

R Programming



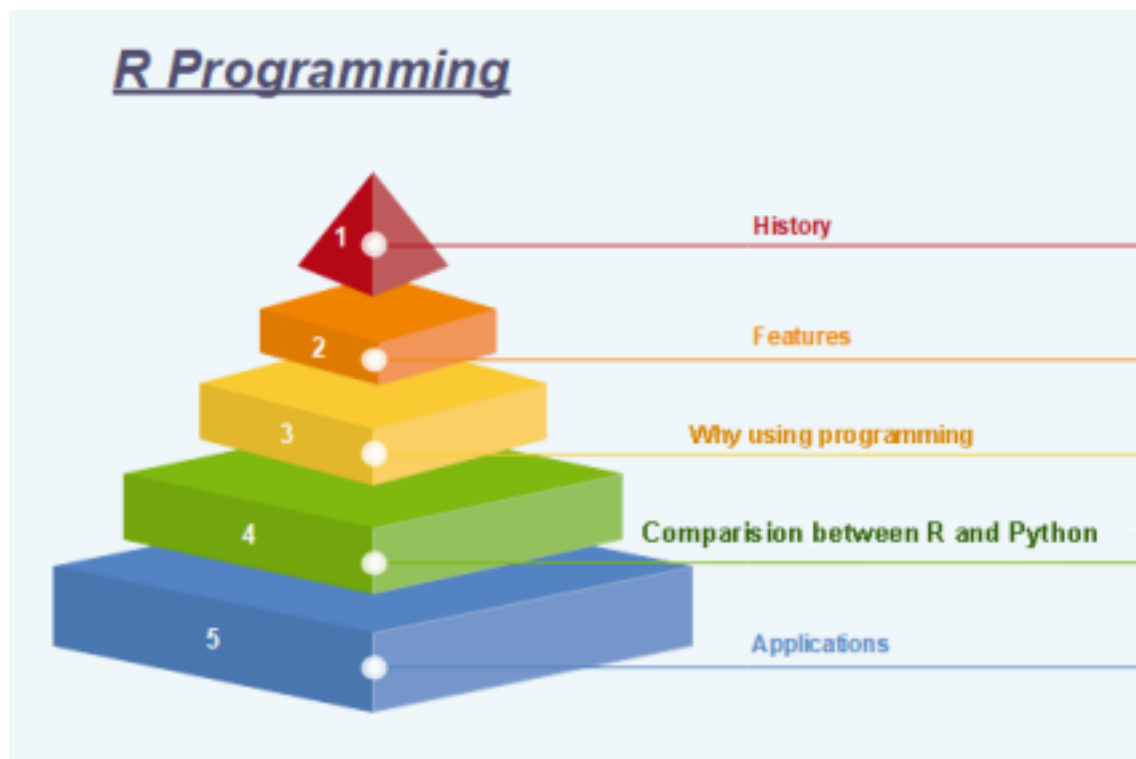
R is a software environment which is used to analyze statistical information and graphical representation. R allows us to do modular programming using functions.

What is R Programming

"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, which is combined with **lexical scoping semantics**.

R not only allows us to do branching and looping but also allows to do modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python, and FORTRAN languages to improve efficiency.

In the present era, R is one of the most important tool which is used by researchers, data analyst, statisticians, and marketers for retrieving, cleaning, analyzing, visualizing, and presenting data.



SHREE

History of R Programming

The history of R goes back about 20-30 years ago. R was developed by Ross Ihaka 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.



The following table shows the release date, version, and description of R language:

Version	Date	Description	
Release			
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.	

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA

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.	

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. These are the following features of R programming:

1. It is a simple and effective programming language which has been well developed.
2. It is data analysis software.
3. It is a well-designed, easy, and effective language which has the concepts of user-defined, looping, conditional, and various I/O facilities.
4. It has a consistent and incorporated set of tools which are used for data analysis. 5. For different types of calculation on arrays, lists and vectors, R contains a suite of operators.

6. It provides effective data handling and storage facility.
7. It is an open-source, powerful, and highly extensible software.
8. It provides highly extensible graphical techniques.
9. It allows us to perform multiple calculations using vectors.
10. 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. 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 has a package to perform **Xgboost**. Xgboost is one of the best algorithms for **Kaggle competition**.

R communicate with the other languages and possibly calls Python, Java, C++. The big data world is also accessible to R. We can connect R with different databases like **Spark** or **Hadoop**.

In brief, 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.

Comparison between R and Python

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. But still, it becomes confusing to choose the better or the most suitable one among the two, R and Python.

<div>Comparison</div> <div>R Python</div> <div>Index</div>		
Overview	"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	Python is an Interpreted high-level programming language used for general purpose programming. Guido Van Rossum created it, and it was first

	Development Core Team currently develops R. R is also a software environment which is used to analyze statistical information, graphical representation, reporting, and data modeling.	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.
Specialties for data science	R packages have advanced techniques which are very useful for statistical work. The CRAN text view is provided by many useful R packages. These packages cover everything from Psychometrics to Genetics to Finance.	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.
Functionalities	For data analysis, R has inbuilt functionalities	Most of the data analysis functionalities are not inbuilt. They are available through packages like Numpy and Pandas
Key domains of application	Data visualization is a key aspect of analysis. R packages such as ggplot2, ggvis, lattice, etc. make data visualization easier.	Python is better for deep learning because Python packages such as Caffe, Keras, OpenNN, etc. allows the development of the deep neural network in a very simple way.
Availability of packages	There are hundreds of packages and ways to accomplish needful data science tasks.	Python has few main packages such as viz, Scikit learn, and Pandas for data analysis of machine learning, respectively.

