What is R Programming

Unit I: Introduction: R interpreter, Introduction to major R data structures like vectors, Matrices, arrays, list and data frames, Control Structures, vectorized if and multiple Selections, functions.

"R is an interpreted computer programming language which was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand." The R Development Core Team currently develops R. It is also a software environment used to analyze statistical information, graphical representation, reporting, and data modeling. R is the implementation of the S programming language.

History of R Programming

The history of R goes back about 20-30 years ago. R was developed by Ross lhaka and Robert Gentleman in the University of Auckland, New Zealand, and the R Development Core Team currently develops it. This programming language name is taken from the name of both the developers. The first project was considered in 1992. The initial version was released in 1995, and in 2000, a stable beta version was released.

Features of R programming

R is a domain-specific programming language which aims to do data analysis. It has some unique features which make it very powerful. The most important arguably being the notation of vectors. These vectors allow us to perform a complex operation on a set of values in a single command. There are the following features of R programming:

- ✓ It is a simple and effective programming language which has been well developed.
- ✓ It is data analysis software.
- ✓ It is a well-designed, easy, and effective language which has the concepts of user-defined, looping, conditional, and various I/O facilities.
- ✓ It has a consistent and incorporated set of tools which are used for data analysis.
- ✓ For different types of calculation on arrays, lists and vectors, R contains a suite of operators.
- ✓ It provides effective data handling and storage facility.
- ✓ It is an open-source, powerful, and highly extensible software.
- ✓ It provides highly extensible graphical techniques.
- ✓ It allows us to perform multiple calculations using vectors.
- ✓ R is an interpreted language.

Why use R Programming?

There are several tools available in the market to perform data analysis. Learning new languages is time taken. The data scientist can use two excellent tools, i.e., R and Python. We may not have time to learn them both at the time when we get started to learn data science. Learning statistical modeling and algorithm is more important than to learn a programming language. A programming language is used to compute and communicate our discovery.

The important task in data science is the way we deal with the data: clean, feature engineering, feature selection, and import. It should be our primary focus. Data scientist job is to understand the data, manipulate it, and expose the best approach. For machine learning, the best algorithms can be implemented with R. Keras and TensorFlow allow us to create high-end machine learning techniques. R is a great tool to investigate and explore the data. The elaborate analysis such as clustering, correlation, and data reduction are done with R.

Data science deals with identifying, extracting, and representing meaningful information from the data source. R, Python, SAS, SQL, Tableau, MATLAB, etc. are the most useful tools for data science. R and Python are the most used ones

Difference between R and Python

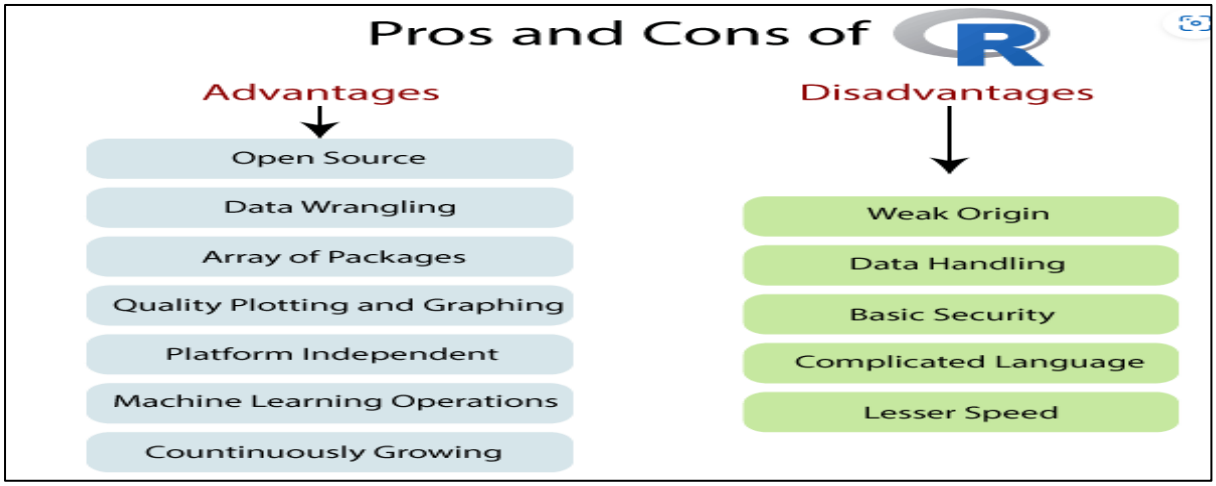
R Programming

- ✓ "R is an interpreted computer programming language. R is also a software environment which is used to analyze statistical information, graphical representation, reporting, and data modeling.
- ✓ R packages have advanced techniques which are very useful for statistical work.
- ✓ For data analysis, R has inbuilt functionalities
- ✓ Data visualization is a key aspect of analysis. R packages such as ggplot2, ggvis, lattice, etc. make data visualization easier.
- ✓ There are hundreds of packages and ways to accomplish needful data science tasks.

Python Programming

- ✓ Python is an Interpreted high-level programming language used for general-purpose programming. Guido Van Rossum created it, and it was first released in 1991. Python has a very simple and clean code syntax. It emphasizes the code readability and debugging is also simple and easier in Python.
- ✓ For finding outliers in a data set both R and Python are equally good. But for developing a web service to allow peoples to upload datasets and find outliers, Python is better.
- ✓ Most of the data analysis functionalities are not inbuilt. They are available through packages like Numpy and Pandas
- ✓ Python is better for deep learning because Python packages such as Caffe, Keras, OpenNN, etc.
- ✓ Python has few main packages such as viz, Sccikit learn, and Pandas for data analysis of machine learning.

R Advantages and Disadvantages



R is the most popular programming language for statistical modeling and analysis. Like other programming languages, R also has some advantages and disadvantages. It is a continuously evolving language which means that many cons will slowly fade away with future updates to R.

Pros/Advantages

1) Open Source

An open-source language is a language on which we can work without any need for a license or a fee. R is an open-source language.

2) Platform Independent

R is a platform-independent language or cross-platform programming language which means its code can run on all operating systems. R enables programmers to develop software for several competing platforms by writing a program only once. R can run quite easily on Windows, Linux, and Mac.

3) Machine Learning Operations

R allows us to do various machine learning operations such as classification and regression. For this purpose, R provides various packages and features for developing the artificial neural network. R is used by the best data scientists in the world.

4) Exemplary support for data wrangling

R allows us to perform data wrangling. R provides packages such as dplyr, readr which are capable of transforming messy data into a structured form.

5) Quality plotting and graphing

R simplifies quality plotting and graphing. R libraries such as ggplot2 and plotly advocates for visually appealing and aesthetic graphs which set R apart from other programming languages.

6) The array of packages

R has a rich set of packages. R has over 10,000 packages in the CRAN repository which are constantly growing. R provides packages for data science and machine learning operations.

