Module-01

Introduction of the language, numeric, arithmetic, assignment, and vectors, Matrices and Arrays, Non-numeric Values, Lists and Data Frames, Special Values, Classes, andCoercion, Basic Plotting

---

What is programming Language.

➢ Programming language is defined it is set of instruction.

➢ The instruction are written by the programmer/developer and deliver the instruction to the computer for the particular task.

➢ Because of computer is not understanding the human language it only understand the computer language/binary language(byte code).

Example:C,C++,Java,Python,Javascript,................etc

What is Statistics ?

Statistics is the branch of mathematics.It is defined as study and manipulation of data is called as Statistics.

According to the definition statistics

- Studying
- Analysis
- Interprestation
- presenting
- Organising the data or finalise the data.

Example: The number of people in the town who are watching in the TV

Out of the total population in the town.

What is Statistcal Computing ?

Statistical computing is the defined as it is the bond between statistics and computerscience, is called statistical computing.

Statistical computing is also called as computational statistics.

Example: 1.Regression model

2.Machine learning algorithms

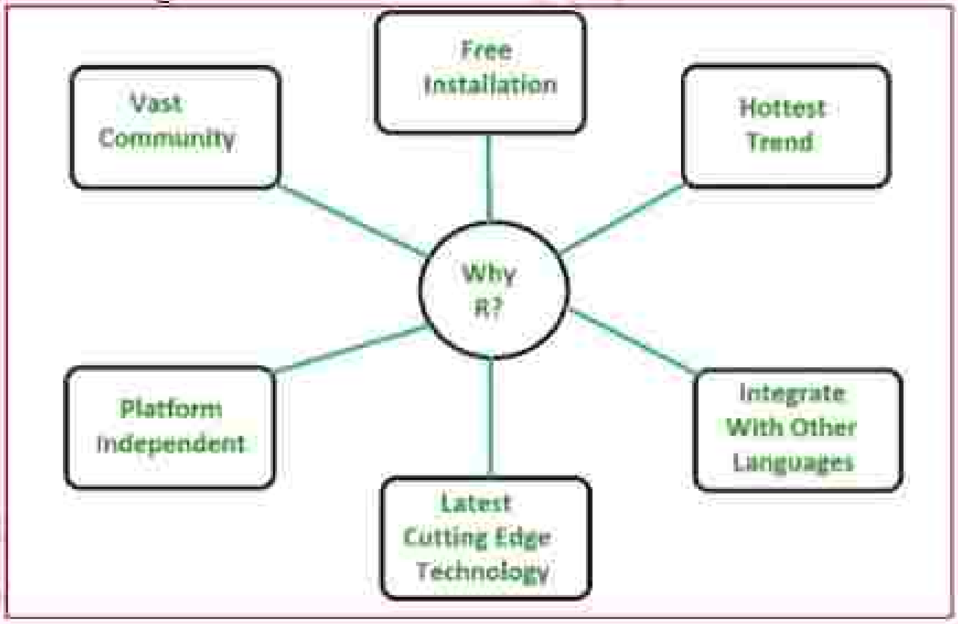
3.Time series Model

What is R Programming ?

- ✓ R programming is the general purpose of the programming language .
- ✓ It is also one of the interpreter programming language and execute line by line code.
- ✓ R programming mainly used in the data Analysis and research fields.
- ✓ R supports procedural programming with functions and, for some functions, object-oriented programming with generic functions.
- ✓ That is widely used as a statistical software and data analysis tool.

Why learn R programming language.

- R is one of the most popular statistical programming languages for data scientists. It is heavily used in the field of machine learning, scientific computing, and statistical analysis.
- Since R is an interpreted programming language, you can run your code without any compiler using interpreter. This makes development easier.
- R can be used to perform vector calculations. It is a vector language and can be used to add functions to a single vector.



Major uses of R Programming Language.

1.Statistical Interfaces

2.Data Analysis

3.Machine learning algorithms.

History of 'R' Programming

The 'R' Programming is introduced in the middle year of 1960's to 1970's S programming language it used with lexical scoping semantics.Become 1990's S programming updated as 'R' programming. It was designed by **Ross Ihaka and Robert Gentleman** at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. Why we called as 'R' Programming because of is invented the two scientist names start with 'R'.So programming language as the R programming language.

Ross Ihaka and Robert Gentleman

- New Zealand statisticians
- Originators of the R programming language
- R is a free software environment for statistical computing and graphics
- Currently, the CRAN (The Comprehensive R Archive Network) package repository features 15363 packages

R programming language was initially developed by Ross Ihaka and Robert Gentleman in the early 1990s while they were at the University of Auckland, New Zealand. R was influenced by the S programming language, which was developed at Bell Laboratories by John Chambers and his colleagues.

Ross Ihaka and Robert Gentleman created R as an open-source programming language specifically designed for statistical computing and graphics. They aimed to provide a free and flexible software tool for statistical analysis and data visualization, making it accessible to researchers, statisticians, and data analysts worldwide.

The name "R" is derived from the initials of the first names of its creators, Ross Ihaka and Robert Gentleman. R gained popularity among statisticians and data scientists due to its powerful features, extensive statistical libraries, and the ability to create high-quality graphics.

Since its inception, R has been continuously developed and improved by a dedicated community of users and developers worldwide. The R Project for Statistical Computing, led by core members and volunteers, maintains and enhances the R language and its ecosystem. The R Foundation, a non-profit organization, provides support for the development and distribution of R, ensuring its open-source nature and promoting its use in various fields.

Ross Ihaka and Robert Gentleman's contributions to the field of statistical computing and the development of R have been significant. They have played a crucial role in creating a versatile and widely adopted programming language that continues to empower researchers and data analysts to explore and analyze data effectively.

Version-Release	**Date**	**Description**
0.49	**1997-04-23**	**First time R's source was released, and CRAN (Comprehensive R Archive Network) was started.**
0.60	**1997-12-05**	**R officially gets the GNU license.**
0.65.1	**1999-10-07**	**update.packages and install.packages both are included.**
1.0	**2000-02-29**	**The first production-ready version was released.**
1.4	**2001-12-19**	**First version for Mac OS is made available.**
2.0	**2004-10-04**	**The first version for Mac OS is made available.**
2.1	**2005-04-18**	**Add support for UTF-8encoding, internationalization, localization etc.**
2.11	**2010-04-22**	**Add support for Windows 64-bit systems.**
2.13	**2011-04-14**	**Added a function that rapidly converts code to byte code.**
2.14	**2011-10-31**	**Added some new packages.**

2.15	2012-03-30	**Improved serialization speed for long vectors.**
3.0	2013-04-03	**Support for larger numeric values on 64-bit systems.**
3.4	2017-04-21	**The just-in-time compilation (JIT) is enabled by default.**
3.5	2018-04-23	**Added new features such as compact internal representation of integer sequences, serialization format etc.**
4.3.1	2023-06-16	**The package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server.**

Features of 'R' Programming Language

- 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.

Application of 'R' Programming

1. Fintech Companies (financial services)
2. Academic Research
3. Government (FDA, National Weather Service)
4. Retail
5. Social Media
6. Data Journalism
7. Manufacturing
8. Healthcare

The companies or organizations that use R.

- Airbnb
- Microsoft
- Uber
- Facebook
- Ford
- Google
- Twitter
- IBM
- American Express
- HP

Advantages and Disadvantages of 'R' Programming

1. Excellent for Statistical Computing and Analysis

R is a statistical language created by statisticians. Thus, it excels in statistical computation. R is the **most used** programming language for developing statistical tools.