Applications of R

There are several-applications available in real-time. Some of the popular applications are as follows:

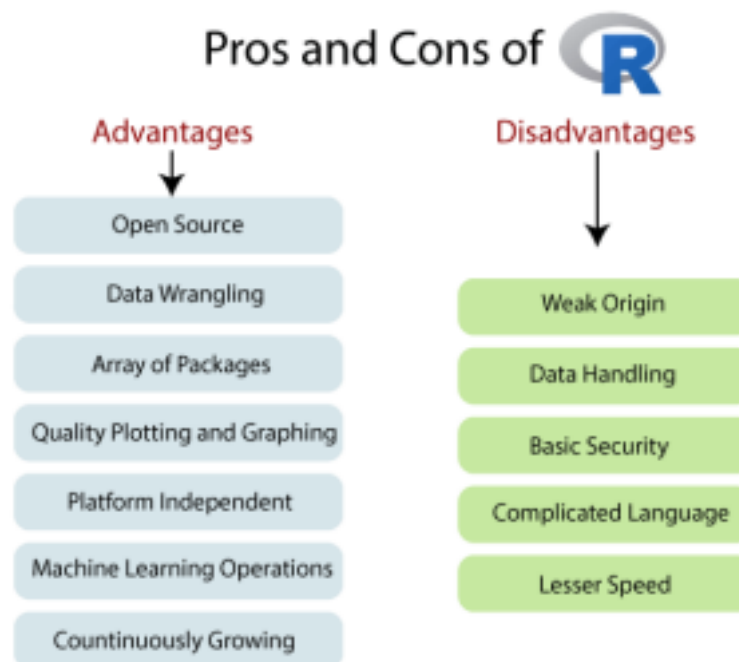
- Facebook
- Google
- Twitter
- HRDAG
- Sunlight Foundation
- RealClimate
- NDAA

- XBOX ONE
- ANZ
- FDA

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.

There are the following pros and cons of R



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. We can contribute to the development of R by optimizing our packages, developing new ones, and resolving issues.

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

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

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, like our taste in music and clothes, which evolve as we get older. R is a state of the art which provides updates whenever any new feature is added.

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. It is not an ideal option when we deal with Big Data.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

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.

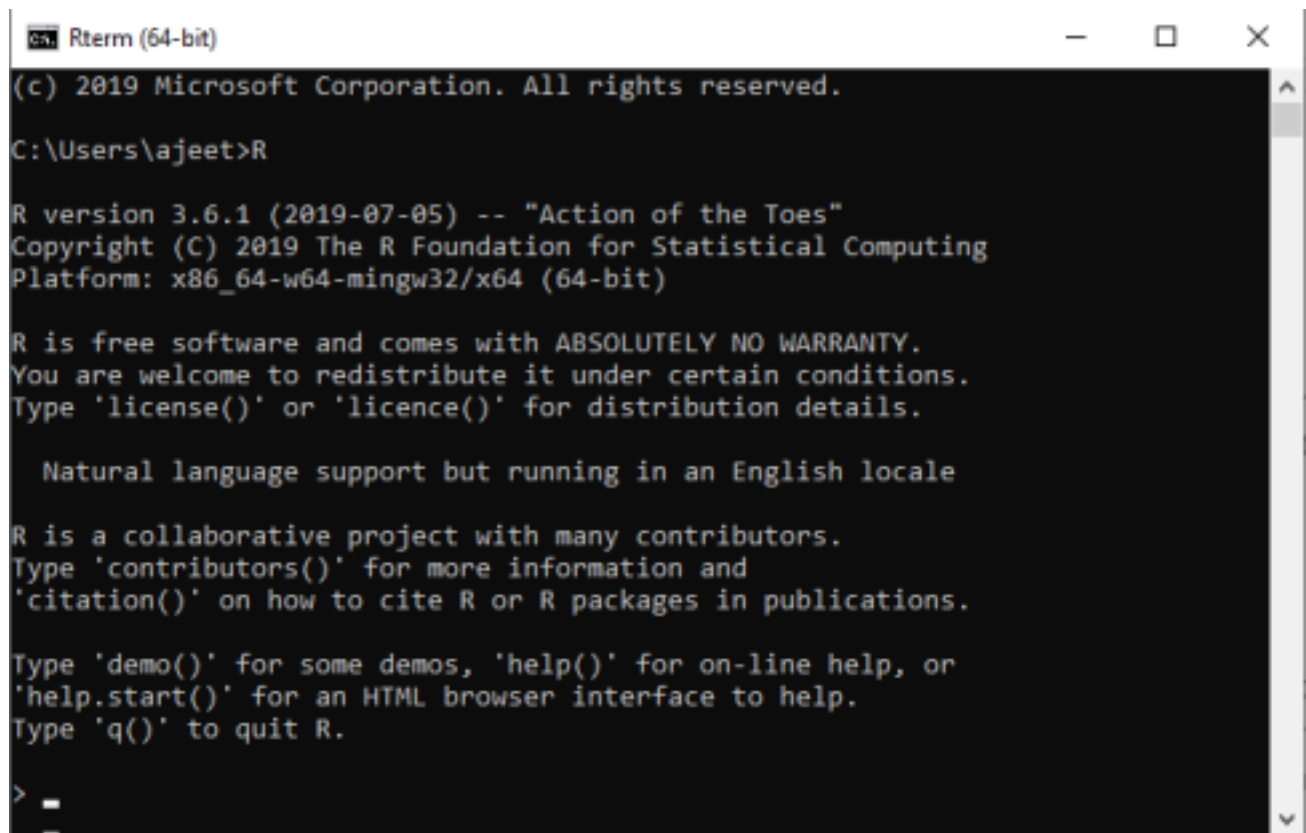
Syntax of R Programming

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, and now we will understand the syntax of R programming with "Hello world" program. We can write our code either in command prompt, or we can use an R script file.

R Command Prompt

It is required that we have already installed the R environment set up in our system to work on the R command prompt. After the installation of R environment setup, we can easily start R command prompt by typing R in our Windows command prompt. When we press enter after typing R, it will launch interpreter, and we will get a prompt on which we can code our program.

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA

A screenshot of an Rterm window titled "Rterm (64-bit)". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The terminal text shows the R startup sequence: a copyright notice for 2019 Microsoft Corporation, the command "C:\Users\ajeet>R", the R version "3.6.1 (2019-07-05)" with the slogan "Action of the Toes", copyright for The R Foundation, the platform "x86_64-w64-mingw32/x64 (64-bit)", a disclaimer about warranty, information about natural language support, contributors, and help options. The prompt ">" is visible at the bottom.

```
Rterm (64-bit)

(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>R

R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> _
```

"Hello, World!" Program

The code of "Hello World!" in R programming can be written as:

A screenshot of an R console window showing the execution of a simple R script. The user enters two commands: "string <- 'Hello World!'" and "print(string)". The output is "[1] 'Hello World!'", indicating the string has been printed. The prompt ">" is visible at the bottom.

```
> string <- "Hello World!"
> print(string)
[1] "Hello World!"
> _
```

In the above code, the first statement defines a **string variable** string, where we assign a string "Hello World!". The next statement print() is used to print the value which is stored in the variable string.