7) Statistics

R is mainly known as the language of statistics. It is the main reason why R is predominant than other programming languages for the development of statistical tools.

8) Continuously Growing

R is a constantly evolving programming language. Constantly evolving means when something evolves, it changes or develops over time. R is a state of the art which provides updates whenever any new feature is added.

Cons/Disadvantages:

1) Data Handling

In R, objects are stored in physical memory. It is in contrast with other programming languages like Python. R utilizes more memory as compared to Python. It requires the entire data in one single place which is in the memory.

2) Basic Security

R lacks basic security. It is an essential part of most programming languages such as Python. Because of this, there are many restrictions with R as it cannot be embedded in a web-application.

3) Complicated Language

R is a very complicated language, and it has a steep learning curve. The people who don't have prior knowledge or programming experience may find it difficult to learn R.

4) Weak Origin

The main disadvantage of R is, it does not have support for dynamic or 3D graphics. The reason behind this is its origin. It shares its origin with a much older programming language "S."

5) Lesser Speed

R programming language is much slower than other programming languages such as MATLAB and Python. In comparison to other programming language, R packages are much slower.

In R, algorithms are spread across different packages. The programmers who have no prior knowledge of packages may find it difficult to implement algorithms.

Identifiers

Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.

- ❖ Rules for writing Identifiers

1. Identifiers can be a combination of letters, digits, period (.) and underscore (_).
2. It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
3. Reserved words in R cannot be used as identifiers.

Example:

Valid identifiers

Sum, .fine.with.dot, this_is_acceptable, Number5

Invalid identifiers

tot@l, 5um, _fine, TRUE, .0ne

Below are the list of the identifiers

1. Variable
2. Constants
3. Symbolic constant
4. key words
5. Data Types
6. Data Structure.

1. Variable

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number. Variables are used to store data.

Assigning Values to Variabels: In R an assignment to a variable can be done in three ways:
Using =,<-(leftword), ->(Rightword) signs.The Assignment operator (<-) consists of the two characters, "<"(Less than) and "-"(minus) occurring strictly side-by-side.

Examples:

```
>#Defining variable "myfirst" and assigning "Hello,World" to it
>myfirst<-"Hello,world!"
>#viewing the value of the variable
>myfirst
[1] "Hello,world!"
>#Left Assignment in R
>X<-20
>#Right Assignment in R
>20->X
>#Assignment using equal sign
>x=20
```

Types of variable

1) **Boolean Variables:** This is the simplest type of variable. It contains a single bit, and indicate a binary result (0 and 1, yes and no, or true and false).

e.g. a = TRUE

b = FALSE

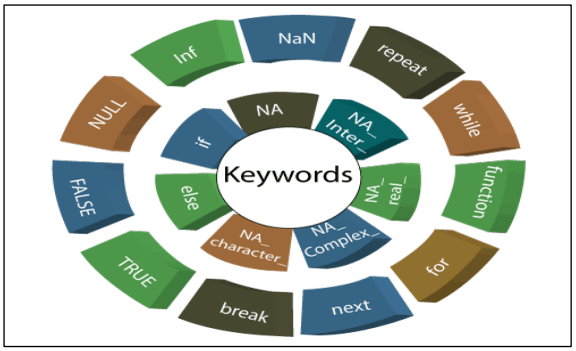
2) Integer variables: Numbers with no floating point are called integers

3) Numeric Variables: Numeric variables are used to store numbers. It can contain floating point numbers.
e.g. a = 1
b = 3.14
4) Characters Variables: Character variables are used to store non-numeric data. Unlike other programming languages, there are a no differences between characters and strings in R.
e.g. a = "x"
b = "6"
5) String Variables: String variables are those variables which contain one or more characters.
e.g. x= "abcd2"
y= "Hello World"
name= "x"
Removing Variables: It is good practice to remove the variable names at the end of each session in R.
Syntax: rm(i,j,- -)
Where i,j,….. are names of variables separated by comma.
>#Variable can be deleted by using the rm( ) function
>rm(new3)
># A deleted variable will throw an error if printed
>new3
Error:Object 'new3' not found
2. Constants
Constants are entities within a program whose value can't be changed. There are 2 basic types of constant. These are numeric constants and character constants.
1) **Numeric Constants:** numeric constants are the numbers which can be integer, double or complex. You can check the type of constant through the typeof() function. Numeric constant suffix with ***L*** are the integer type and suffix with ***i*** are called complex type.

e.g. > typeof(10)
[1] "double"
> typeof(10L)
[1] "Integer"
> typeof(10i)
[1] "complex"

2) **Character Constant:** Character constant can be declared using either single quote (' ') or double quote (" ").
e.g. > typeof("CUAP")
[1] "character"
> typeof('CUAP')
[1] "character"
3) **Built-in Constants:** Some of the built-in constants of R along with their values are shown below:
e.g. > LETTERS
>letters
>pi
>month.name
>month.abb

3. key words



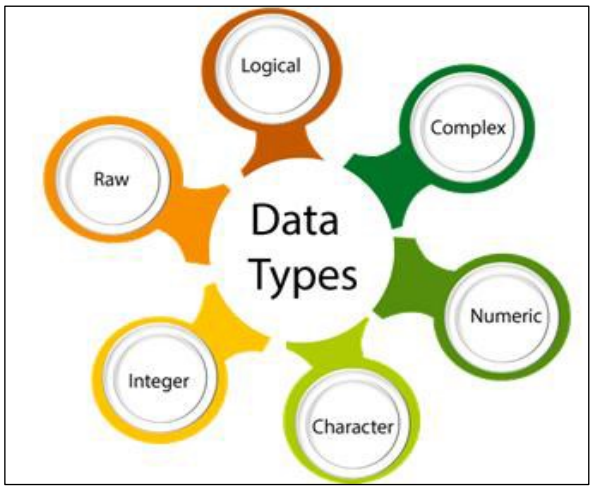
In programming, a keyword is a word which is reserved by a program because it has a special meaning. A keyword can be a command or a parameter. Like in C, C++, Java, there is also a set of keywords in R. A keyword can't be used as a variable name. Keywords are also called as "reserved names."

4. Data Types

In programming languages, we need to use various variables to store various information. Variables are the reserved memory location to store values. As we create a variable in our program, some space is reserved in memory.

In R, there are several data types such as integer, string, etc. The operating system allocates memory based on the data type of the variable and decides what can be stored in the reserved memory.

There are the following data types which are used in R programming:



1) Numeric Data Type :

The numeric data type is for numeric values. It is the default data type for numbers in R. Examples of numeric values would be 1, 34.5, 3.145, -24, -45.003, etc.

e.g. > num <- 1

> **class**(num)

[1] “numeric”

> typeof(num)

[1] “double”

Note: When R stores a number in a variable, it converts the number into a 'double' value or a decimal type with at least two decimal places. This means that a value such as '1' is stored as 1.00 with a type of double and a class of numeric.