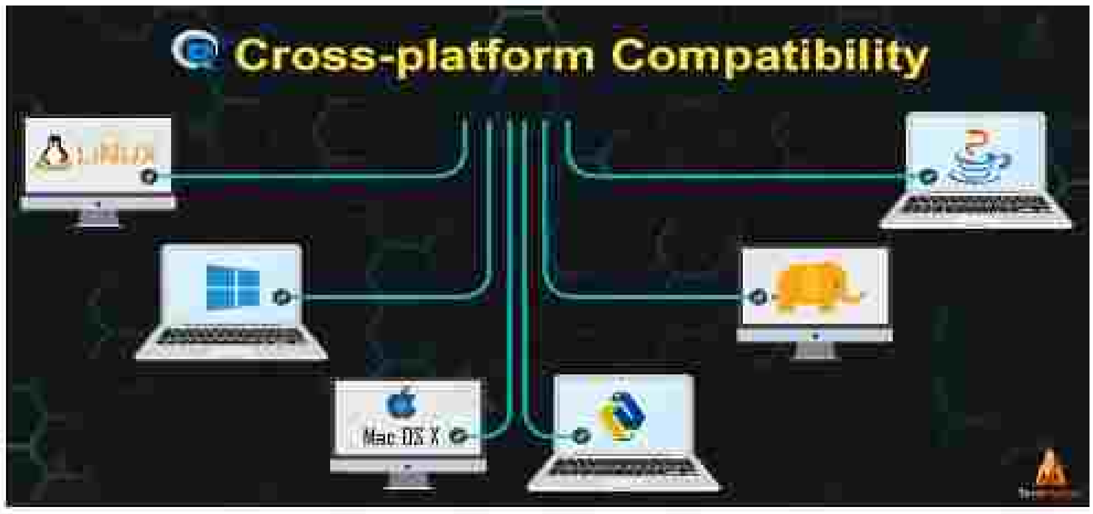
2. Open-source

R is an open-source programming language. Anyone can work with R without any license or fee. Due to this, R has a **huge community** that contributes to its environment.

3. A Large Variety of Libraries

R's **massive community** support has resulted in a very large collection of libraries. R is famous for its graphical libraries. These libraries support and enhance the R development environment. R has libraries with a huge variety of applications.

4. Cross-platform Support



R is **machine-independent**. It supports the cross-platform operation. Thus, it is usable on many different operating systems.

5. Supports various Data Types

R can perform operations on vectors, arrays, matrices, and various other data objects of varying sizes.

6. Can do Data Cleansing, Data Wrangling, and Web Scraping

R can collect data from the internet through web scraping and other means. It can also perform data cleansing. Data cleansing is the process of **detecting** and **removing/correcting** inaccurate or corrupt records. R is also useful for data wrangling which is the process of converting raw data into the desired format for easier consumption.

7. Powerful Graphics

R has extensive libraries that can produce production **quality graphs** and visualizations. These graphics can be of static as well as dynamic nature.

8. Highly Active Community

The R community is very active. There are users from all around the **world** to help and support you. Many latest ideas and technology appear in the R community.

9. Parallel and Distributed Computing

Using libraries like **ddR** or **multiDplyr**, R can process large data sets using parallel or distributed computing.

10. Doesn't need a Compiler

R is an **interpreted language**. This means that it does not need a compiler to turn the code into an executable program. Instead, R interprets the provided code into lower-level calls and pre-compiled code.

11. Compatible with other Programming Languages

R is compatible with other languages like C, C++, and FORTRAN. Other languages like .NET, Java, Python can also directly manipulate objects.

12. Used in Machine learning

R can be useful for machine learning as well. Facebook does a lot of its machine learning research with R. **Sentiment analysis** and **mood prediction** are all done using R. The best use of R when it comes to machine learning is in case of exploration or when building one-off models.

13. Can Interact with Databases

R contains several packages that enable it to interact with databases. Some of these packages are Roracle, Open Database Connectivity Protocol), RmySQL, etc

14. Comprehensive Environment

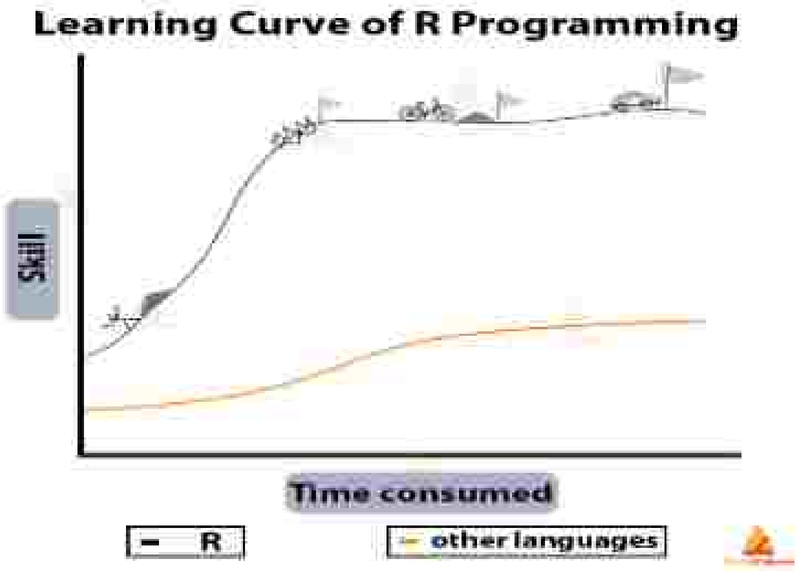
R has a very comprehensive development environment. It helps in statistical computing as well as software development. R is an **object-oriented** programming language. It also has a robust package called **Rshiny** which can produce full-fledged web apps. R can also be useful for developing software packages.

<u>Disadvantages of R Programming</u>

It's no secret that there's also a dark side to R programming. Let's move on to the disadvantages of using R:

1. Steep Learning Curve

As many have said, R makes easy things hard, and hard things easy. **R's syntax is very different** than other languages so are its data types. The learning curve for R is pretty steep for a beginner. Though R is a bit difficult in the beginning, data science enthusiasts still prefer to learn it due to the amazing ***features of R.***



2. Some Packages may be of Poor Quality

CRAN houses more than **10,000 libraries** and packages. Some of them are redundant as well. Due to the large quality, some of the packages may be of poor quality.

3. Poor Memory Management

R commands don't concern with memory management. As a result, R can take up all the available space.

4. Slow Speed

The programs and functions in R are spread across different packages. This makes it slower than alternatives such as MATLAB and Python.

5. Poor Security

R lacks basic security measures. So making web-apps with it is not always safe.

6. No Dedicated Support Team

R has no dedicated support team to help a user with their issues and problems. But the community is quite large, so everybody helps each other out.

7. Flexible Syntax

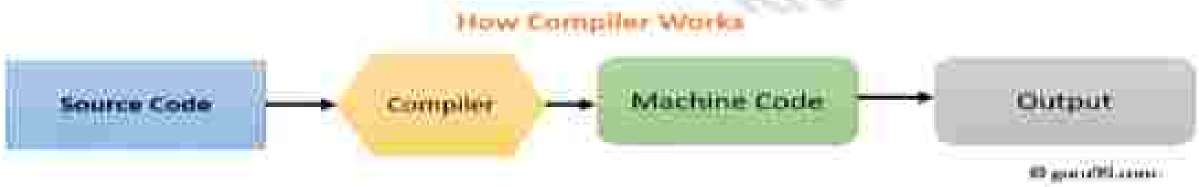
R is a flexible programming language and there are no strict guidelines to follow. You need to maintain proper coding standards to avoid messy and complicated code.

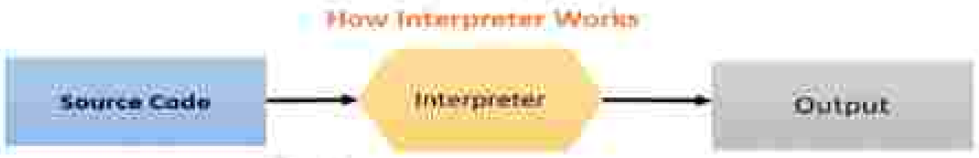
Installation of R programming

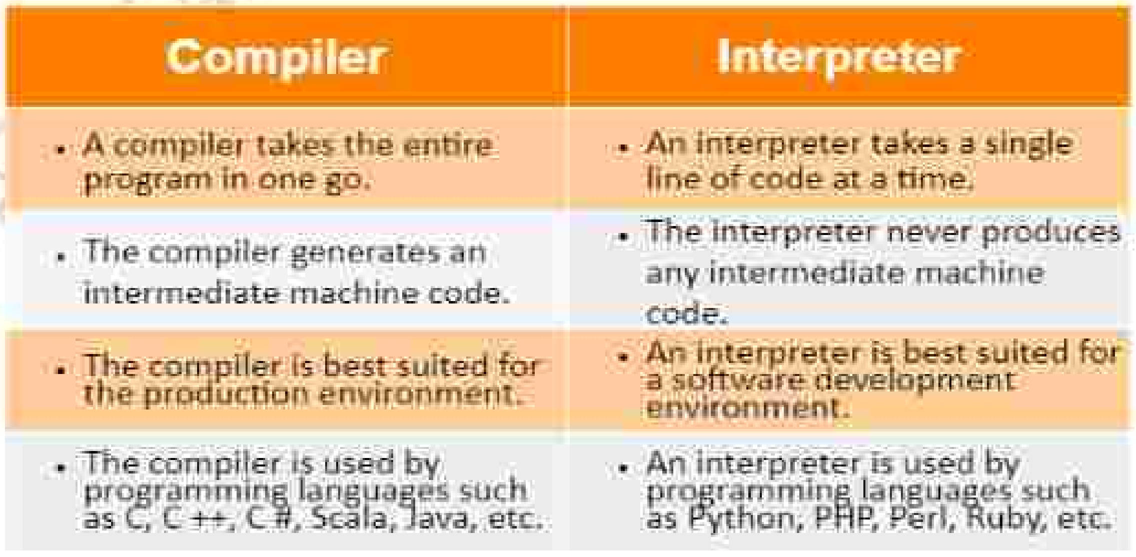
Definition of compiler: A compiler is a special program that translates a programming language's source code into machine code, bytecode or another programming language. The source code is typically written in a high-level, human-readable language such as Java or C,C++,