R Script File

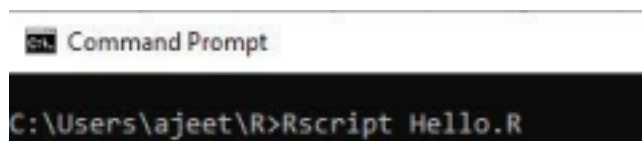
The R script file is another way on which we can write our programs, and then we execute those scripts at our command prompt with the help of R interpreter known as **Rscript**. We make a text file and write the following code. We will save this file with .R extension as:

Demo.R

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

1. `string <- "Hello World!"`
2. `print(string)`

To execute this file in Windows and other operating systems, the process will remain the same as mentioned below.



```
Command Prompt
C:\Users\ajeet\R>Rscript Hello.R
```

When we press enter it will give us the following output:



```
[1] "Hello World!"
```

Comments

In R programming, comments are the programmer readable explanation in the source code of an R program. The purpose of adding these comments is to make the source code easier to understand. These comments are generally ignored by compilers and interpreters.

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 programming language which was created by  
Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand "  
}  
#My First program in R programming  
string <- "Hello World!"  
print(string)
```

SHREE MEDHA DEGREE COLLEGE MANJESH M

Command Prompt

```
C:\Users\ajeet\R>Rscript Hello.R  
[1] "Hello World!"  
C:\Users\ajeet\R>
```

R PROGRAMMING V - SEM BCA

Data Types in R Programming

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:



Data type	Example	Description
Logical	True, False	It is a special data type for data with only two possible values which can be construed as true/false.
Numeric	12,32,112,5432	Decimal value is called numeric in R, and it is the default computational data type.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

Integer	3L, 66L, 2346L	Here, L tells R to store the value as an integer,
Complex	Z=1+2i, t=7+3i	A complex value in R is defined as the pure imaginary value i.
Character	'a', '"good"', "TRUE", '35.4'	In R programming, a character is used to represent string values. We convert objects into character values with the help of as.character() function.
Raw		A raw data type is used to holds raw bytes.

Let's see an example for better understanding of data

types: **#Logical Data type**

```
variable_logical<- TRUE
```

```
cat(variable_logical,"\n")
```

```
cat("The data type of variable_logical is ",class(variable_logical),"\n\n")
```

#Numeric Data type

```
variable_numeric<- 3532
```

```
cat(variable_numeric,"\n")
```

```
cat("The data type of variable_numeric is ",class(variable_numeric),"\n\n")
```

#Integer Data type

```
variable_integer<- 133L
```

```
cat(variable_integer,"\n")
```

```
cat("The data type of variable_integer is ",class(variable_integer),"\n\n")
```

#Complex Data type

```
variable_complex<- 3+2i
```

```
cat(variable_complex,"\n")
```

```
cat("The data type of variable_complex is ",class(variable_complex),"\n\n")
```

#Character Data type

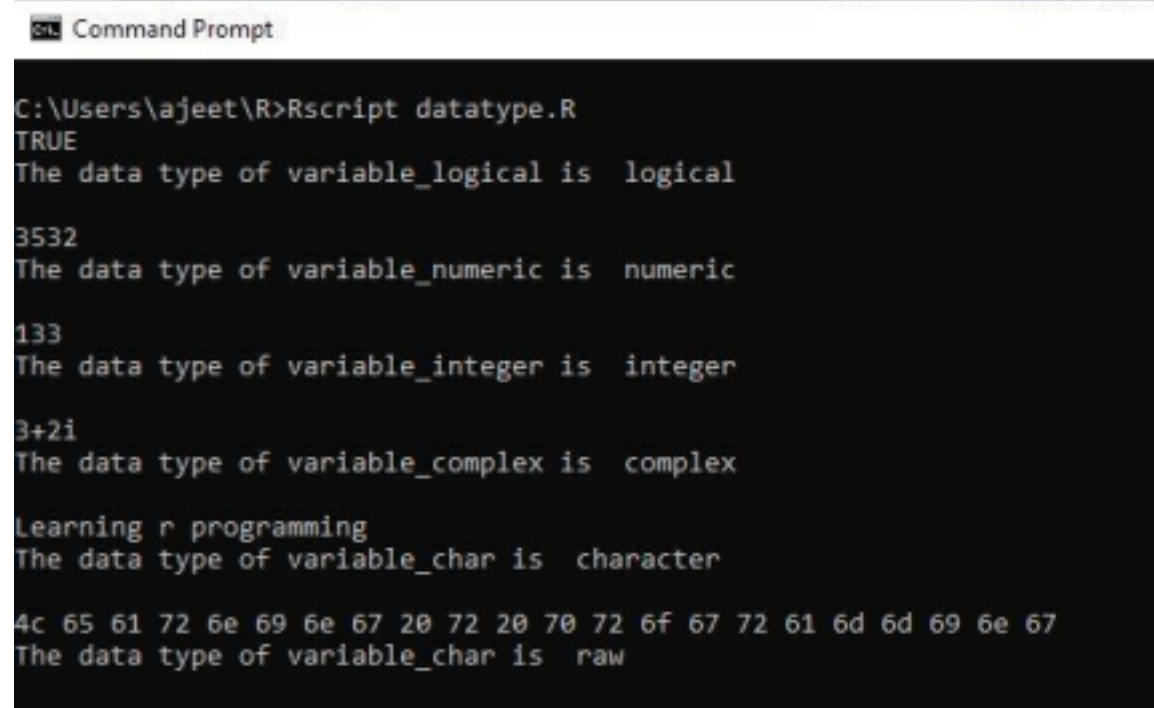
```
variable_char<- "Learning r programming"  
cat(variable_char,"\\n")  
cat("The data type of variable_char is ",class(variable_char),"\\n\\n")
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