2) Integers Data Type :

The Integer data type is used for integer values. To store a value as an integer, we need to specify it as such. The integer data type is commonly used for discrete only values like unique ids. We can store as well as convert a value into an integer type using the as.integer() function.

e.g,

> int <- as.integer(16)

> class(int)

[1] "integer"

> typeof(int)

[1] "integer"

> num=1

> int2 <- as.integer(num)

> int2

[1] 1

> class(int2)

[1] "integer"

> typeof(int2)

[1] "integer"

3) Complex Data Type :

The complex data type is to store numbers with an imaginary component. Examples of complex values would be 1+2i, 3i, 4-5i, -12+6i, etc.

e.g > comp <- 22-6i

> class(comp)

[1] "complex"

> typeof(comp)

[1] "complex"

4) Logical Data Type :

The logical data type stores logical or boolean values of TRUE or FALSE.

e.g.

> logi <- FALSE

> class(logi)

[1] "logical"

> typeof(logi)

[1] "logical"

5) Character Data Type :

The character data type stores character values or strings. Strings in R can contain the alphabet, numbers, and symbols.

Example:

> name <- "CUAP"

> class(name)

[1] "character"

> typeof(name)

[1] "character"

Code :

```
comp <- 22-6i

int2 <- as.integer(comp)

int2

char2 <- as.character("hello")

char3 <- as.character(comp)

char2

char3

num2 <- as.numeric(int)

num2

int4 <- as.integer(num)

int4

comp2 <- as.complex(num)

comp2

comp2 <- as.complex(num)

char2 <- as.character(num)
```

Input of Data in R:

Data can be input directly from the terminal during run-time.The scan( ) function is used to take data from the user at the terminal.The use of scan() function is to read a vector of numbers.

```
>#  Reading one set of numeric data
>x<-scan()
1:23
2:40
3:47
4:40
>#Displaying the numeric variable
>x
[1] 23 40 47 40
>#Reading character data,data use new line character as separator
>string1<-scan(what=" ",sep="\n")
1:Hello
2:How are you
3:Thanks
4:Bye
5:See you
>#Displaying character variable
>string1
"Hello"  " How are you"  "Thanks"  "Bye"  "See you"
```

Output in R:

Output of the data is important after processing any data.Display of output in R can be done using functions such as print,cat,paste

Print( ) function

The print( ) function is used to display the output of the program
Syntax: print("Output Message/variablename")

```
>#use of print( ) function
>print("Hello world and welcome to R")
>print("welcome to the world of R programming")
>#Printing multiple strings using multiple print statements
>print("Hello");print("and");print("welcome");
>#stroing a string in a variable
>hellostring<-"Hello,How are you"
>#use of print( ) function for printing variable
>print(hellostring)
```

The cat( ) function:

The cat( ) function is an alternative to print that lets you combine multiple items into a continuous output.
Syntax: cat(s1,s2,……,sep= " ",…)

```
>#use of cat( ) function to display two strings together
>cat("Hello","Welcome")
[1] Hello Welcome
```

>#Joining multiple strings using cat( ) function

```
>tdate<-date( )
>cat("Today is",tdate)
```

5. Data Structure.

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values.

R's base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the six data types which are most frequently utilized in data analysis.

The most essential data structures used in R include:

1. Vectors
2. Lists
3. Data frames
4. Matrices
5. Arrays
6. Factors

1. Vectors

A vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures.

```
e.g. > X = c(1, 3, 5, 7, 8)
> X
[1] 1 3 5 7 8
> length(X)
[1] 5
> class(X)
[1] "numeric"
```

Class of a Vector

The different classes of vectors include logical,numeric,complex,character,integer and raw.

1.Logical Vector:R allows manipulation of logical quantities.The elements of a logical vector can have either of the three values:True,False and NA(Not Available)

2.Character Vector:They are denoted by a sequence of characters delimited by the double quote characters example "India","CUAP"

3.Numeric Vector: This vector is commonly used in R .

4.Integer Vector: An Integer Vector is denoted by a special alphabet in last.

Example: L in last will denote an integer vector.

5.Complex Vector: This vector has an imaginary term also along with the real term.

Example:z=a+bi

6.Raw Vector:A vector can be converted to a raw vector with available inbuilt functions of R

Elements of a Vector:

The vector can store one single item or multiple items.A vector of multiple elements can be created using seven different ways.

1.Concatenation[c( )] functions:A Vector of multiple elements can be created using the c ( ) function.

Syntax: name of the vector < - c(a1,a2,a3……)

Where a1,a2,a3,……. Are the multiple elements

```
>#  Creating numeric vector using c( ) function
> a <-c( 1,2,3,4,5,6)
>#printing the numeric vector
>print(a)
 [1] 1 2 3 4 5 6
>#  Creating logical vector using c ( ) function
>b<-c(true,true,true,false,true,false)
>#Printing the logical vector
>print(b)
 True,true,true,false,true,false
```

2.names( ) functions:

Syntax: names(v)=c

Where v is the vector to which names are to be assigned.

C is character vector holding the names to be assigned.

```
>#  creating a vector with different quantitites
>fruit<-c(5,10,1,20)
>#naing a vector
>names(fruit)<-c(“orange”,”banana”,”apple”,”peach”)
>#printing the vector
>fruit
Orange   banana   apple   peach
 5              10       1         20
```

3. **Using : operator**

We can use : operator to creates the series of numbers in sequence for a vector.

Syntax: name of vector=start:end

Where

Start is the starting number

end is the ending number

```
Data <- 1:20;
print(Data)
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

4, **Using seq() function**

seq() function is used to create a sequence of elements in a Vector. It takes the length and difference between values as optional argument.

Syntax: *seq(from, to, by, length.out)*

Parameters:

from: Starting element of the sequence

to: Ending element of the sequence

by: Difference between the elements
length.out: Maximum length of the vector

```
# R Program to illustrate
# the use of seq() Function
# Creating vector using seq()
vec1 <- seq(1, 10, by = 2)
vec2 <- seq(1, 10, length.out = 7)
# Printing vectors
print(vec1)
print(vec2)
```

[1] 1 3 5 7 9
[1] 1.0 2.5 4.0 5.5 7.0 8.5 10.0

Accessing Vector Elements

Elements of a Vector are accessed using indexing. The **[ ] brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result.**TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

```
>x<-c("a","b","c","d","e","f")
>#Accessing single element from a vector
>x[1]
[1] "a"
>#Extract multiple elements of a vector
>x[1:5]
[1]  "a","b","c","d","e"

>#  Accessing vector elements using position.
>days <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
>u <- days[c(2,3,6)]
> print(u)
```

Vector Manipulation

Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

Live Demo

```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)