Definition of interpreter:

An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Examples of interpreted languages are Perl, Python R , and Mat lab.







Compiler	Interpreter
• A compiler takes the entire program in one go.	• An interpreter takes a single line of code at a time.
• The compiler generates an intermediate machine code.	• The interpreter never produces any intermediate machine code.
• The compiler is best suited for the production environment.	• An interpreter is best suited for a software development environment.
• The compiler is used by programming languages such as C, C ++, C #, Scala, Java, etc.	• An interpreter is used by programming languages such as Python, PHP, Perl, Ruby, etc.

Syntax of R Programming

R Programming is a very popular programming language which is broadly used in data analysis. The way in which we define its code is quite simple. The "Hello World!" is the basic program for all the languages.

To develop the programs in two mode

1. The command prompt
2. The Script file

1.The command Prompt('R' Console)

The Command prompt write the code command line. It is directly interact the with theinterpreter.

Example:"JSS COLLEGEMYSORE" Output:[1]
JSS COLLEGE MYSORE

2.The script file(R Studio)

The R script file is another way on which we can write our programs, and then we executethose scripts at our command prompt with the help of R interpreter known as **Rscript**.

We make a text file and write the following code. We will save this file with .R extension as:

filename.R

```
string <-" JSS COLLEGE MYSORE"
print(string)
```

Output:[1] JSS COLLEGE MYSORE

Comments: In R programming, comments are the programmer readable explanation in thesource code of an R program.

The purpose of adding these comments is to make the source code easier to understand.Compilers and interpreters generally ignore these comments.

There are two types commands:

1.Single line command : In R programming there is only single- line comment. R doesn't support multi- line comment. But if we want to perform multi- line comments, then we can add our code in a false block.

Single-line comment

```
#My First program in R programming

string <-"Hello World!"
print(string)
```

The trick for multi-line comment

```
#Trick for multi- line  comment
if(FALSE) {
  'R is an interpreted computer progamming language which was
  created by Ross Ihaka and Robert Gentleman at the University of
  Auckland, New Zealand.,

 #My First program in R progamming
        -"HelloWorld"'
 print(string)
```

Variables in R Programming

Variables are used to store the information to be manipulated and referenced in the R program.

The R variable can store an atomic vector, a group of atomic vectors, or a combination of many R objects.

R supports three ways of variable assignment:

Using equal operator- operators use an arrow or an equal sign to assign values to variables.

Using the leftward operator- data is copied from right to left.

R Variables Syntax

1) **Using equal to operators**

variable name = value

2) Using leftward operator

variable name ←value

Example:

```
var1 ←"hello"

print(var1)

Output: hello
```

<u>The following rules need to be kept in mind while naming a R variable:</u>

1) A valid variable name consists of a combination of alphabets, numbers, dot(.), and underscore(_) characters. Example: var.1 _ is valid.
2) Apart from the dot and underscore operators, no other special character is allowed. Example: var$1 or var#1 both are invalid.
3) Variables can start with alphabets or dot characters. Example: .var or var is valid
4) The variable should not start with numbers or underscore. Example: 2var or var is invalid.
5) If a variable starts with a dot the next thing after the dot cannot be a number. Example: .3var is invalid
6) The variable name should not be a reserved keyword in R. Example: TRUE, FALSE,etc.

class() function:

This built-in function is used to determine the data type of the variable provided to it.

The R variable to be checked is passed to this as an argument and it prints the data type in return.

Syntax:

class(variable)

Example:

```
var1 = "hello"
print(class(var1))
Output: "character"
```

Data Structures in R Programming language:

1) Vectors:

In R, a sequence of elements which share the same data type is known as vector. Vector is classified into two parts:

i) Atomic Vectors

ii) List

Note: There is only one difference between atomic vectors and lists. In an atomic vector, all the elements are of the same type, but in the list, the elements are of different data types.

How to create a vector in R?

1. Using the c() function

In R, we use c() function to create a vector. This function returns a one-dimensional array or simply vector.

Example:

Myvec ←c(1,3,1,4,2)

Print(Myvec)

Output: 1 3 1 4 2

2. Using the colon(:) operator

We can create a vector with the help of the colon operator. There is the following syntax to use colon operator:

Z ←**x:y**

Example: b←c(1:10)

Print (b)

Output: 1 2 3 4 5 6 7 8 9 10

3.Using the seq() function

A sequence function creates a sequence of elements as a vector. The seq() function is used by setting step size with 'by' parameter.

Example: seq_vec<-**seq**(1,4,by=0.5)

Print (seq_vec)

Output: 1.0 1.5 2.0 2.5 3.0 3.5 4.0

Atomic vectors in R

In R, there are four types of atomic vectors.

1) Numeric vector

A vector which contains numeric elements is known as a numeric vector. If we assign a decimal value to any variable, then that variable will become a numeric type.

```
Example: num_vec<-c(10.1, 10.2, 33.2)
          Print(num_vec )
          class(num_vec)
Output: 10.1     10.2   33.2
 "numeric"
```

2) Integer vector

A non-fraction numeric value is known as integer data. An integer value can be assigned to variable by appending L to the value.

```
Example: int_vec1<-c(1L,2L,3L,4L,5L)
          class(int_vec1)
Output:  "integer"
```

3) Character vector

A vector which contains character elements is known as an integer vector. In R character data type value can be created using double quotes("") or single quotes('').

```
Example: char_vec1<-c("shubham","arpita","nishka","vaishali")
Print(char_vec)
class(char_vec1)
Output: "shubham"  "arpita"  "nishka"  "vaishali"
        "character"
```

4) Logical vector

The logical data types have only two values i.e., True or False. These values are based on which condition is satisfied. A vector which contains Boolean values is known as the logical vector.

```
Example: d<- 5
         e<- 6
         f<- 7
    log_vec<-c(d<e, d<f, e<d,e<f,f<d,f<e)
                print(log_vec)
    class(log_vec)
Output: TRUE  TRUE   FALSE   TRUE   FALSE   FALSE
        "logical"
```

Accessing elements of vectors

We can access the elements of a vector with the help of vector indexing. Indexing denotes the position where the value in a vector is stored.

In R, the indexing starts from 1. We can perform indexing by specifying an integer value in square braces [] next to our vector.

Example: seq_vec<-**seq**(1,4,length.out=6)

```
Print(seq_vec)
Print(seq_vec[2])
Output: 1.0 1.6 2.2 2.8 3.4 4.0
   1.6
```

Vector Operation

1) Combining vectors

By combining one or more vectors, it forms a new vector which contains all the elements of each vector.

Example: p<-**c**(1,2,4,5,7,8)

```
q<-c("shubham","arpita","nishka","gunjan","vaishali","sumit")
r<-c(p,q)
print (r)
```

```
Output: "1"      "2"      "4"      "5"      "7"      "8"
"shubham"  "arpita"          "nishka"  "gunjan"  "vaishali" "sumit"
```

2) Arithmetic operations

We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors.

Example:

```
a<-c(1,3,5,7)
b<-c(2,4,6,8)
print (a+b)
print (a-b)
Output: 3    7    11    15
        -1   -1   -1    -1
```

R Lists

In R, lists are the second type of vector. A list is a data structure which has components of mixed data types. Lists are the objects of R which containelements of different types such as number, vectors, string and another list insideit.