#Raw Data type

```
variable_raw<- charToRaw("Learning r programming")  
cat(variable_raw,"\\n")  
cat("The data type of variable_char is ",class(variable_raw),"\\n\\n")
```

When we execute the following program, it will give us the following output:



```
C:\Users\ajet\R>Rscript datatype.R  
TRUE  
The data type of variable_logical is  logical  
  
3532  
The data type of variable_numeric is  numeric  
  
133  
The data type of variable_integer is  integer  
  
3+2i  
The data type of variable_complex is  complex  
  
Learning r programming  
The data type of variable_char is  character  
  
4c 65 61 72 6e 69 6e 67 20 72 20 70 72 6f 67 72 61 6d 6d 69 6e 67  
The data type of variable_char is  raw
```

Data Structures in R Programming

Data structures are very important to understand. Data structure are the objects which we will manipulate in our day-to-day basis in R. Dealing with object conversions is the most common sources of despairs for beginners. We can say that everything in R is an object.

R has many data structures, which include:

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**



1. Atomic vector
2. List
3. Array
4. Matrices
5. Data Frame
6. Factors

Vectors

A vector is the basic data structure in R, or we can say vectors are the most basic R data objects. There are six types of atomic vectors such as logical, integer, character, double, and raw. **"A vector is a collection of elements which is most commonly of mode character, integer, logical or numeric"**. They can be created using the `c()` function.

```
nv<- c(1,2,3,4,5)
```

```
cv<- c("apple", "banana", "cherry")
```

A vector can be one of the following two types:

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

1. Atomic vector
2. Lists

List

In R, **the list** is the container. Unlike an atomic vector, the list is not restricted to be a single mode. A list contains a mixture of data types. The list is also known as generic vectors because the element of the list can be of any type of R object. **"A list is a special type of vector in which each element can be a different type."**

We can create a list with the help of `list()` or `as.list()`. We can use `vector()` to create a required length empty list.

Arrays

There is another type of data objects which can store data in more than two dimensions known as arrays. **"An array is a collection of a similar data type with contiguous memory allocation."** Suppose, if we create an array of dimension (2, 3, 4) then it creates four rectangular matrices of two rows and three columns.

In R, an array is created with the help of `array()` function. This function takes a vector as an input and uses the value in the `dim` parameter to create an array.

Matrices

A matrix is an R object in which the elements are arranged in a two-dimensional rectangular layout. In the matrix, elements of the same atomic types are contained. For mathematical calculation, this can use a matrix containing the numeric element. A matrix is created with the help of the `matrix()` function in R.

Syntax

The basic syntax of creating a matrix is as follows:

1. `matrix(data, no_row, no_col, by_row, dim_name)`

Data Frames

A **data frame** is a two-dimensional array-like structure, or we can say it is a table in which each column contains the value of one variable, and row contains the set of value from each column.

There are the following characteristics of a data frame:

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

1. The column name will be non-empty.
2. The row names will be unique.
3. A data frame stored numeric, factor or character type data.
4. Each column will contain same number of data items.

Factors

Factors are also data objects that are used to categorize the data and store it as levels. Factors can store both strings and integers. Columns have a limited number of unique values so that factors are very useful in columns. It is very useful in data analysis for statistical modeling.

Factors are created with the help of **factor()** function by taking a vector as an input parameter.

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.

Language like C++ is statically typed, but R is a dynamically typed, means it check the type of data type when the statement is run. A valid variable name contains letter, numbers, dot and underlines characters. A variable name should start with a letter or the dot not followed by a number.

Name of variable

**Validity Reason for valid and
invalid**

<code>_var_name</code>	Invalid
<code>var_name,</code> <code>var.name</code>	Valid
<code>var_name%</code>	Invalid
<code>2var_name</code>	Invalid
<code>.2var_name</code>	Invalid

Variable name can't start with an underscore(_).

Variable can start with a dot, but dot should not be followed by a number. In this case the variable will be invalid.

In R, we can't use any special character in the variable name

except dot and underscore. Variable name can't start with a

numeric digit.

A variable name cannot start with a dot which is followed by a digit.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

<code>var_name2</code>	Valid
------------------------	-------

The variable contains letter, number and underscore and starts with a letter.

Assignment of variable

In R programming, there are three operators which we can use to assign the values to the variable. We can use leftward, rightward, and equal_to operator for this purpose.

There are two functions which are used to print the value of the variable i.e., `print()` and `cat()`. The `cat()` function combines multiple values into a continuous print output.

Assignment using equal operator.

```
variable.1 = 124
```

Assignment using leftward operator.

```
variable.2 <- "Learn R Programming"
```

Assignment using rightward operator.

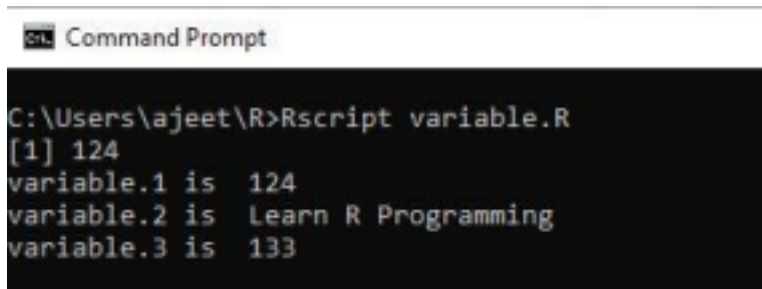
```
133L -> variable.3
```

```
print(variable.1)
```

```
cat ("variable.1 is ", variable.1 , "\n")
```

```
cat ("variable.2 is ", variable.2 ,"\n")
cat ("variable.3 is ", variable.3 ,"\n")
```

When we execute the above code in our R command prompt, it will give us the following output:



```
C:\Users\ajeet\R>Rscript variable.R
[1] 124
variable.1 is 124
variable.2 is Learn R Programming
variable.3 is 133
```

Data types of variable

R programming is a dynamically typed language, which means that we can change the data type of the same variable again and again in our program. Because of its dynamic nature, a variable is

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA

not declared of any data type. It gets the data type from the R-object, which is to be assigned to the variable.