# Vector addition.
add.result <- v1+v2
print(add.result)

# Vector subtraction.
sub.result <- v1-v2
print(sub.result)

# Vector multiplication.
multi.result <- v1*v2
print(multi.result)
```

# Vector division.
divi.result <- v1/v2
print(divi.result)

Functions for Vectors:

1.sort( ) function: In the real world.Elements in a vector can be sorted using the sort( ) function

Syntax: sort(v,decreasing=true/false)
Where
V is the vector which needs to get sorted
decreasing =true denotes that the sorting needs to be done in descending order
decreasing=false denotes that the sorting needs to be done in ascending order
>num<-c(3,85,41,52,0,11,-9,304)
>#sort the elements of the vector.Default is ascending order
> sortas<-sort(num)
>print(sortas)
2.rank( ) function: This function is used to rank the different values in a a vector according to their order in the vector.
>#Assign ranks according to their occurrence
>num1<-c(3,85,41,52,0,11,-9,304)
>rank(num1)
3.max( ) -Returns the max value of a vector
4.min( )-Returns the min value of a vector
5.abs( )-Returns absolute value of a vector
6.sqrt( )- Returns square root of a vector
>sqrt(c(4,9,16,25,36))
[1] 2 3 4 5 6
7.sum()-Adding the numbers in a vector
>sum(c(12,3,15,7))
8.prod( )-Multiplication of numbers in a vector
>prod(c(2,3,5,7))
[1] 210

List

Lists are very important data types in R.A List is a special type of vector that contains elements of different like numbers,strings,vectors,or even another list inside it.

Creating List

A list is a special type of object that can contain data of multiple types.It can be created by using the list( ) function with different data objects.
Syntax: list(v1,v2…….)
Where v1,v2,….are the vectors combined together to create list.
>#creating four different lists
>north<-list("Delhi","Uttarkhand")
>south<-list("Andhra Pradesh","Telangana","Kerala")
>east<-list("West Bengal")
>west<-list("Rajastan");
>#Creating a List of a combination of other lists

>area <-list(north,south,east,west)
>#Determing the number of top level components
>length(area)
[1] 4

Input of data in the form a line: The scan( ) function is used to take input from data and store data in the form a a list.

># Creating a list with multiple items of different data types
>list1<-scan(what=list(student=" ",marks1=0,marks2=0,city=" "))
1:Arun 28 36 Andhra Pradesh
>#displaying the list
>list1

Accessing List Elements:

R provides two ways through which we can access the elements of a list. First one is the indexing method performed in the same way as a vector. In the second one, we can access the elements of a list with the help of names. It will be possible only with the named list.

Example 1: Accessing elements using index

```
# Creating a list containing a vector, and List
list_data <- list(c("Shubham","Arpita","Nishka"),list("MAI","MTC","MCS"))
# Accessing the first element of the list.
print(list_data[1])
# Accessing the third element. The third element is also a list, so all its elements will be printed.
print(list_data[3])
```

Example 2: Accessing elements using names

```
1. # Creating a list containing a vector, a matrix and a list.
2. list_data <- list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80), nrow = 2),list("MAI"
   ,"MTC","MCS"))
3. # Giving names to the elements in the list.
4. names(list_data) <- c("Student", "Marks", "Course")
5. # Accessing the first element of the list.
6. print(list_data["Student"])
7. print(list_data$Marks)
8. print(list_data)
```

Functions for Lists:

Various operations can be applied on the elements of a list like data manipulation and concatenation of different lists.List can also be converted to a matrix or to any other data object in R.

1.Manipulating List Elements:

Addition,deletion and changes to elements can be done in a list but only at end of the list.Examples of data manipulation that can take place on a list are shown below

>#Add element at the end of the list
>listsample[4]<-"welcome to the latest element"
>print(listsample)
>#Remove the last element
>listsample[4]<-NULL
>print(listsample)
>#Update the third element
>listsample[3]<-"Updated at last"
>print(listsample)

2.Concatenating List:

When the concatenation [c( )] function is used for list arguments,the result is a list whose components are those of the argument lists joined together in an order.

```
>#create two lists
>list1<-list(1,2,3)
>list2<-list("CUAP","AP")
>#Merge the two list
>mergelist<-c(list1,list2)
>#print the merged list
>print(mergedlist)
```

Mathematical Operations on List

Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function.

```
# Create lists.
list1 <- list(1:5)
print(list1)
list2 <-list(10:14)
print(list2)
# Convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
print(v1)
print(v2)
# Now add the vectors
result <- v1+v2
print(result)
```

dim( ) function: Creating a matrix from a list using the dim( ) function.We use dim( ) function to assign dimensions to the list.