Lists creation

The function which is used to create a list in R is list( ).

Example:

1) list_1<-**list**(1,2,3)
   list_2<-**list**("Shubham","Arpita","Vaishali")
   Output:
```
1
2
3
"Shubham"
 "Arpita"
"Vaishali"
```

2) list_data□**list**("Shubham","Arpita",c(1,2,3,4,5),TRUE,FALSE,22.5,12L)
   print(list_data)
   Output:
```
"Shubham"
 "Arpita"
1 2 3 4 5
TRUE
FALSE
 22.5
12
```

Giving a name to list elements

There are only three steps to print the list data corresponding to the name:

1. Creating a list.
2. Assign a name to the list elements with the help of names() function.
3. Print the list data.

Example:

# Creating a list containing a vector, a matrix and a list.

```
list_data <- list(c("Shubham","Nishka","Gunjan"), matrix(c(40,80,60,70,90,80),
 nrow = 2),
list("BCA","MCA","B.tech"))
```

```
# Giving names to the elements in the list.
 names(list_data) <- c("Students", "Marks", "Course")
```

```
# Show the list.
 print(list_data)
```

Output:

```
 $Students
[1] "Shubham" "Nishka"  "Gunjan"

 $Marks
[,1] [,2] [,3]
[1,]    40   60   90
[2,]    80   70   80

 $Course
$Course[[1]]
 [1] "BCA"

$Course[[2]]
 [1] "MCA"

$Course[[3]]
 [1] "B. tech."
```

Accessing List Elements

R provides two ways through which we can access the elements of a list.

1) First one is the indexing method performed in the same way as a vector.
2) In the second one, we can access the elements of a list with the help of names.

Example 1: Accessing elements using index

```
# Creating a list containing a vector, a matrix and a list.
  list_data <- list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80),
  nrow = 2),list("BCA","MCA","B.tech"))
# Accessing the first element of the list.
  print(list_data[1])
```

Output:

```
  "Shubham" "Arpita"  "Nishka"
```

Example 2: Accessing elements using names

```
# Creating a list containing a vector, a matrix and a list.
  list_data <- list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80),
  nrow = 2),list("BCA","MCA","B.tech"))

  # Giving names to the elements in the list.
  names(list_data) <- c("Student", "Marks", "Course")
  # Accessing the first element of the list.
  print(list_data["Student"])
```

Output:

```
  $Student
"Shubham" "Arpita"  "Nishka"
```

<u>Manipulation of list elements</u>

R allows us to add, delete, or update elements in the list.

Example

```
  # Creating a list containing a vector, a matrix and a list.
  list_data <- list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80),
  nrow = 2),list("BCA","MCA","B.tech"))

# Giving names to the elements in the list.
  names(list_data) <- c("Student", "Marks", "Course")

  #Adding element at the end of the list.
  list_data[4] <- "Moradabad"
  print(list_data[4])
```

```
#Removing the last element.
  list_data[4] <- NULL

# Printing the 4th Element.
  print(list_data[4])

# Updating the 3rd Element.
  list_data[3] <- "Masters of computer applications"
  print(list_data[3])
```

Output:

```
"Moradabad"

  $<NA>
  NULL

$Course
  "Masters of computer applications"
```

Converting list to vector

There is a drawback with the list, i.e., we cannot perform all the arithmetic operations on list elements.

This drawback can be overcome with the function unlist( ), this function converts the list into vectors.

Example:

```
# Creating lists.
list1 <- list(1:5)
print(list1)
list2 <-list(10:14)
print(list2)

# Converting the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)

adding the vectors
result <- v1+v2
print(result)
```

Output:
```
1 2 3 4 5
10 11 12 13 14
11 13 15 17 19
```

Merging Lists

R allows us to merge one or more lists into one list.

To merge the lists, we have to pass all the lists into list function as a parameter, and it returns a list which contains all the elements which are present in the lists.

Example

```
# Creating two lists.
Even_list <- list(2,4,6,8)
Odd_list <- list(3,5,7,9)

# Merging the two lists.
merged.list <- list(Even_list,Odd_list)

# Printing the merged list.
print(merged.list)
```

Output:
```
2
4
6
8
3
5
7
9
```

R Matrix

In R, a two-dimensional rectangular data set is known as a matrix.

A matrix is created with the help of the vector input to the matrix function.

In R, we use matrix( ) to create matrix.

Syntax: matrix(data,nrow,ncol,byrow,dim_names)

data:- It is the input vector which is the data elements of the matrix.

nrow:- It is the number of rows which we want to create in the matrix.

ncol: It is the number of columns which we want to create in the matrix.

byrow:- The byrow parameter is a logical clue. If its value is true, then the input vector elements are arranged by row.

dim_name:- It is the name assigned to the rows and columns.

Example: p <- matrix(c(5:16), nrow=4, ncol=3, byrow=TRUE)

Print(p)

```
Output:    5    6    7
        8    9    10
        11   12   13
        14   15   16
```

Assigning names to the matrix:

```
# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
ccol_names = c("col1", "col2", "col3")

R <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_
names))  print(R)
```

Output:

```
         col1 col2 col3
row1  3     4    5
row2  6     7    8
row3  9    10   11
row4   12   13   14
```

Accessing matrix elements in R

There are three ways to access the elements from the matrix.

1. We can access the element which presents on nth row and mth column.
2. We can access all the elements of the matrix which are present on the nth row.

3. We can also access all the elements of the matrix which are present on the mth column.

Example: For the above created R matrix, accessing the elements as follow

```
#Accessing element present on 3rd row and 2nd column
print(R[3,2])

#Accessing element present in 3rd row
print(R[3,])

#Accessing element present in 2nd column
print(R[,2])
```

```
Output: 12

col1 col2 col3
 11   12   13

row1 row2 row3 row4
  6    9   12   15
```

Modification of the matrix

Assign a single element

In matrix modification, the first method is to assign a single element to the matrix at a particular position. By assigning a new value to that position, the old value will get replaced with the new one.

Syntax: 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:

```
R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, co
l_names))

#Assigning value 20 to the element at 3d roe and 2nd column
R[3,2]<-20
print(R)
```

Output:
```
          col1 col2 col3
 row1  5     6    7
 row2  8     9   10
 row3   11    20   13
 row4   14    15   16
```

Use of Relational Operator
```
 R[R==12]<-0
 print(R)
```
output: col1 col2 col3
```
 row1  5     6    7
 row2  8     9   10
 row3   11    0  13
 row4   14    15 16
```

Addition of Rows and Columns

The **cbind()** and **rbind()** function are used to add a column and a row respectively.

Example:
```
 R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, co
 l_names))
 #Adding row
 rbind(R,c(17,18,19))
 print(R)
```
Output:
```
          col1 col2 col3
 row1  5     6    7
 row2  8     9   10
 row3   11   12   13
 row4   14   15   16
        17   18   19
```

```
#Adding column
 cbind(R,c(17,18,19,20))
 print(R)
```
Output:
```
          col1 col2 col3
 row1  5     6    7 17
 row2  8     9   10 18
```

```
row3    11   12   13 19
  row4   14   15   16 20
```

Matrix operations

In R, we can perform the mathematical operations on a matrix such as addition, subtraction, multiplication, etc.

Example:

```
R <- matrix(c(5:16), nrow = 4,ncol=3)
S <- matrix(c(1:12), nrow = 4,ncol=3)

#Addition
sum<-R+S
print(sum)

#Subtraction
sub<-R-S
print(sub)

#Multiplication
mul<-R*S
print(mul)

#Division
div<-R/S
print(div)
```

Output:

```
 [,1] [,2] [,3]
 [1,]    6  14   22
 [2,]    8  16   24
 [3,]   10   18 26
 [4,]   12   20 28

      [,1] [,2] [,3]
 [1,]    4  4     4
 [2,]    4  4     4
 [3,]    4  4     4
 [4,]    4  4     4
```

```
     [,1] [,2] [,3]
[1,]    5   45  117
[2,]  1260     140
[3,]  2177     165
[4,]  3296     192

          [,1]     [,2]     [,3]
[1,] 5.000000 1.800000 1.444444
[2,] 3.000000 1.666667 1.400000
[3,] 2.333333 1.571429 1.363636
[4,] 2.000000 1.500000 1.333333
```

Applications of matrix

1. Matrix is the representation method which helps in plotting common survey things.
2. In robotics and automation, Matrices have the topmost elements for the robot movements.
3. In computer-based application, matrices play a crucial role in the creation of realistic seeming motion.

