of Lists and to extract a column, $ is used (like lists).
- It is used as a fundamental data structure by most of R’s modeling s/w
- Data Frames are usually created by calling read.table() or read.csv() function



Packages

- All R functions and datasets are stored in packages. Only when a package is loaded are its contents available. This is done both for efficiency and to aid package developers, who are protected from name clashes with other code.

- To see which packages are installed at your site, issue the command

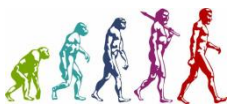
  > library()

- To see which packages are currently loaded, use

  > search()

- To load a particular package (e.g., the boot) use a command

  >library(boot)

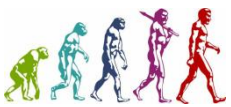


Understanding dplyr Package

- dplyr package helps in manipulating data frames

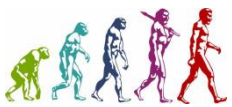
dplyr verbs:

- select : returns a subset of the columns of a data frame
- filter : extract a subset of rows from a data frame based on logical conditions
- arrange : reorder rows of a data frame
- rename: rename variables in a data frame
- mutate : add new variables/columns or transform existing variables
- summarise/summarize : generate summary statistics of different variables in the data frame, possibly within strata



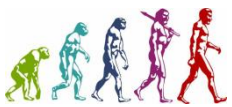
Looping in command line – apply functions

- Writing for, while loops is useful when programming but not particularly easy when working interactively on the command line.
- There are some functions which implement looping in the command line to make life easier
  - lapply : Loop over a list and evaluate a function on each element
  - sapply : Same as lapply but try to simplify the results
  - apply : Apply a function over the margins of an array
  - tapply : Apply a function over subsets of a vector
  - mapply : Multivariate version of lapply
- An auxiliary function split is also useful, particularly in conjunction with lapply



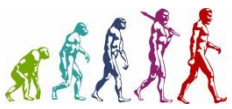
lapply, sapply and vapply Function

- lapply takes a list and a function. It loops over the list and apply this function over the elements of the list.
- If input is not a list then it will be coerced to a list.
- lapply always returns a list
- sapply will try to simplify the result of lapply wherever possible
- If the result is a list where every element is length 1 then a vector is returned
- If the result is a list where every element is a vector of the same length (>1) then a matrix is returned
- If it can’s figure things out then a list is returned
- vapply is similar to sapply but has a pre-specified type of return value so it is safer to use



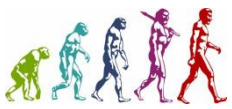
apply() Function

- apply is used to evaluate a function over the margins of an array
- It is most often used to apply a function to the rows and columns of a matrix
- It can be used with general arrays; e.g. taking the average of an array of matrices
- It is not faster than a loop, but it works in one line



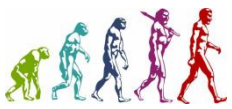
tapply() Function

- Is used to apply a function over subset of a vector.
- tapply(x, INDEX, FUN, simplify)
  - x is the vector on which to apply tapply
  - INDEX is a factor or a list of factors which identify subset of x
  - FUN is the function to apply
  - simplify : should we simplify the results (default is TRUE)



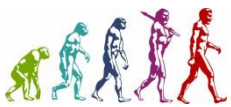
mapply() Function

- mapply is a multivariate apply which applies a function in parallel over a set of arguments
- mapply(FUN, ARGS, FUN_ARGS, SIMPLIFY)
- FUN is the function to apply
- ARGS is the arguments to apply over
- FUN_ARGS are the function arguments
- SIMPLIFY indicate whether results need to be simplified



Managing Data with R

- Entering data using R command
- Entering data using GUI
- Saving and Loading R objects
- Importing data from External files
- Importing data from other sources
- Exporting data



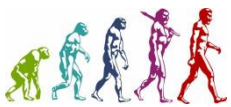
Objects in R

- The entities that R creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components.
- During an R session, objects are created and stored by name
- The R command

    > objects() or alternatively, ls()

can be used to display the names of the objects which are currently stored within R.