```
>#creating a heterogeneous list
>x<-list(1,2,3,4,"w","a","c","d")
>#Dimention length is same as list size
>dim(x)<-c(4,2)
>print(x)
```

3.Data frames

Data frames are generic data objects of R which are used to store the tabular data. Data frames are the foremost popular data objects in R programming because we are comfortable in seeing the data within the tabular form. They are two-dimensional, heterogeneous data structures. These are lists of vectors of equal lengths**.**

Data frames have the following constraints placed upon them:

- ✓ A data-frame must have column names and every row should have a unique name.
- ✓ Each column must have the identical number of items.
- ✓ Each item in a single column must be of the same data type.
- ✓ Different columns may have different data types.

To create a data frame we use the data.frame() function.

Assume, we have been asked to store data of our employees (such as employee ID, name and the project that they are working on). We have been given three independent vectors, viz., namely, “EmpNo”, “EmpName” and “ProjName” that holds details such as employee ids, employee names and project names, respectively.

>EmpNo <- c(1000, 1001, 1002, 1003, 1004)

>EmpName <- c(“ramudu”, “ranga”, “balaswamy”, “narayana”, “pushpa”)

>ProjName <- c(“PO1”, “PO2”, “PO3”, “PO4”, “PO5”)

However, we need a data structure similar to a database table or an Excel spreadsheet that can bind all these details together. We create a data frame by the name, “Employee” to store all the three vectors together.

>Employee <- data.frame(EmpNo, EmpName, ProjName)

Let us print the content of the date frame, “Employee”.

> Employee

	EmpNo	EmpName	ProjName
1	1000	ramudu	PO1
2	1001	ranga	PO2
3	1002	balaswamy	PO3
4	1003	narayana	PO4
5	1004	pushpa	PO5

Data Frame Access:

There are two ways to access the content of data frames:

i. By providing the index number in square brackets

ii. By providing the column name as a string in double brackets.

i) By Providing the Index Number in Square Brackets

Example 1

To access the second column, “EmpName”, we type the following command at the R prompt.

> Employee[2]

Example 2

To access the first and the second column, “EmpNo” and “EmpName”, we type the following command at the R prompt.

> Employee[1:2]

Example 3

> Employee [3,]

Example 4

Let us define row names for the rows in the data frame.

> row.names(Employee) <- c("Employee 1", "Employee 2", "Employee 3","Employee 4", "Employee 5")

> row.names (Employee)

>Employee

Let us retrieve a row by its name.

> Employee ["Employee 1",]

Let us pack the row names in an index vector in order to retrieve multiple rows.

> Employee [c ("Employee 3", "Employee 5"),]

ii) By Providing the Column Name as a String in Double Brackets

> Employee [["EmpName"]]

[1] "ramudu" "ranga" "balaswamy" "narayana" "pushpa"

To retrieve a data frame slice with the two columns, “EmpNo” and “ProjName”, we pack the column names in an index vector inside the single square bracket operator.

Let us add a new column to the data frame.

To add a new column, “EmpExpYears” to store the total number of years of experience that the employee has in the organisation, follow the steps given as follows:

Employee$EmpExpYears <-c(5, 9, 6, 12, 7)

Print the contents of the date frame, “Employee” to verify the addition of the new column.

> Employee

Ordering the Data Frames:

Let us display the content of the data frame, “Employee” in ascending order of “EmpExpYears”.

> Employee[order(Employee$EmpExpYears),]

Use the syntax as shown next to display the content of the data frame, “Employee” in descending order of “EmpExpYears”.

> Employee[order(-Employee$EmpExpYears),]

R Functions for understanding Data in Data Frames:

We will explore the data held in the data frame with the help of the following R Functions:

- dim()
- nrow()
- ncol()
- str()
- summary()
- names()
- head()
- tail()
- edit()

1) dim() Function:

The dim() function is used to obtain the dimensions of a data frame. The output of this function returns the number of rows and columns.

> dim (Employee)

a) nrow() Function

The nrow() function returns the number of rows in a data frame.

> nrow(Employee)

b) ncol() Function

The ncol() function returns the number of columns in a data frame.

> ncol(Employee)

2) str() Function

The str() function compactly displays the internal structure of R objects. We will use it to display the internal structure of the dataset, “Employee”.

> str (Employee)

3) summary() Function

We will use the summary() function to return result summaries for each column of the dataset.

> summary (Employee)

4) names() Function

The names()function returns the names of the objects. We will use the names() function to return the column headers for the dataset, “Employee”.

> names (Employee)

5) head() Function

The head()function is used to obtain the first n observations where n is set as 6 by default.

Examples

1. The value of n is set as 3 and hence, the resulting output would contain the first 3 observations of the dataset.

> head (Employee, n=3)

6) tail() Function

The tail()function is used to obtain the last n observations where n is set as 6 by default.

> tail(Employee, n=3)

7) edit()

Edit function will invoke the the text editor on the R Object.

>edit(employee)

R-Matrices:

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types.

We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

Syntax

The basic syntax for creating a matrix in R is −matrix(data, nrow, ncol, byrow, dimnames)

Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Example

Create a matrix taking a vector of numbers as input.

```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)
# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element.

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
# Create the matrix.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
# Access the element at 3rd column and 1st row.
print(P[1,3])
```

```
# Access the element at 2nd column and 4th row.
print(P[4,2])
# Access only the  2nd row.
print(P[2,])
# Access only the 3rd column.
print(P[,3])
```

Modification of the matrix

R allows us to do modification in the matrix. There are several methods to do modification in the matrix, which are as follows:

Assign a single element

In matrix modification, the first method is to assign a single element to the matrix at a particular position.

matrix[n, m]<-y

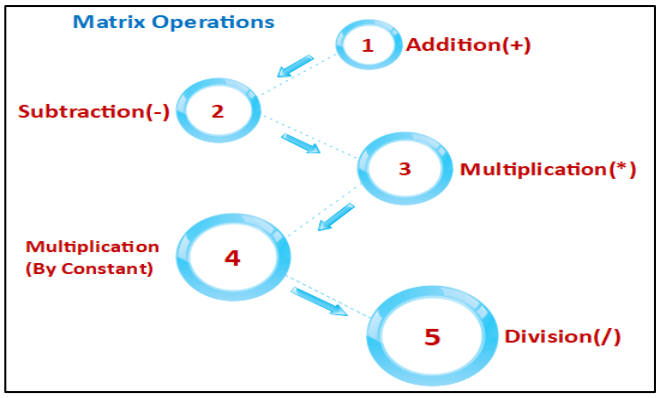
Here, n and m are the rows and columns of the element, respectively. And, y is the value which we assign to modify our matrix.

Example:

```
# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3")
R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print(R)
#Assigning value 20 to the element at 3d roe and 2nd column
R[3,2]<-20
print(R)
```

Matrix operations

In R, we can perform the mathematical operations on a matrix such as addition, subtraction, multiplication, etc. For performing the mathematical operation on the matrix, it is required that both the matrix should have the same dimensions.



Let see an example to understand how mathematical operations are performed on the matrix.

Example 1

```
1. R <- matrix(c(5:16), nrow = 4,ncol=3)
2. S <- matrix(c(1:12), nrow = 4,ncol=3)
3. #Addition
4. sum<-R+S
5. print(sum)
6. #Subtraction
7. sub<-R-S
8. print(sub)
9. #Multiplication
10. mul<-R*S
```

```
print(mul)
#Multiplication by constant
mul1<-R*12
print(mul1)
#Division
div<-R/S
print(div)
```

Transpose and Crossprod: In R Programming we can also calculate transpose of a matrix directly using the transpose t( ) function.The crossprod( ) function in R is used to calculate manipulation of a matrix with its transpose.

```
>M=matrix(c(2,6,5,1,10,4),nrow=2,ncol=3,byrow=TRUE)
>M
>#t(x) represents transpose of x matrix
>t(M)
>#using crossprod( ) for multiplying matrix with its transpose
>crossprod(M)
```

Arrays:

Arrays are the R data objects which can store data in more than two dimensions unlike a matrix.For example,if we create an array of dimension(2,3,4) then it creates 4 rectangular matrices each with 2 rows and 3 columns.

Creating an Array:

An array is created using the array( ) function.It takes vectors as input.

Syntax:array(vectors,dim=,dimname=)

Where

dim has a dimension vector of non-negitive integers

dimname has a vector of rowname,colname,matrixname.

```
>#create an array of character elements from a single vector
>newarray<-array(c('India','Russia'),dim=(3,3,2))
>print(newarray)
>#create a numeric array of two vectors of different lengths
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

Naming Columns and Rows

We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
  matrix.names))
print(result)
```

1.Accessing Elements of an Array

1.Accessing Rows and Columns of an Array:An array can be considered as multiple subscripted collection of data entries.The elements of an array is represented as x[i,j,k] where i denotes the row number,j denotes the column number and k denotes the matrix number.