Arrays

In R, arrays are the data objects which allow us to store data in more than two dimensions. In R, an array is created with the help of the **array()** function.

This array() function takes a vector as an input and to create an array it uses vectors values in the **dim** parameter.

For example- if we will create an array of dimension (2, 3, 4) then it will create 4 rectangular matrices of 2 row and 3 columns.

Syntax:

array_name <- array(data, dim= (row_size, column_size, matrices, dim_names))

data: It is an input vector which is given to the array.

Matrices: In R, the array consists of multi-dimensional mat

row_size: This parameter defines the number of row elements which an array can store.

column_size: This parameter defines the number of columns elements which an array can store.

dim_names: This parameter is used to change the default names of rows and columns.

Creation of an Array:

There are only two steps to create a matrix which are as follows

1. In the first step, we will create two vectors of different lengths.
2. Once our vectors are created, we take these vectors as inputs to the array.

Example:

```
#Creating two vectors of different lengths
vec1 <-c(1,3,5)
vec2 <-c(10,11,12,13,14,15)

#Taking these vectors as input to the array
res <- array(c(vec1,vec2),dim=c(3,3,2))
print(res)
```

Output:

```
, , 1
    [,1] [,2] [,3]
[1,]   1   10   13
[2,]   3   11   14
[3,]   5   12   15

, , 2
    [,1] [,2] [,3]
[1,]   1   10   13
[2,]   3   11   14
[3,]   5   12   15
```

Naming rows and columns

In R, we can give the names to the rows, columns, and matrices of the array. This is done with the help of the dim name parameter of the array() function.

Example:

```
#Creating two vectors of different lengths
vec1 <-c(1,3,5)
vec2 <-c(10,11,12,13,14,15)

#Initializing names for rows, columns and matrices
col_names <- c("Col1","Col2","Col3")
row_names <- c("Row1","Row2","Row3")
matrix_names <- c("Matrix1","Matrix2")

#Taking the vectors as input to the array
res <- array(c(vec1,vec2),dim=c(3,3,2),dimnames=list(row_names,col_names,m
atrix_names))
print(res)
```

Output:

```
, , Matrix1

     Col1 Col2 Col3
Row1  1    10  13
Row2  3    11  14
Row3  5    12  15

, , Matrix2

     Col1 Col2 Col3
Row1  1    10  13
Row2  3    11  14
Row3  5    12  15
```

Accessing array elements

The elements are accessed with the help of the index.

Example: For the above created array

Print(res[3, ,2]) **#To print third row of second matrix**

Output: 5 12 15

Print(res[3,2,2]) **#To print third row second column element of 2nd matrix**

Output: 12

Print(res[,2,1]) **#To print second column element of 1nd matrix**

Output: 10 11 12

Manipulation of elements

The array is made up matrices in multiple dimensions so that the operations on elements of an array are carried out by accessing elements of the matrices.

Example:

```
#Creating two vectors of different lengths
vec1 <-c(1,3,5)
vec2 <-c(10,11,12,13,14,15)

#Taking the vectors as input to the array1
res1 <- array(c(vec1,vec2),dim=c(3,3,1))
print(res1)

#Creating two vectors of different lengths
vec1 <-c(8,4,7)
vec2 <-c(16,73,48,46,36,73)

#Taking the vectors as input to the array2
res2 <- array(c(vec1,vec2),dim=c(3,3,1))
print(res2)

#Creating matrices from these arrays
res3 <- mat1+mat2
print(res3)
```

Output:

```
, , 1

     [,1] [,2] [,3]
[1,]    1   10   13
[2,]    3   11   14
[3,]    5   12   15
```

```
, , 1
     [,1] [,2] [,3]
[1,]    8   16   46
[2,]    4   73   36
[3,]    7   48   73

     [,1] [,2] [,3]
[1,]    9   26   59
[2,]    7   84   50
[3,]   12   60   88
```

Data Frame

A data frame is a two-dimensional array-like structure or a table in which a column contains values of one variable, and rows contains one set of values from each column.

A data frame is a special case of the list in which each component has equal length.

A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.

There are following characteristics of a data frame.

- The columns name should be non-empty.
- The rows name should be unique.
- The data which is stored in a data frame can be a factor, numeric, or character type.
- Each column contains the same number of data items.

How to create Data Frame

In R, the data frames are created with the help of frame() function of data. This function contains the vectors of any type such as numeric, character, or integer.

Example: we create a data frame that contains employee id (integer vector), employee name(character vector), salary(numeric vector), and starting date(Date)

```
# Creating the data frame.
emp.data<- data.frame(
employee_id = c (1:5),
employee_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
sal = c(623.3,915.2,611.0,729.0,843.25),
starting_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-
05-11",
"2015-03-27")),
stringsAsFactors = FALSE
)
```

```
# Printing the data frame.
print(emp.data)
```

Output:

```
employee_id employee_name    sal      starting_date
1     1           Shubham  623.30    2012-01-01
2     2            Arpita  915.20    2013-09-23
3     3            Nishka  611.00    2014-11-15
4     4            Gunjan  729.00    2014-05-11
5     5             Sumit  843.25    2015-03-27
```

To print the structure of data frame

```
# Printing the structure of data frame.
str(emp.data)
```

Output:

```
'data.frame'       :   5 obs. of  4 variables:
 $ employee_id  : int  1 2 3 4 5
 $ employee_name: chr  "Shubham" "Arpita" "Nishka" "Gunjan" ...
 $ sal          : num  623 515 611 729 843
 $ starting_date: Date, format: "2012-01-01" "2013-09-23" ...
```

Extracting data from Data Frame

We can extract the data in three ways which are as follows:

1. We can extract the specific columns from a data frame using the column name.
2. We can extract the specific rows also from a data frame.
3. We can extract the specific rows corresponding to specific columns.

Extracting the specific columns from a data frame

Example: For the above created data frame

```
# Extracting specific columns from a data frame
final <- data.frame(emp.data$employee_id,emp.data$sal)
print(final)
```

Output:

```
emp.data.employee_id     emp.data.sal
1           623.30
2           515.20
3                            611.00
4           729.00
5           843.25
```

Extracting the specific rows from a data frame

Example

```
# Extracting first row from a data frame
final <- emp.data[1,]
print(final)
```

Output:

```
   employee_id  employee_name   sal     starting_date
      1           Shubham         623.3   2012-01-01
```

Extracting specific rows corresponding to specific columns

Example:

```
# Extracting 2nd and 3rd row corresponding to the 1st and 4th column
final <- emp.data[c(2,3),c(1,4)]
print(final)
```

Output:

```
employee_id   starting_date
2             2013-09-23
3             2014-11-15
```

Modification in Data Frame

It is possible to add and delete rows and columns to the data frame.

Example: Adding rows and columns

```
#Adding row in the data frame
  x <- list(6,"Vaishali",547,"2015-09-01")
  rbind(emp.data,x)
```

```
Output: employee_id  employee_name        sal        starting_date
        1            Shubham     623.30        2012-01-01
        2            Arpita      515.20        2013-09-23
        3            Nishka      611.00        2014-11-15
        4            Gunjan      729.00        2014-05-11
        5            Sumit       843.25        2015-03-27
        6            Vaishali    547.00        2015-09-01
```

#Adding column in the data frame

```
y <- c("Moradabad","Lucknow","Etah","Sambhal","Khurja")
cbind(emp.data,Address=y)
```

Output:

```
employee_id  employee_name   sal       starting_date    Address            1
Shubham      623.30       2012-01-01      Moradabad
2            Arpita       515.20       2013-09-23      Lucknow
3            Nishka       611.00       2014-11-15      Etah
4            Gunjan       729.00       2014-05-11      Sambhal
5            Sumit        843.25       2015-03-27      Khurja
```

Non-Numeric Values

Logical-values

Introduction to Logical-values

• Logical-values can only take on two values: TRUE or FALSE.

• Logical-values represent binary states like

- >yes/no

->one/zero

• Logical-values are used to indicate whether a condition has been met or not.

TRUE and FALSE Notation

• Logical-values are represented as TRUE and FALSE.

• TRUE and FALSE are abbreviated as T and F, respectively.