- The collection of objects currently stored is called the *workspace*.

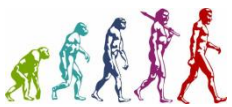


Saving Objects in R

- All objects created during an R session can be stored permanently in a file for use in future R sessions

```
> save(student, file = "C:/Users/Abhinav Srivastava/Documents/student.RData")
```

- Multiple object may be saved to same file by listing them under same save command
- To save every object in workspace use save.image() function
- At the end of each R session you are given the opportunity to save all the currently available objects
- If you indicate that you want to do this, the objects are written to a file called .Rdata in the current directory, and the command lines used in the session are saved to a file called .Rhistory.



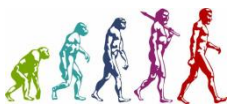
Removing and Loading Objects in R

- To remove objects the function rm() is available:

  > rm(student,student.name,roll.no)

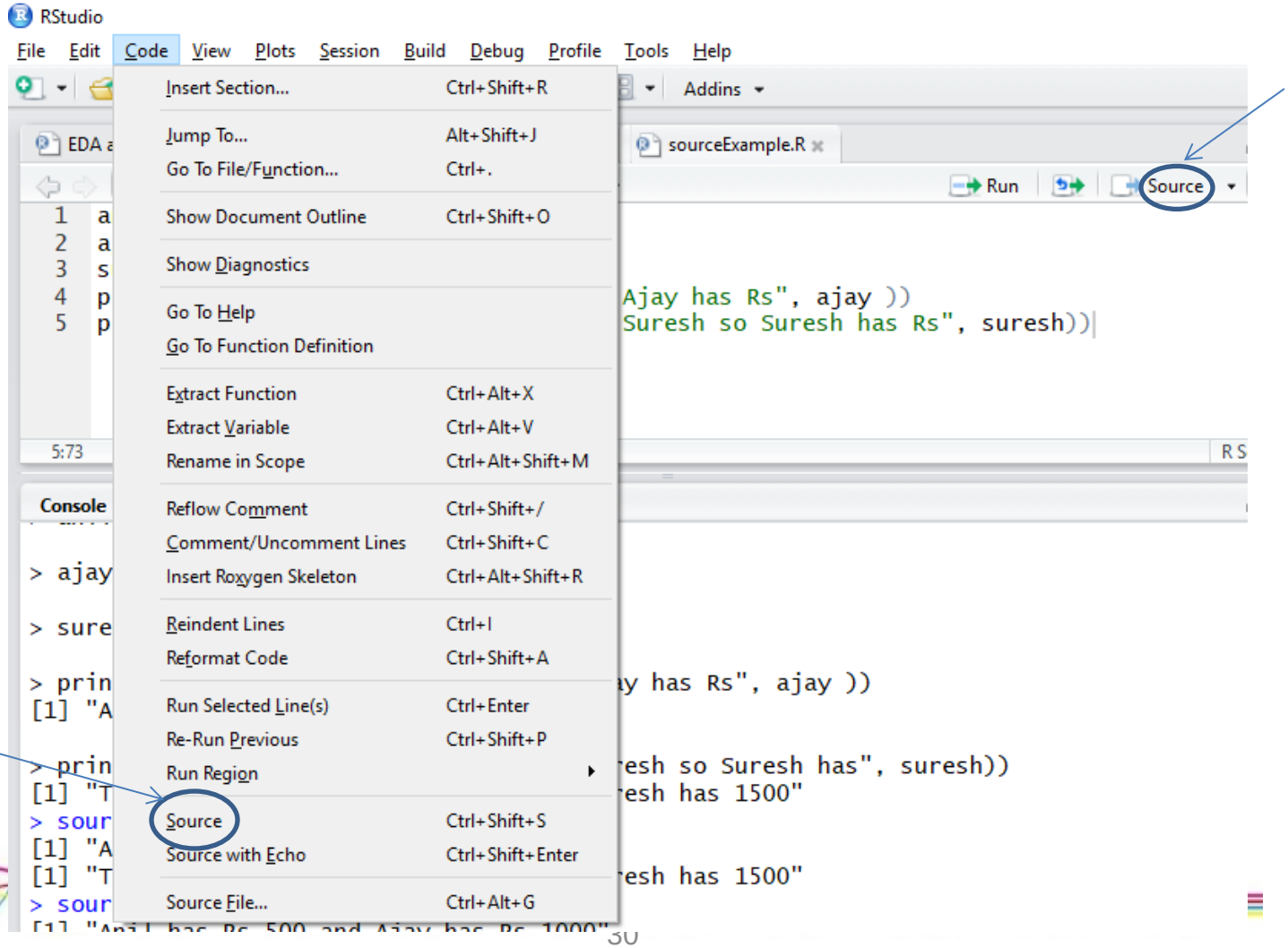
- For loading r objects from .Rdata file, function load() is available:

  > load("~/student1.RData")



How to run an R Script

- An R script can be run using following menu option in R Studio

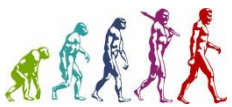


Executing Command from File (Batch Script)

- If commands are stored in an external file, say commands.R in the working directory work, they may be executed at any time in an R session with the command

```
> source("commands.R")
```

- For Windows *Source* is also available on the *File* menu.



Diverting output to File

- The function sink,

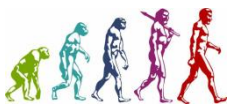
  > sink("record.lis")

  will divert all subsequent output from the console to an external file, record.lis.

- The command

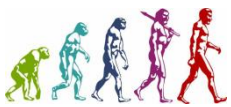
  > sink()

  restores it to the console once again.



How to import files in R?

- It is easy to pull data from other programs in R
- R can import data from text files, spreadsheets and other statistics software
- You don’t even need a local copy of the file. You can specify a file at a URL and R will fetch the file for you over internet



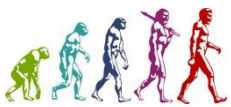
Scan () function

- Read data into a vector or list from the console or file
- “inline” usage from console

```
> scan(text = "1 2 3")
Read 3 items
[1] 1 2 3
```

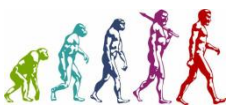
- To read data from file:

```
> numb.in.file<- scan("numb_in_file.txt", what = "character", nmax = 5, sep = " ", skip = 1, nlines = 1)
```



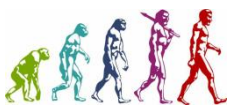
Importing Table and CSV files

- Most text files containing data are formatted similarly: each line of a text file represents an observation (or record)
- Each line contains a set of different variables associated with the observation
- Sometimes, different observations are separated by a special character called a delimiter (delimited files)
- Other times, variables are differentiated by their location on each line (fixed-width files)
- **Delimited file**: R includes a family of functions for importing delimited text files into R based on read.table() function
- The read.table() function reads a text file into R and returns a data.frame object.
- Each row is interpreted as an observation and each column a variable. Each field is separated by a delimiter



read.table() function

- This is one of the most commonly used functions for reading data. It has few important arguments:
  - file: Name of file or connection
  - header: logical indicating if the file has a header line (default false)
  - sep: a string indicating how the columns are separated
  - colClasses: c Character vector indicating the class of each column in the dataset
  - nrows: the maximum number of rows in the dataset to be read
  - comment.char: a character string indicating the comment character
  - skip: the number of lines to skip from the beginning
  - stringsAsFactors: should character variable be coded as factors?
- For moderate sized datasets, you can directly call read.table directly with only file argumant. R will figure out most things automatically
- read.csv() is identical to read.table except that the default separator is comma and header is TRUE