```
>#Printing the specified row of the specified matrix of the array
>print(newarray[1, ,2])
>#Printing the specified column of the specified matrix of the array
>print(newarray[,1,1])
```

2.Accessing Matrices of an Array: An array can also be considered as collection of matrices.Hence a matrix can be directly created from an array using proper subscripts

```
>newarray<-array(1:18,dim=c(3,3,2))
>#Printing the first of an array
>matrix1<-newarray[, ,1]
```

3.Accessing Individual Elements of an Array: Individual elements of an array may be referenced by giving the name of the array followed by the subscripts in square brackets

```
>#Accessing the element of first row and first column of second matrix
>print(newarray[1,1,2])
```

Functions for Arrays:

We can do calculations across the elements in an array using the apply() function.

Syntax

apply(x, margin, fun)

Following is the description of the parameters used −

- x is an array.
- margin is the name of the data set used.
- fun is the function to be applied across the elements of the array.

Example

We use the apply() function below
to calculate the sum of the elements in the rows of an array across all the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
# Take these vectors as input to the array.
new.array <- array(c(vector1,vector2),dim = c(3,3,2))
print(new.array)
# Use apply to calculate the sum of the rows across all the matrices.
result <- apply(new.array, c(1), sum)
print(result)
```

Decision Making in R Programming

Decision making is about deciding the order of execution of statements based on certain conditions. In decision making programmer needs to provide some condition which is evaluated by the program, along with it there also provided some statements which are executed if the condition is true and optionally other statements if the condition is evaluated to be false.

The decision making statement in R are as followed:

- if statement
- if-else statement
- if-else-if ladder
- nested if-else statement
- switch statement

if structure:When the need is to execute a set of statements based on a condition then we need to use the if structure.

1.if statement: A simple if statement consists of a Boolean expression followed of one or more statements.If the Boolean expression evaluates to be true,then the block of code inside if statement will be executed;on the other hand,if the condition is false,then the statements inside if statement body are completely ignored.

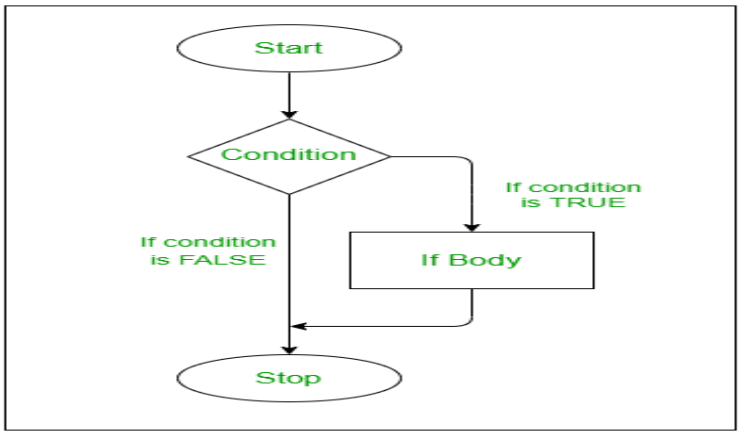
Syntax:

if(boolean expression)

{

Statements will execute if the Boolean expression is true

}



If (x > 10)

{

Cat(x, "is greater than 10")

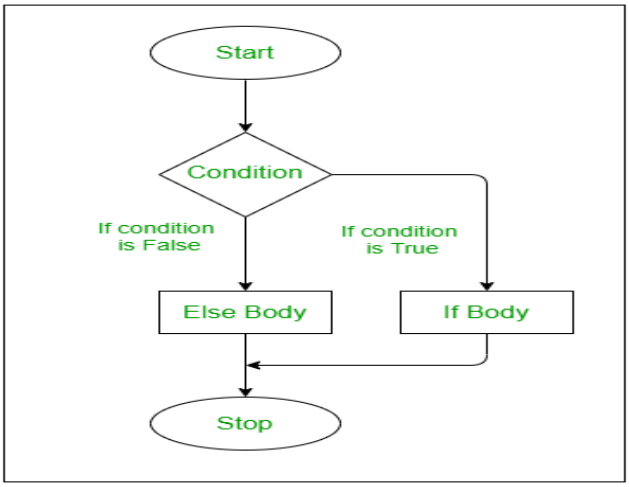
}

Output:

[1] "100 is greater than 10"

if-else: provides us with an optional else block which gets executed if the condition for if block is false. If the condition provided to if block is true then the statement within the if block gets executed, else the statement within the else block gets executed.

Flow chart



Syntax:

```
if(condition is true)
{
   execute this statement
}
else
  {
   execute this statement
}
```

Example:

```
x <- 5
if (x > 10) {
cat (x, "is greater than 10")
} else
{
cat (x, "is less than 10")
}
```

Output:

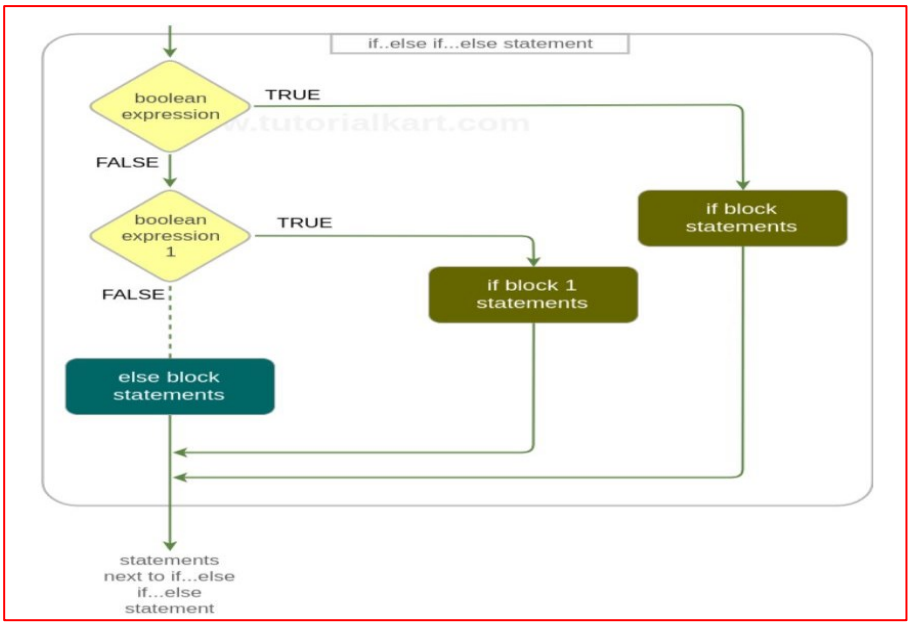
[1] "5 is less than 10"

if-else-if statement in R:

if-else-if ladder in R Programming Language is used to perform decision making. This ladder is used to raise multiple conditions to evaluate the expressions and take an output based on it.