We can check the data type of the variable with the help of the `class()` function. Let's see an example:

```
variable_y<- 124
cat("The data type of variable_y is ",class(variable_y),"\n")
```

```
variable_y<- "Learn R Programming"
cat(" Now the data type of variable_y is ",class(variable_y),"\n")
```

```
variable_y<- 133L
cat(" Next the data type of variable_y becomes ",class(variable_y),"\n")
```

When we execute the above code in our R command prompt, it will give us the following output:

```
Command Prompt
C:\Users\ajeet\R>Rscript datatype.R
The data type of variable_y is numeric
Now the data type of variable_y is character
Next the data type of variable_y becomes integer
```

Keywords

in R Programming

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."

There are the following keywords as per **?reserved** or **help(reserved)** command:

if	else	repeat for
while	function	TRUE
next	break	

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

FALSE	NULL
NaN	NA
NA_real_	NA_complex_

Inf

NA_integer_

NA_character_



Operators in R

In **computer programming**, an operator is a symbol which represents an action. An operator is a symbol which tells the compiler to perform specific **logical** or **mathematical** manipulations. R programming is very rich in built-in operators.

In **R programming**, there are different types of operator, and each operator performs a different task. For data manipulation, There are some advance operators also such as model formula and list indexing.

There are the following types of operators used in R:

SHREE MEDHA DEGREE COLLEGE MANJESH M



R PROGRAMMING V - SEM BCA

1. [Arithmetic Operators](#)
2. [Relational Operators](#)
3. [Logical Operators](#)
4. [Assignment Operators](#)
5. [Miscellaneous Operators](#)

Arithmetic Operators

Arithmetic operators are the symbols which are used to represent arithmetic math operations. The operators act on each and every element of the vector. There are various arithmetic operators which are supported by R.

S. No	Operator Description Example		
1.	+	This operator is used to add two vectors in R. <code>a <- c(2, 3.3, 4)</code>	<code>b <- c(11, 5, 3)</code> <code>print(a+b)</code> It will give us the following output: [1] 13.0 8.3 5.0
2.	-	This operator is used to divide a vector from another one. <code>a <- c(2, 3.3, 4)</code>	<code>b <- c(11, 5, 3)</code> <code>print(a-b)</code> It will give us the following output:

			[1] -9.0 -1.7 3.0
3.	*	This operator is used to multiply two vectors with each other. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a*b)</pre> <p>It will give us the following output: [1] 22.0 16.5 4.0</p>
4.	/	This operator divides the vector from another one. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a/b)</pre> <p>It will give us the following output: [1] 0.1818182 0.6600000 4.0000000</p>
5.	%%	This operator is used to find the remainder of the first vector with the second vector. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a%%b)</pre> <p>It will give us the following output: [1] 2.0 3.3 0</p>
6.	%/%	This operator is used to find the division of the first vector with the second(quotient).	<pre>a <- c(2, 3.3, 4) b <- c(11, 5, 3) print(a%/%b)</pre> <p>It will give us the following output: [1] 0 0 4</p>
7.	^	This operator raised the first vector to the exponent of the second vector. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a^b)</pre> <p>It will give us the following output: [1] 0248.0000 391.3539 4.0000</p>

Relational Operators

A relational operator is a symbol which defines some kind of relation between two entities. These include numerical equalities and inequalities. A relational operator compares each element of the first vector with the corresponding element of the second vector. The result of the comparison will be a Boolean value. There are the following relational operators which are supported by R:

S. No		Operator Description Example	
1.	>	This operator will return TRUE when every element in the first vector is greater than the corresponding element of the second vector.	<pre>a <- c(1, 3, 5) b <- c(2, 4, 6) print(a>b)</pre> <p>It will give us the following output:</p> <pre>[1] FALSE FALSE FALSE</pre>
2.	<	This operator will return TRUE when every element in the first vector is less then the corresponding element of the second vector.	<pre>a <- c(1, 9, 5) b <- c(2, 4, 6) print(a<b)</pre> <p>It will give us the following output:</p> <pre>[1] FALSE TRUE FALSE</pre>
3.	<=	This operator will return TRUE when every element in the first vector is less than or equal to the corresponding element of another vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a<=b)</pre> <p>It will give us the following output:</p> <pre>[1] TRUE TRUE TRUE</pre>
4.	>=	This operator will return TRUE when every element in the first vector is greater than or equal to the corresponding element of another vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a>=b)</pre> <p>It will give us the following output:</p> <pre>[1] FALSE TRUE FALSE</pre>
5.	==	This operator will return TRUE when every element in the first vector is equal to the corresponding element of the second vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a==b)</pre>

			It will give us the following output: [1] FALSE TRUE FALSE
6.	!=	This operator will return TRUE when every element in the first vector is not equal to the corresponding element of the second vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a>=b)</pre> It will give us the following output: [1] TRUE FALSE TRUE

Logical Operators

The logical operators allow a program to make a decision on the basis of multiple conditions. In the program, each operand is considered as a condition which can be evaluated to a false or true value. The value of the conditions is used to determine the overall value of the op1 **operator** op2. Logical operators are applicable to those vectors whose type is logical, numeric, or complex.

The logical operator compares each element of the first vector with the corresponding element of the second vector.