Assigning Logical-values

• Example:

```
b1 <- TRUE
b2 <- FALSE
```

Creating Vectors

- Vectors can be filled with logical-values using T or F.
- Example:

```
myvec <- c(T,T,F,F,F)
```

Vector Length

- You can determine the length of a vector using the `length` function.
- Example:

`length(myvec)` returns 5

A Logical Outcome: Relational Operators

Using Relational Operators

- Relational operators are used to find the relationship between two operands.
- The output of relational expression is either TRUE or FALSE.
- The 2 operands may be constants, variables or expressions.
- There are 6 relational operators:

Operator	Meaning of Operator	Example
>	Greater than	4 > 5 returns FALSE
<	Less than	4 < 5 returns TRUE
>=	Greater than or equal to	4 >= 5 returns FALSE
<=	Less than or equal to	4 <= 5 returns TRUE
==	Equal to	4 == 5 returns FALSE
!=	Not equal to	4 != 5 returns TRUE

any & all Functions

- any() checks whether at least one element in a vector meets a specific condition.
- It returns TRUE if any element satisfies the condition; otherwise, it returns FALSE.

Example:

```
 # Creating a vector
 vector1 <- c(1, 2, 3, 4, 5)

# Checking if any element is greater than 3
 result <- any(vector1 > 3)
```

Output:

```
TRUE
```

• all() checks whether all elements in a vector meet a specific condition.

• It returns TRUE if all elements satisfy the condition; otherwise, it returns FALSE

Example:

```
# Creating a vector
vector2 <- c(1, 2, 3, 4, 5)
# Checking if all elements are greater than 0
result <- all(vector2 > 0)
```

Output:

TRUE

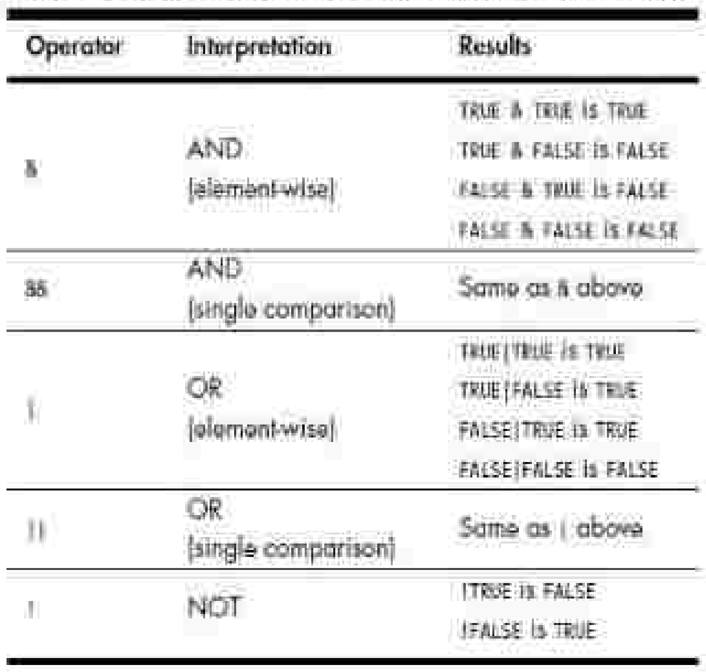
Multiple Comparisons: Logical Operators

Using Logical Operators for Multiple Conditions

• These operators are used to perform logical operations like negation, conjunction
and disjunction.

• The output of logical expression is either TRUE or FALSE.

Table 4-2: Logical Operators Comparing Two Logical Values



Operator	Interpretation	Results
&	AND (element-wise)	TRUE & TRUE is TRUE TRUE & FALSE is FALSE FALSE & TRUE is FALSE FALSE & FALSE is FALSE
&&	AND (single comparison)	Same as & above
|	OR (element-wise)	TRUE|TRUE is TRUE TRUE|FALSE is TRUE FALSE|TRUE is TRUE FALSE|FALSE is FALSE
||	OR (single comparison)	Same as | above
!	NOT	!TRUE is FALSE !FALSE is TRUE

Short and Long Versions

• There are versions of logical operators: i) Short versions ii) Long versions.

i) Short versions are for element-wise comparisons.
Short versions return multiple logical-values.
Eg: `&`, `|`

• Element-wise comparisons are performed when comparing two vectors of equal length.

• Element-wise comparisons return a vector of logical-values.

• Example:

`b1 <- c(T, F, F)`
`b2 <- c(F, T, F)`
`b1 & b2` returns `[F, F, F]`.
`b1 | b2` returns `[T, T, F]`.

ii) Long versions are for comparing individual values.
Long versions return a single logical-value.
Eg: `&&`, `||`

• Using long versions of logical operators evaluates only the first pair of logicals two vectors.

• Example:
`b1 <- c(T, F, F)`
`b2 <- c(F, T, F)`
`b1 && b2` returns `F`.
`b1 || b2` returns `T`.

Logical Subsetting and Extraction

• Logical subsetting and extraction involve using logical conditions to select elements
that satisfy a particular criterion.

• You create a logical vector with TRUE and FALSE values based on the condition.

•Then, you use this vector to subset or extract elements.

Example:

```
# Creating a numeric vector
numeric_vector <- c(1, 2, 3, 4, 5)

# Logical subsetting to extract even numbers
even_numbers <- numeric_vector[numeric_vector %% 2 == 0]
```

Output:

```
2 4
```

Explanation of above program:

• We have a numeric vector called numeric_vector.

• We use the logical condition numeric_vector %% 2 == 0 to create a logical vector
with TRUE for even numbers and FALSE for odd numbers.

• We use this logical vector for subsetting, resulting in even_numbers containing only
the even elements from numeric_vector.

String

• A string is a data type.

• It is used to represent text or character data.

• Strings can consist of almost any combination of characters, including numbers.

• Strings are commonly used for storing and manipulating textual information.
For e.g.: names, sentences, and text-data extracted from files or databases.

Creating a String

• You can create strings using single or double quotation marks.

• Example:

```
single_quoted <- 'This is a single-quoted string.'
double_quoted <- "This is a double-quoted string."
```

Finding String Length with `nchar()`

• nchar() can be used to determine the number of characters in a given string.

• It calculates and returns the length of a string in terms of the number of characters

Example:

```
  # Define a string
  my_string <- "Hello, World!"

# Use nchar() to determine the length of the string
  string_length <- nchar(my_string)

# Print the result
  cat("The length of the string is:", string_length)
```

Output:

```
  The length of the string is: 13
```

Explanation of above program:

• We define a string my_string containing the text "Hello, World!".

• We use the nchar() function to calculate the length of the string, including spaces and special characters.

• The result is stored in the string_length variable.

• We then use cat() to display the length of the string.

Concatenation

Concatenation Functions

• Two main functions are used for concatenating strings: `cat` and `paste`.

1) Using the **cat()** Function

• cat() can be used for concatenating and printing strings with optional separators.

• Example:

```
# Concatenate and print two strings with a space separator
  cat("Hello", "World")
```

Output: "Hello World"

2) Using the **paste()** Function

• paste() can be used to concatenate multiple strings into one, with optional separator and other arguments.

Example:

```
  # Concatenate two strings with a space separator
  concatenated <- paste("Hello", "World")
```

Output:

```
  "Hello World"
```

cat() vs. paste()

- cat() is used for printing (displaying) text to the console.
It concatenates and prints its arguments with optional separators.
- paste() is used for concatenating strings into a single character vector.
The concatenated string can be assigned to a variable for further use.
It doesn't print directly to the console.

Separator (sep) Argument

- An optional argument `sep` is used as a separator between concatenated strings.
- Example:

```
# Concatenate two strings with a custom separator
concatenated <- paste("Hello", "World", sep = ", ")
#Output:
"Hello, World"
```

Escape Sequences

Backslash (\) Usage

- The backslash (\) is used to invoke an escape sequence.
- Escape sequences allow you to enter characters that control the format and spacing of the string.

Common Escape Sequences

`\n` starts a newline.

`\t` represents a horizontal tab.

`\b` invokes a backspace.

`\\` is used to include a single backslash.

`\"` includes a double quote.