Syntax:

```
if(condition1)
{
 execute this statement
}
else if(condition2)
{
      execute this statement
} .
  .
  .
  .
else {
      execute this statement
}
```



Nested if-else:

The if-else statements can be nested together to form a group of statements and evaluate expressions based on the conditions one by one, beginning from the outer condition to the inner one by one respectively.

```
if(condition1)
{
# execute only if condition 1 satisfies
if(condition 2)
{
# Execute if both condition 1 and 2 satisfy
}
}
else
{
if(condition)
{
  }
else
{
  }
 }
```

switch statement:

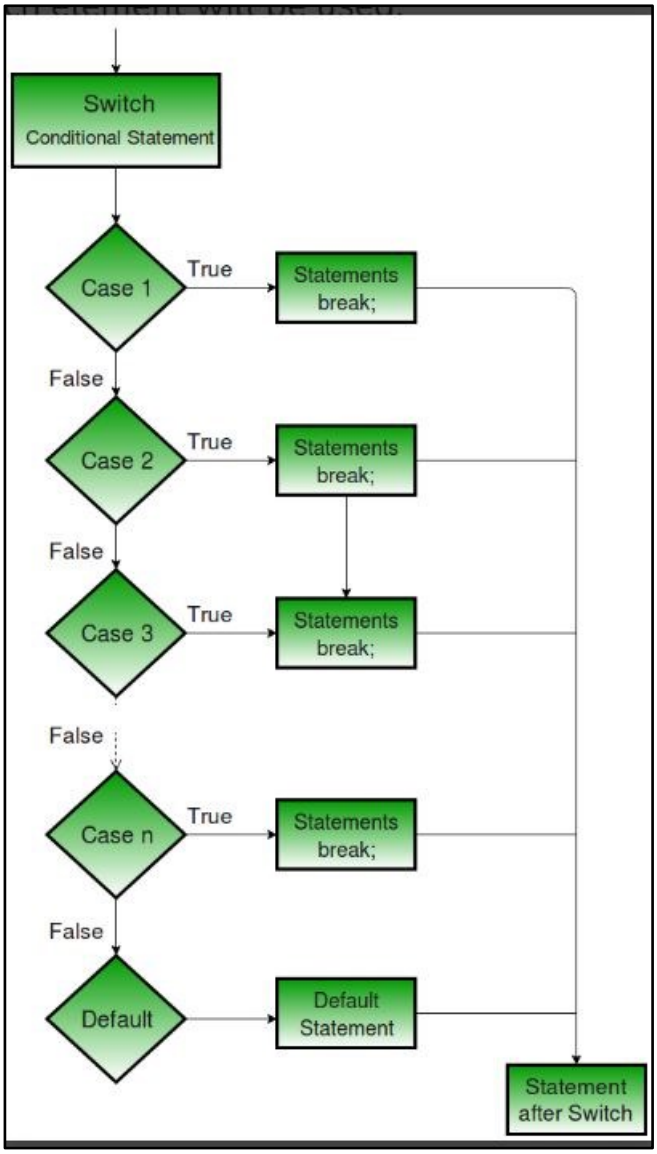
Switch case in R is a multiway branch statement. It allows a variable to be tested for equality against a list of values. Switch statement follows the approach of mapping and searching over a list of values. If there is more than one match for a specific value, then the switch statement will return the first match found of the value matched with the expression.

Syntax:

```
switch(expression, case1, case2, case3....)
```

important Points about Switch Case Statements:

- An expression type with character string always matched to the listed cases.
- An expression which is not a character string then this exp is coerced to integer.
- For multiple matches, the first match element will be used.



Example:

Following is a simple R program
to demonstrate syntax of switch.
val<-
switch(4,"AP","TS","KRNT","GJ","RJ","DL")

print(val)

Example-2:

choice <- "B"
result <- switch(choice, "A" = "Apple", "B" = "Banana", "C" = "Cherry", "Unknown Choice")
print(result)

Looping Statements:

Loops: Looping constructs repetitively execute a statement or series of statements until a condition isn't true. These include the for, while and repeat structures with additional clauses break and next.

1.FOR LOOP:- The for loop executes a statement repetitively until a variable's value is no longer contained in the sequence seq.

- The syntax is for(var in sequence)

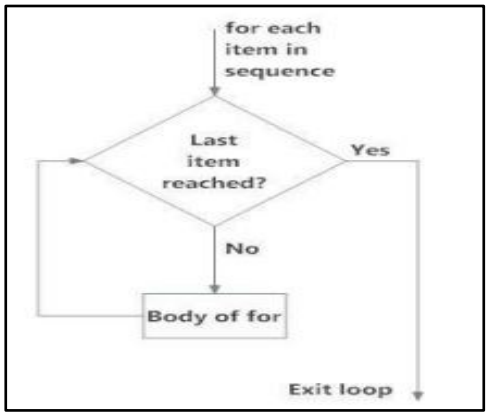
{
statement
}

Here, sequence is a vector and var takes on each of its value during the loop. In each iteration, statement is evaluated.

- for (n in x)

{
- - -
}

It means that there will be one iteration of the loop for each component of the vector x, with taking on the values of those components—in the first iteration, n = x[1]; in the second iteration, n = x[2]; and so on.



- In this example

for (i in 1:10)
print("Hello")
the word Hello is printed 10 times.

- Square of every element in a vector:

> x <- c(5,12,13)
> for(n in x)
print(n^2)
[1] 25 [1] 144 [1] 169

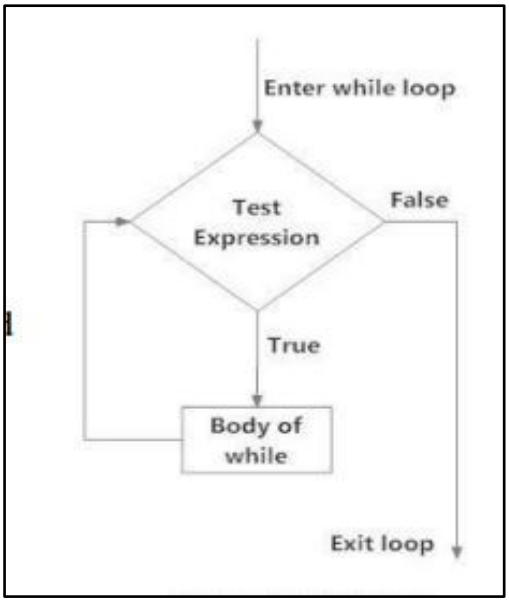
2) WHILE:- A while loop executes a statement repetitively until the condition is no longer true.
Syntax:

```
while (expression)
{
 statement
}
```

- Here, expression is evaluated and the body of the loop is entered if the result is TRUE.
- The statements inside the loop are executed and the flow returns to evaluate the expression again.
- This is repeated each time until expression evaluates to FALSE, in which case, the loop exits

Example

```
> i <- 1
>while(i<=10)
i <- i+4
 > i
[1] 13
```



Program to find the sum of first n natural numbers

```
sum = 0
# take input from the user
num=as.integer(readline(prompt="Enter anumber: "))
Enter a number: 4
 while(num>0)
{
sum=sum+num
 num = num - 1
}
print (paste("The sum is", sum))
```