There are the following types of operators which are supported by R:

S. Operator Description Example No			
1.	&	This operator is known as the Logical AND operator. This operator takes the first element of both the vector and returns TRUE if both the elements are TRUE.	<pre>a <- c(3, 0, TRUE, 2+2i) b <- c(2, 4, TRUE, 2+3i) print(a&b)</pre> It will give us the following output: [1] TRUE FALSE TRUE TRUE

2.		This operator is called the Logical OR operator. This operator takes the first element of both the vector and returns TRUE if one of them is TRUE.	<pre>a <- c(3, 0, TRUE, 2+2i) b <- c(2, 4, TRUE, 2+3i) print(alb)</pre> <p>It will give us the following output:</p> <pre>[1] TRUE TRUE TRUE TRUE</pre>
3.	!	This operator is known as Logical NOT operator. This operator takes the first element of the vector and gives the opposite logical value as a result.	<pre>a <- c(3, 0, TRUE, 2+2i) print(!a)</pre> <p>It will give us the following output:</p> <pre>[1] FALSE TRUE FALSE FALSE</pre>
4.	&&	This operator takes the first element of both the vector and gives TRUE as a result, only if both are TRUE.	<pre>a <- c(3, 0, TRUE, 2+2i) b <- c(2, 4, TRUE, 2+3i) print(a&& b)</pre> <p>It will give us the following output:</p> <pre>[1] TRUE</pre>
5.		This operator takes the first element of both the vector and gives the result TRUE, if one of them is true.	<pre>a <- c(3, 0, TRUE, 2+2i) b <- c(2, 4, TRUE, 2+3i) print(allb)</pre> <p>It will give us the following output:</p> <pre>[1] TRUE</pre>

Assignment Operators

An assignment operator is used to assign a new value to a variable. In R, these operators are used to assign values to vectors. There are the following types of assignment

S. No Operator Description Example			
1.	<- or = or <<-	These operators are known as left assignment operators.	<pre>a <- c(3, 0, TRUE, 2+2i) b <<- c(2, 4, TRUE, 2+3i) d = c(1, 2, TRUE, 2+3i) print(a) print(b) print(d)</pre> <p>It will give us the following output:</p> <pre>[1] 3+0i 0+0i 1+0i 2+2i [1] 2+0i 4+0i 1+0i 2+3i [1] 1+0i 2+0i 1+0i 2+3i</pre>
2.	-> or ->>	These operators are known as right assignment operators.	<pre>c(3, 0, TRUE, 2+2i) -> a c(2, 4, TRUE, 2+3i) ->> b print(a) print(b)</pre> <p>It will give us the following output:</p> <pre>[1] 3+0i 0+0i 1+0i 2+2i [1] 2+0i 4+0i 1+0i 2+3i</pre>

operators which are supported by R:

Miscellaneous Operators

Miscellaneous operators are used for a special and specific purpose. These operators are not used for general mathematical or logical computation. There are the following miscellaneous operators which are supported in R

S. No Operator Description Example			
1.	:	The colon operator is used to create the series of numbers in sequence for a vector.	<pre>v <- 1:8 print(v)</pre> <p>It will give us the following output: [1] 1 2 3 4 5 6 7 8</p>

2.	<code>%in%</code>	This is used when we want to identify if an element belongs to a vector.	<pre>a1 <- 8 a2 <- 12 d <- 1:10 print(a1 %in%t) print(a2 %in%t)</pre> <p>It will give us the following output:</p> <pre>[1] FALSE [1] FALSE</pre>
3.	<code>%*%</code>	It is used to multiply a matrix with its transpose.	<pre>M=matrix(c(1,2,3,4,5,6), nrow=2, ncol=3, byrow=TRUE) T=m%*%T(m) print(T)</pre> <p>It will give us the following output:</p> <pre>14 32 32 77</pre>

1) if

The if statement consists of a Boolean expression which is followed by one or more statements. In R, if statement is the simplest conditional statement which is used to decide whether a block of the statement will be executed or not.

Example:

Backward Skip 10sPlay VideoForward Skip 10s

1. `a<-11`
2. `if(a<15)`
3. + `print("I am lesser than 15")`

Output:

2) else

The R else statement is associated with if statement. When the if statement's condition is false only then else block will be executed. Let see an example to make it clear:

Example:

R PROGRAMMING V - SEM BCA

```
1. a<-22
2. if(a<20){
3. cat("I am lesser than 20")
4. }else{
5. cat("I am larger than 20")
6. }
```

Output:



3) repeat

The repeat keyword is used to iterate over a block of code multiple numbers of times. In R, repeat is a loop, and in this loop statement, there is no condition to exit from the loop. For exiting the loop, we will use the break statement.

Example:

```
1. x <- 1
2. repeat {
3. cat(x)
4. x = x+1
5. if (x == 6){
6. break
7. }
8. }
```

Output:



4) while

R PROGRAMMING V - SEM BCA

A while keyword is used as a loop. The while loop is executed until the given condition is true. This is also used to make an infinite loop.

Example:

```
1. a <- 20
2. while(a!=0){
3. cat(a)
4. a = a-2
5. }
```

Output:



5) function

A function is an object in R programming. The keyword function is used to create a user-defined function in R. R has some pre-defined functions also, such as seq, mean, and sum.

Example:

```
1. new.function<- function(n) {
2. for(i in 1:n) {
3. a <- i^2
4. print(a)
5. }
6. }
7. new.function(6)
```

Output:

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**



6) for

The **for** is a keyword which is used for looping or iterating over a sequence (dictionary, string, list, set or tuple).

We can execute a set of a statement once for each item in the iterator (list, set, tuple, etc.) with the help of for loop.

Example:

1. `v <- LETTERS[1:4]`
2. `for (i in v) {`
3. `print(i)`
4. `}`

Output:



7) next

The **next** keyword skips the current iteration of a loop without terminating it. When R parser found **next**, it skips further evaluation and starts the new iteration of the loop.

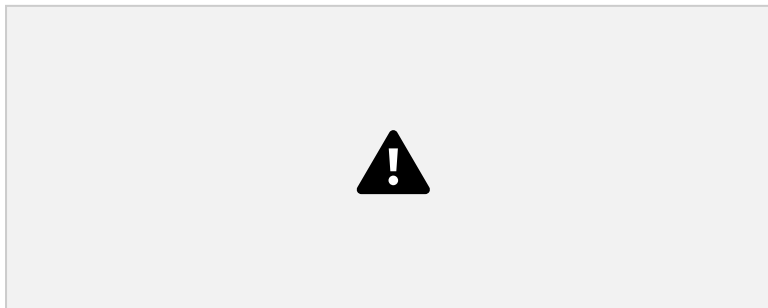
Example:

```
1. v <- LETTERS[1:6]
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

```
2. for ( i in v) {  
3.   if (i == "D") {  
4.     next  
5.   }  
6.   print(i)  
7. }
```

Output:



8) break

The **break** keyword is used to terminate the loop if the condition is true. The control of the program firstly passes to the outer statement then passes to the body of the break statement.

Example:

```
1. n<-1  
2. while(n<10){  
3.   if(n==3)  
4.     break  
5.   n=n+1  
6.   cat(n, "\n")
```

7. }

8. cat("End of the program")

Output:

SHREE MEDHA DEGREE COLLEGE MANJESH M



R PROGRAMMING V - SEM BCA

9) TRUE/FALSE

The TRUE and FALSE keywords are used to represent a Boolean true and Boolean false. If the given statement is true, then the interpreter returns true else the interpreter returns false.



10) NULL

In R, NULL represents the null object. NULL is used to represent missing and undefined values. NULL is the logical representation of a statement which is neither TRUE nor FALSE.

Example:

1. as.null(list(a = 1, b = "c"))

Output:



11) Inf and NaN

The `is.finite` and `is.infinite` function returns a vector of the same length indicating which elements are finite or infinite.

`Inf` and `-Inf` are positive and negative infinity. `NaN` stands for 'Not a Number.' `NaN` applies on numeric values and real and imaginary parts of complex values, but it will not apply to the values of integer vectors.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

Usage

1. `is.finite(x)`
2. `is.infinite(x)`
3. `is.nan(x)`
- 4.
5. `Inf`
6. `NaN`

12) NA

`NA` is a logical constant of length 1 that contains a missing value indicator. It can be coerced to any other vector type except `raw`. There are other types of constant also, such as `NA_Integer_`, `NA_real_`, `NA_complex_`, and `NA_character`. These constants are of the other atomic vector type which supports missing values.

Usage

1. `NA`
2. `is.na(x)`
3. `anyNA(x, recursive = FALSE)`
- 4.
5. `## S3 method for class 'data.frame'`
6. `is.na(x)`
- 7.
8. `is.na(x) <- value`

symbol which tells the compiler to perform specific **logical** or **mathematical** manipulations. R programming is very rich in built-in operators.

In **R programming**, there are different types of operator, and each operator performs a different task. For data manipulation, There are some advance operators also such as model formula and list indexing.

There are the following types of operators used in R:

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**



1. [Arithmetic Operators](#)
2. [Relational Operators](#)
3. [Logical Operators](#)
4. [Assignment Operators](#)
5. [Miscellaneous Operators](#)

Arithmetic Operators

Arithmetic operators are the symbols which are used to represent arithmetic math operations. The

operators act on each and every element of the vector. There are various arithmetic operators which are supported by R.

S. Operator Description Example No			
1.	+	This operator is used to add two vectors in R. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a+b)</pre> <p>It will give us the following</p> <p>output: [1] 13.0 8.3 5.0</p>

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA

2.	-	This operator is used to divide a vector from another one. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a-b)</pre> <p>It will give us the following</p> <p>output: [1] -9.0 -1.7 3.0</p>
3.	*	This operator is used to multiply two vectors with each other. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a*b)</pre> <p>It will give us the following</p> <p>output: [1] 22.0 16.5 4.0</p>
4.	/	This operator divides the vector from another one. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a/b)</pre> <p>It will give us the following output: [1]</p> <p>0.1818182 0.6600000 4.0000000</p>
5.	%%	This operator is used to find the remainder of the first vector with the second vector. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a%%b)</pre> <p>It will give us the following</p> <p>output: [1] 2.0 3.3 0</p>
6.	%%/	This operator is used to find the division of the first vector with the second(quotient).	<pre>a <- c(2, 3.3, 4) b <- c(11, 5, 3) print(a%%/b)</pre> <p>It will give us the following</p> <p>output: [1] 0 0 1</p>

7.	^	This operator raised the first vector to the exponent of the second vector. a <- c(2, 3.3, 4)	<pre>b <- c(11, 5, 3) print(a^b)</pre> <p>It will give us the following output:</p> <pre>[1] 0248.0000 391.3539 4.0000</pre>	
----	---	---	--	--

Relational Operators

A relational operator is a symbol which defines some kind of relation between two entities. These include numerical equalities and inequalities. A relational operator compares each element of the

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

first vector with the corresponding element of the second vector. The result of the comparison will be a Boolean value. There are the following relational operators which are supported by R:

S. No				
Operator Description Example				
1.	>	This operator will return TRUE when every element in the first vector is greater than the corresponding element of the second vector.	<pre>a <- c(1, 3, 5) b <- c(2, 4, 6) print(a>b)</pre> <p>It will give us the following output: [1] FALSE FALSE FALSE</p>	
2.	<	This operator will return TRUE when every element in the first vector is less than the corresponding element of the second vector.	<pre>a <- c(1, 9, 5) b <- c(2, 4, 6) print(a<b)</pre> <p>It will give us the following output: [1] FALSE TRUE FALSE</p>	

3.	<=	This operator will return TRUE when every element in the first vector is less than or equal to the corresponding element of another vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a<=b)</pre> <p>It will give us the following</p> <p>output: [1] TRUE TRUE TRUE</p>
4.	>=	This operator will return TRUE when every element in the first vector is greater than or equal to the corresponding element of another vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a>=b)</pre> <p>It will give us the following</p> <p>output: [1] FALSE TRUE FALSE</p>
5.	==	This operator will return TRUE when every element in the first vector is equal to the corresponding element of the second vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a==b)</pre> <p>It will give us the following</p> <p>output: [1] FALSE TRUE FALSE</p>

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

6.	!=	This operator will return TRUE when every element in the first vector is not equal to the corresponding element of the second vector.	<pre>a <- c(1, 3, 5) b <- c(2, 3, 6) print(a!=b)</pre> <p>It will give us the following</p> <p>output: [1] TRUE FALSE TRUE</p>
----	----	---	--

Logical Operators

The logical operators allow a program to make a decision on the basis of multiple conditions. In the program, each operand is considered as a condition which can be evaluated to a false or true value. The value of the conditions is used to determine the overall value of the op1 **operator** op2. Logical operators are applicable to those vectors whose type is logical, numeric, or complex.

The logical operator compares each element of the first vector with the corresponding element of the second vector.

There are the following types of operators which are supported by R:

S.

Operator Description Example

No

1.	&	<p>This operator is known as the logical AND operator. This operator takes the both the vector and returns TRUE if both elements are TRUE.</p> <p>It will give us the following output:</p> <pre>a <- c(3, 0, TRUE, 2+2i) b <- c(2, 4, TRUE, 2+3i) print(a&b) [1] TRUE FALSE TRUE TRUE</pre>
2.		<p>This operator is called the logical OR operator. This operator takes the first element of the vector and returns TRUE if one of the elements is TRUE.</p> <p>It will give us the following output:</p> <pre>a <- c(3, 0, TRUE, 2+2i) b <- c(2, 4, TRUE, 2+3i) print(a b) [1] TRUE TRUE TRUE TRUE</pre>
3.	!	<p>This operator is known as the logical NOT operator. This operator takes the first element of the vector and gives the opposite logical value.</p> <p>It will give us the following output:</p> <pre>a <- c(3, 0, TRUE, 2+2i) print(!a) [1] FALSE TRUE FALSE FALSE</pre>

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

			5.		This operator takes the first element of both vectors and gives the result TRUE, if one of the elements is true.
4.	&&	This operator takes the first element of both vectors and gives TRUE as a result if both elements are TRUE.			

Assignment Operators