Substrings and Matching

Pattern Matching

- Pattern matching allows you to inspect a given string to identify smaller strings within it.
- substr() can be used to extract substrings within a given string

Example:

```
# Extracting a substring from a string
original_string <- "Hello, World!"
substring <- substr(original_string, start = 1, stop = 5)
Output:
"Hello"
```

sub() can be used for replacing the first occurrence of a pattern within a string

Example:

# Replacing the first occurrence of "apple" with "banana"
text <- "I like apples, but apples are red."
new_text <- sub("apple", "banana", text)
Output:
I like bananas, but apples are red."

gsub() can be used for replacing all occurrences of a pattern within a string.
Example:
# Replacing all occurrences of "apple" with "banana"
text <- "I like apples, but apples are red."
new_text <- gsub("apple", "banana", text)
Output:
I like bananas, but bananas are red."

Special Values

When a data set has missing observations or when a practically infinite number is calculated the software has some unique terms reserved for these situations. They are:

1) **NA (Not Available):-** If the value is not define, data value is out of range, in such cases NA values be printed as output.
   Example:
   X< - c(1,2,3)
   X[4]
   Output: NA

2) **INF and -INF**: When a number is to large for R to represent, the value is given as Infinite.
   Example:
   r> 1 / 0
   Output: INF

3) **Nan (Not a Number):** In some situations, it is impossible to express the result of calculation using number, in such cases Nan is given as the output.
   Example: factorial(-66)
   s

4) **NULL:** This value is used to explicitly define an empty entity.
   Example: f< - NULL
   Print(f)

   Output: NULL

Classes:

In R, a class is a blueprint or a template for creating objects and defining their structure, attributes, and methods (functions) that operate on those objects.

- Classes and Objects are basic concepts of Object-Oriented Programming that revolve around the real-life entities.
- A **class** is just a blueprint or a sketch of these objects. It represents the set of properties or methods that are common to all objects of one type.
- In R programming, classes are used to define data structures and objects. They facilitate the creation of custom data types and provide a way to bundle data and functions together.

TYPES OF CLASSES

1)S3 Class

2)S4 Class

3)S5 Class (Reference class)

S3 Class:

- S3 classes are simple and flexible. They don't enforce formal class definitions.
- An object is given a class attribute, allowing functions to determine how to handle it based on this attribute.
- Creating an S3 class involves using the **class()** function and defining methods that work with objects of that class.

S3 Class:Syntax

```
class name <- list(classname,classmembers)
class name ←Object / create object name
```

Example:

```
student <- list(student.name = c("Ram","Ravi"),age=c(29,12))
student
```

Output:

```
$name
[1] "Ram"  "Ravi"

$age
[1] 29 12
```

S4 class:

- S4 classes are more formal and structured compared to S3. They offer stricter definitions and encapsulation.
- They use **setClass()** to define classes and **setMethod()** to define methods.
- To create an object, we use the new() function.

Syntax

```
setClass ("name of the class", member variable)
objectname←new(classname,value of the member variable)
Objectname
```

Example:

```
# create a class "Student_Info" with three member variables
setClass("Student_Info",slots=list(name="character", age="numeric", GPA="numeric"))

# create an object of class
student1 <- new("Student_Info", name = "John", age = 21, GPA = 3.5)

# call student1 object
student1
}
```

Output:

```
An object of class "student_infor"
Slot "name":
[1] "Shubham"

Slot "age":
[1] 22

Slot "GPA":
[1] 3.5
```

S5 class:

- Reference Class is an improvement over S4 Class. Here the methods belong to the classes. These are much similar to object-oriented classes of other languages.
- Defining a Reference class is similar to defining S4 classes.
- We use **setRefClass()** instead of **setClass()**

Syntax

```
Classname←setRefClass(name of the class,member variable)
objectname<-classname(name of the class,value of the member)
objectname
```

Example:

```
# create a class "Student_Info" with three member variables
Student_Info <- setRefClass("Student_Info",
  fields = list(name = "character", age = "numeric", GPA = "numeric"))
# Student_Info() is our generator function which can be used to create new objects
student1 <- Student_Info(name = "John", age = 21, GPA = 3.5)
# call student1 object
student1
```

Coercion

In R programming, converting from one object or data type to another object or data type is referred as coercion.

Ther are two types of coercion. They are:

Implicit coercion: This type of coercion occurs automatically.

Example: The logical value True will be treated as 1 and False will be treated as 0.

Explicit coercion: This type of coercion can be done with the help of as.integer, as.logical etc., functons.

Example:

```
X <- c(0,1,0,3)
Class (x)
Output: "numeric"
as.integer(x)
Class(x)
Output: "integer"
as.complex(x)
Class(x)
Output: "complex"
```

Basic plotting in R:

The plot() function is used to draw points (markers) in a diagram.

The function takes parameters for specifying points in the diagram.

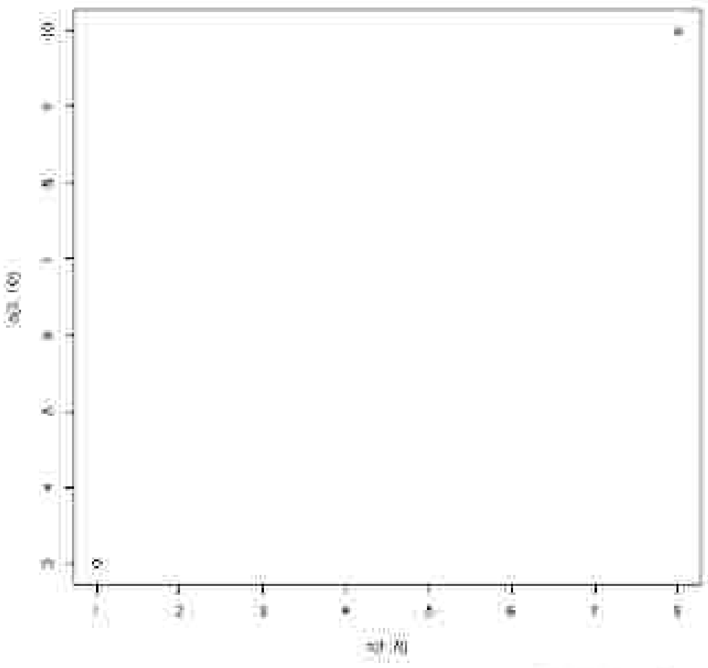
Parameter 1 specifies points on the **x-axis**.

Parameter 2 specifies points on the **y-axis**.

Example

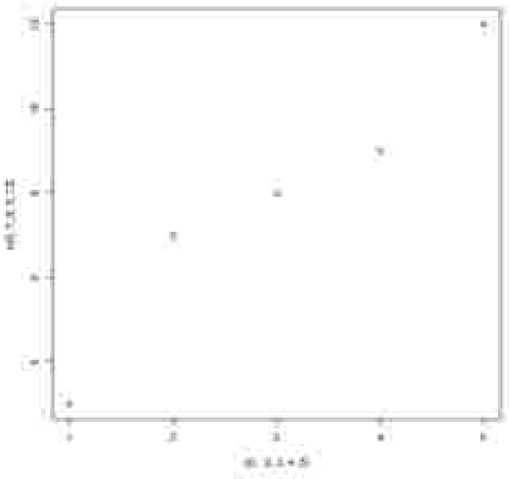
Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

1) plot(c(1, 8), c(3, 10))



2) x<-c(1, 2, 3, 4, 5)
   y<-c(3, 7, 8, 9, 12)

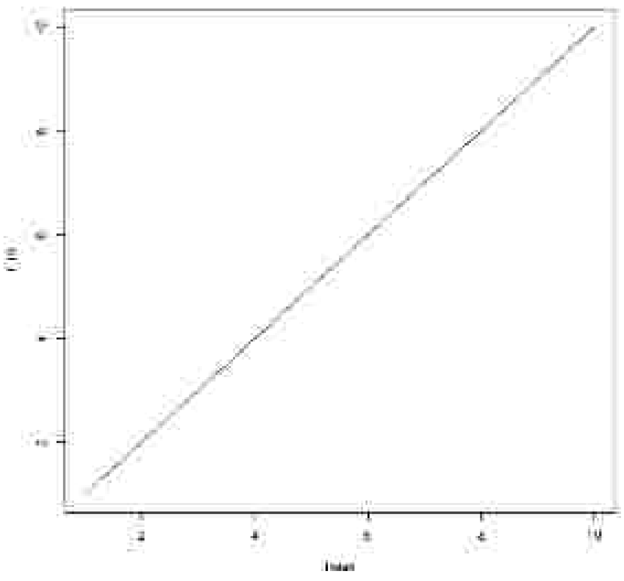
   plot(x, y)