3) **Break statement:** A break statement is used inside a loop (repeat, for, while) to stop the iterations and flow the control outside of the loop.In a nested looping situation, where there is a loop inside another loop, this statement exits from the innermost loop that is being evaluated.
Syntax:- break
Example

```
x <- 1:5
```

```
for (val in x)
{
 if (val== 3)
{
 break
}
 print(val)
}
```

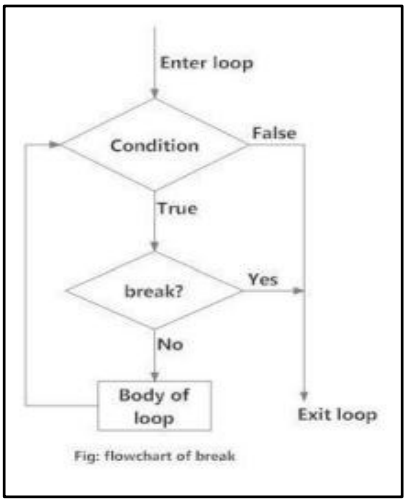


Fig: flowchart of break

4) **Next statement:-** A next statement is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R parser skips further evaluation and starts next iteration of the loop.
Syntax:- next
Example:

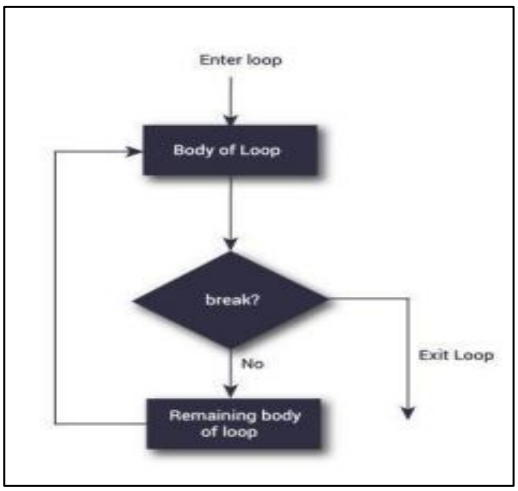
```
x <- 1:5
for (val in x)
{
if (val==3)
{
 next
 }
 print(val)
 }
```

5) Repeat:- Repeat loop is used to iterate over a block of code multiple number of times.There is no condition check in repeat loop to exit the loop.
We must ourselves put a condition explicitly inside the body of the loop and use the break statement to exit the loop. Failing to do so will result into an infinite loop.
Syntax:

```
repeat
{
 statement
```

}

```
x <- 1
repeat
{
        print(x)
        x = x+1
        if (x== 6)
                break
}
```



User Defined Functions:

A function is a set of statements organized together to perform a specific task.Functions are very much useful when a block of statements must be written/executed again and again.Functions are useful when the program size is too large or complex. A function is called to perform each task sequentially from the main program.

Functions are defined using the function( ) directive and are stored are stored as R objects.A function body contains a collection of statements that define what the function does.When a function is called,we may pass a value to the argument.

Syntax:

```
Fname<-function(arg1,arg2,….)
{
Function body
}
```

Where fname is the actual name of the function stored in R environment
arg1,arg2 are the arguments that can have default values

Function without Arguments

A function can be created either without any argument or with single or multiple arguments.

```
>#Creating an empty function
>myfunc1<-function( )
{
--
---
}
>#Displaying the class of the function
>class(myfun1)
>#Executing my first function
>myfunc1( )
NULL
```

Here “myfunc1” is the name of the function which has bo body.Class of “myfunc1”displays the function because it is a user-defined function.Executing the function will return NULL since ther is no body attached to it.

>#Creating a function that has a non-trivial function body

```
>myfunc2<-function( )
{
print("Hello world!")
}
>#Executing a function
>myfunc2( )
>#Creating a function which computes the result
>myfunc3<-function( )
{
a=10
b=20
print(a*b)
print(a+b)
}
>#calling a function
>myfunc3( )
```

Function with Arguments: User defined function with no argument does not allow the user to change the input given to the function.But generally ,the user needs to pass the argument,which changes with the requirement.

Creating a function with single argument

```
>#Display answer when each number in sequence is multiplied by 3
>myfunction<-function(arg1)
{
for(i in 1:arg1)
{
result<-i*3
print(result)
  }
}
>#calling a function with 9 as an argument passed to function
>myfunction (9)
>#calling a function with 5 as an argument passed to function
```

Explanation:

In the first statement ,a user-defined function is created by the name "myfunction".There is only one argument "arg1" which is passed inside the function.The function body starts after the first curly bracket.it has a for loop which has "i" as the counter variable.A sequence is generated from 1 till the number which is passed as argument.

Creating a Function with Multiple Arguments

We can also pass multiple arguments while creating a function.The following example uses four arguments.

```
>newfunction<-function(a,b,c,d)
{
result<-a*b+c-d
print(result)
 }
>#Calling a function with four arguments by position of arguments
>newfunction(7,8,9,2)
```

Nesting of Functions:

We can define one function within other function.

```
>#Creating first function
>function1<-function( )
print("This function calculates the answer for the power of 4")
>#Creating function which includes the above function
>function2<-function(x)
{
function1( )
{
 return(x^4)
}
>#Calling the function
>function2(5)

>#Creating the second version of the function
>function3<-function(x)
{
return(x^4)
function1( )
}
```

Vectorized IF Statement in R

In R Language vectorized operations are a powerful feature that allows you to apply functions or operations over entire vectors at once. The ifelse() function is a primary tool for creating vectorized if statements in R Programming Language.

Introduction to Vectorized ifelse()

The ifelse( ) function in R is used to create vectorized conditional statements. It applies a test condition across each element of a vector and returns one value for TRUE elements and another for FALSE elements. Vectorization in R means that operations are applied element-wise across entire vectors or arrays.

ifelse(test_expression, value_if_true, value_if_false)

- ✓ **test_expression:** A logical vector indicating the condition to test.
- ✓ **value_if_true**: Values returned for elements where the test expression is TRUE.
- ✓ **value_if_false:** Values returned for elements where the test expression is FALSE.

Example of Vectorized ifelse()

```
# Define a numeric vector
numbers <- c(10, 15, 20, 25, 30)
# Apply vectorized if statement
result <- ifelse(numbers > 20, "Above 20", "20 or Below")
# Print the result
print(result)
```

In this example, ifelse() checks each element of the numbers vector:

- If the element is greater than 20, it returns "Above 20".
- Otherwise, it returns "20 or Below".

Example-2:

```
1. # Sample vector
2. x <- c(12, 9, 23, 14, 20, 1, 5)
```

```
# Determine if each number is even or odd
result <- ifelse(x %% 2 == 0, "EVEN", "ODD")
print(result)
# Output: [1] "EVEN" "ODD" "ODD" "EVEN" "EVEN" "ODD" "ODD"
```