How to import an Excel file, Minitab file

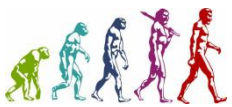
One of the best approaches to import data from Excel, SQL Databases and other software is to export the data to a text file (e.g. csv file) and then import into R using read.table() or read.csv()

Import from Excel file:

```
>library(xlsx)
>emp.data<-read.xslx("./emp.xlsx",sheetIndex=1,header=TRUE)
>head(emp.data)
```

Import from Minitab file:

Use read.mtp() function



Importing Data from SQL Databases

SQL Databases (MySQL):

> library(RMySQL)

Connecting and listing Databases

>myDB <- dbConnect(MySQL(), user = “abc”, host = “abc.myorg.com”)

> result <-dbGetQuery(myDB,”show databases;”); dbDisconnect(myDB)

Listing Tables and Fields

> db19 <- dbConnect(MySQL(), user = “abc”, db=“db19”, host = “abc.myorg.com”)

> allTables <- dbListTables(db19)

> dbListFields(db19, “tab5”)

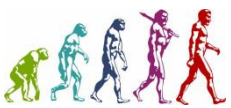
Getting Query results

> dbGetQuery(db19,”select count(*) from tab5”)

> query<- dbSendQuery(db19, “select * from tab5 where fld3 between 1 and 3”)

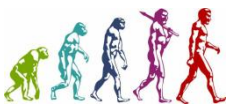
> affyMis <- fetch(query); quantile(affyMis$fld)

> affyMisSmall <- fetch(query, n=10); dbClearResult(query); dbDisconnect(db19)



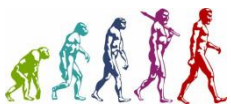
How to export files from R?

- R can also export R data objects (usually data frames and matrices) as text files.
- To export data to a text file, use the write.table function
- The main arguments of write.table function are
  - x : object to export
  - file : Character value specifying filename or connection object to which output is written
  - append : To append (TRUE) or replace the file (FALSE)
  - sep : character separating values in a row
  - col.names : Logical value specifying whether to include column names
- Use write.xslx() function to export data directly from R to excel



Thank You

Abhinav Srivastava

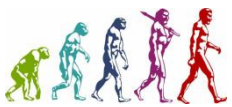


Objects their modes and attributes

- The entities R operates on are technically known as objects. Examples are vectors of numeric (real) or complex values, vectors of logical values and vectors of character strings.
- These are known as “atomic” structures since their components are all of the same type, or mode, namely numeric, complex, logical, character and raw.
- By the mode of an object we mean the basic type of its fundamental constituents.
- Vectors must have their values all of the same mode. Thus any given vector must be un-ambiguously either logical, numeric, complex, character or raw. (The only exception is NA)
- R also operates on objects called lists, which are of mode list. These are ordered sequences of objects which individually can be of any mode.
- The other recursive structures are those of mode function and expression.
- Another property of every object is its length. The functions mode(object) and length(object) can be used to find out the mode and length of any defined structure

Objects their modes and attributes

- Further properties of an object are usually provided by attributes(object). Because of this, mode and length are also called “intrinsic attributes” of an object.
- The function attributes(object) returns a list of all the non-intrinsic attributes currently defined for that object.
- The function attr(object, name) can be used to select a specific attribute.
- These functions are rarely used, except in rather special circumstances when some new attribute is being created for some particular purpose, for example to associate a creation date or an operator with an R object.



Functions

- functions are themselves objects in R which can be stored in the project's workspace. This provides a simple and convenient way to extend R.

- A function is defined by an assignment of the form

  > name <- function(arg_1, arg_2, ...) expression

- These are true R functions that are stored in a special internal form and may be used in further expressions and so on.

- It should be emphasized that most of the functions supplied as part of the R system, such as mean(), var(), postscript() and so on, are themselves written in R and thus do not differ materially from user written functions.