Draw a Line

The plot() function also takes a type parameter with the value l to draw a line to connect all the points in the diagram:

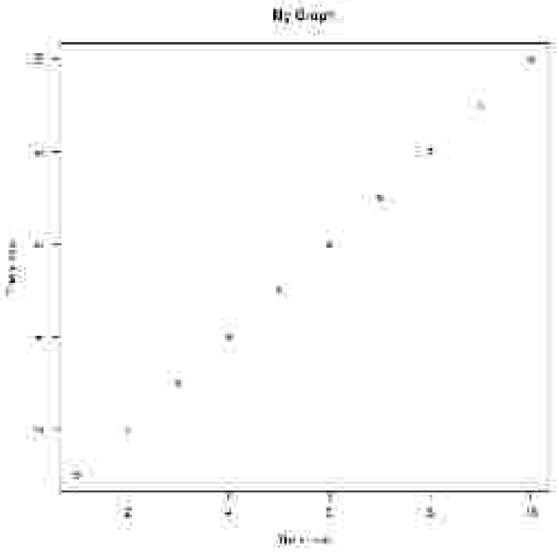
plot(1:10, type="l")



Plot Labels

The plot() function also accept other parameters, such as main, xlab and ylab if you want to customize the graph with a main title and different labels for the x and y-axis:

Example: plot(1:10, main="My Graph", xlab="The x-axis", ylab="The y axis")

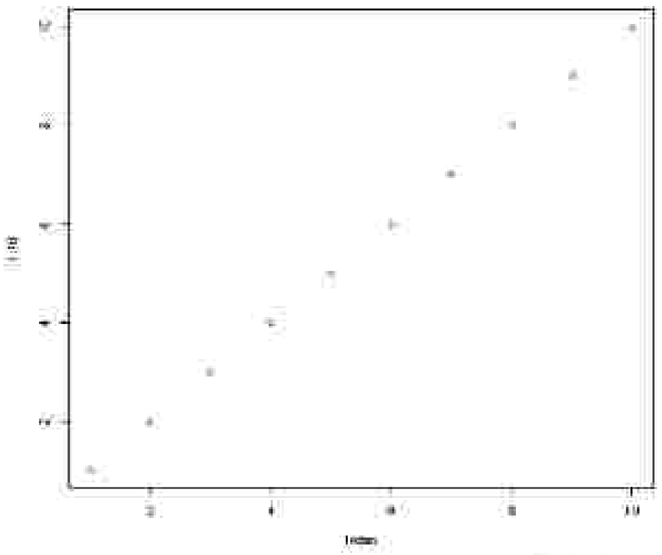


Colors

Use col="*color*" to add a color to the points:

Example:

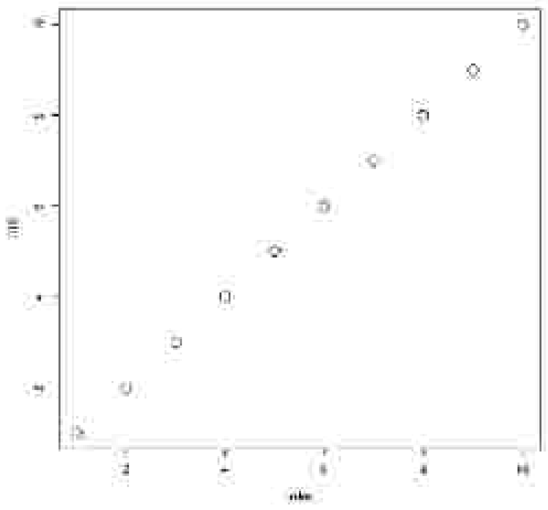
plot(1:10, col="red")



Size

Use cex=*number* to change the size of the points

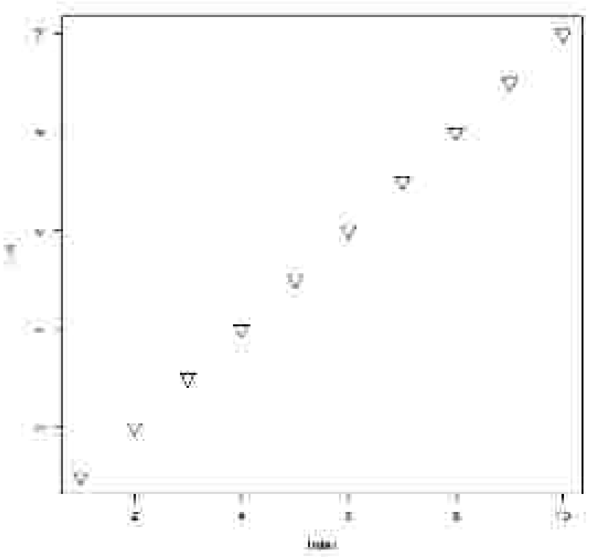
Example: plot(1:10, cex=2)



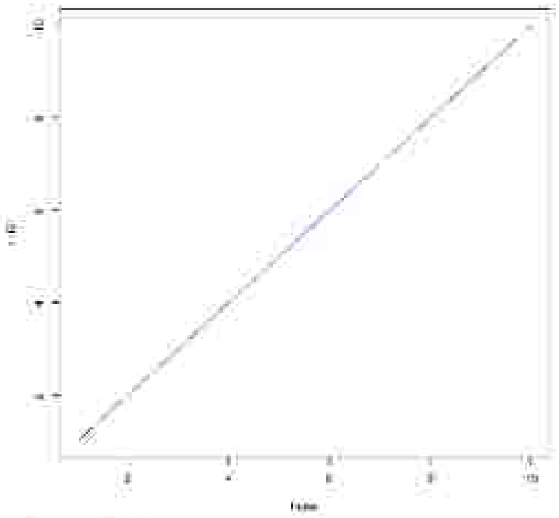
Point Shape

Use pch with a value from 0 to 25 to change the point shape format:

Example: plot(1:10, pch=25, cex=2)



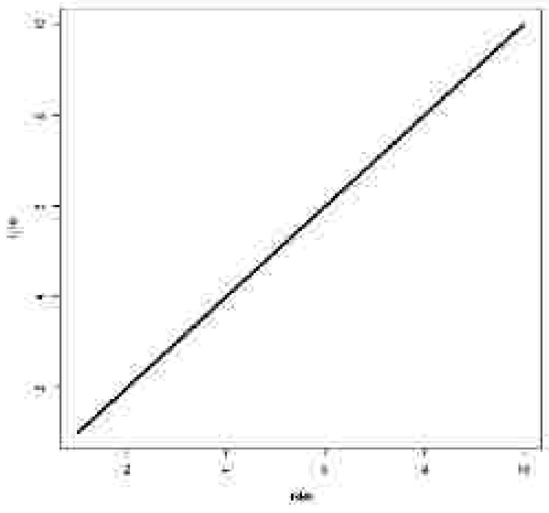
To change the colour of line



Example: plot(1:10, type="l", colour="blue")

To change the width of the line:

Example: plot(1:10, type="l", lwd=2)



******************************END**********************************

Module-01 questions

2 Marks Questions:

Module-01

1.Who is invented by R Program. The file extension of R program

2.What is Operator.ExpalinArithmetic operator in R.

3.What are all the datatypes using in R

4.What are all the special values in R.

5.What is data frame with syntax.

3Marks Questions:

1.To solve the equation using PEMDAS Rule: $\frac{8^3+(9-2)}{87-3+9}$

2.To solve the equation using PEMDAS Rule: $\left(\frac{3*5^2}{15}\right) - (5 - 2^2)$

3.What are all the data strctures in R programming

4.What are all the data types in R programming

5.To implement reverse a given number using rev(),strsplit()

5 Marks Questions:

1.Define Data frame with syntax. To create a data frame name "Student information" data frame fields are Name, Age,Class,Mobile Number filled with at least five data fields.

2.To implants calculator program using arithmetic operators and switch statement in R.

3.what is class. Explain S3,S4,S5 class (reference class)with syntax and example.

4.Explain Basic Plotting with example.

5.Explain in detail Array data structures and Matric data structures syntax with example.

10 Marks Questions:

1.Explian the features and Application of R programming.

2. Explain Types Data structures in Detail

3. Define operators explain types of operators with example

4. Define class and object. Explain class and types with example program.

5. Define variable,constant,data types. To build one Hospital service related R program include variables and constants and data types.