```
[1] FALSE TRUE FALSE
FALSE
```

```
a <- c(3, 0, TRUE, 2+2i)
b <- c(2, 4, TRUE, 2+3i)
print(a&&b)
```

It will give us the following output: [1] TRUE

```
a <- c(3, 0, TRUE, 2+2i)
b <- c(2, 4, TRUE, 2+3i)
print(allb)
```

It will give us the following output: [1] TRUE

An assignment operator is used to assign a new value to a variable. In R, these operators are used to assign values to vectors. There are the following types of assignment

S.			
No	Operator	Description	Example
1.	<- or = or <<-	These operators are known as left assignment operators.	<pre>a <- c(3, 0, TRUE, 2+2i) b <<- c(2, 4, TRUE, 2+3i) d = c(1, 2, TRUE, 2+3i) print(a) print(b) print(d)</pre> <p>It will give us the following output:</p> <pre>[1] 3+0i 0+0i 1+0i 2+2i [1] 2+0i 4+0i 1+0i 2+3i [1] 1+0i 2+0i 1+0i 2+3i</pre>
2.	-> or ->>	These operators are known as right assignment operators.	<pre>c(3, 0, TRUE, 2+2i) -> a c(2, 4, TRUE, 2+3i) ->> b print(a) print(b)</pre>

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

			<p>It will give us the following output:</p> <pre>[1] 3+0i 0+0i 1+0i 2+2i [1] 2+0i 4+0i 1+0i 2+3i</pre>
--	--	--	--

operators which are supported by R:

Miscellaneous Operators

Miscellaneous operators are used for a special and specific purpose. These operators are not used for general mathematical or logical computation. There are the following miscellaneous operators which are supported in R

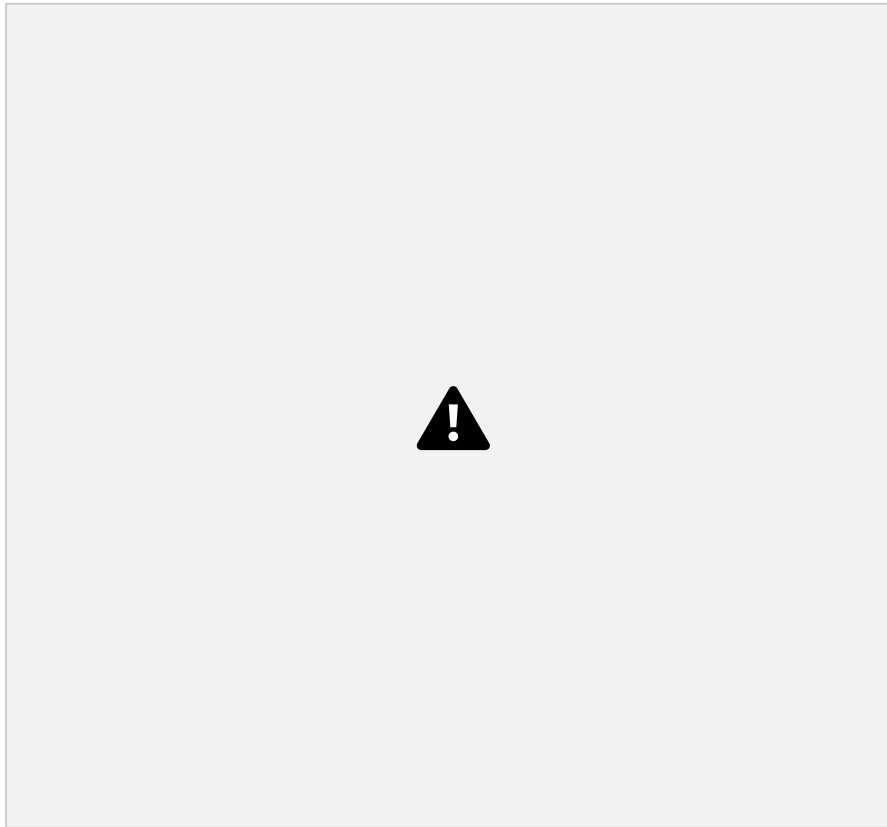
S. No Operator Description Example			
1.	:	The colon operator is used to create the series of numbers in sequence for a vector.	<pre>v <- 1:8 print(v)</pre> <p>It will give us the following output:</p> <pre>[1] 1 2 3 4 5 6 7 8</pre>
2.	%in%	This is used when we want to identify if an element belongs to a vector.	<pre>a1 <- 8 a2 <- 12 d <- 1:10 print(a1%in%d) print(a2%in%d)</pre> <p>It will give us the following output:</p> <pre>[1] FALSE [1] FALSE</pre>
3.	%*%	It is used to multiply a matrix with its transpose.	<pre>M=matrix(c(1,2,3,4,5,6), nrow=2, ncol=3, byrow=TRUE) T=m%*%T(m) print(T)</pre> <p>It will give us the following output:</p> <pre>14 32 32 77</pre>

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

R Vector

A **vector** is a basic data structure which plays an important role in R programming.

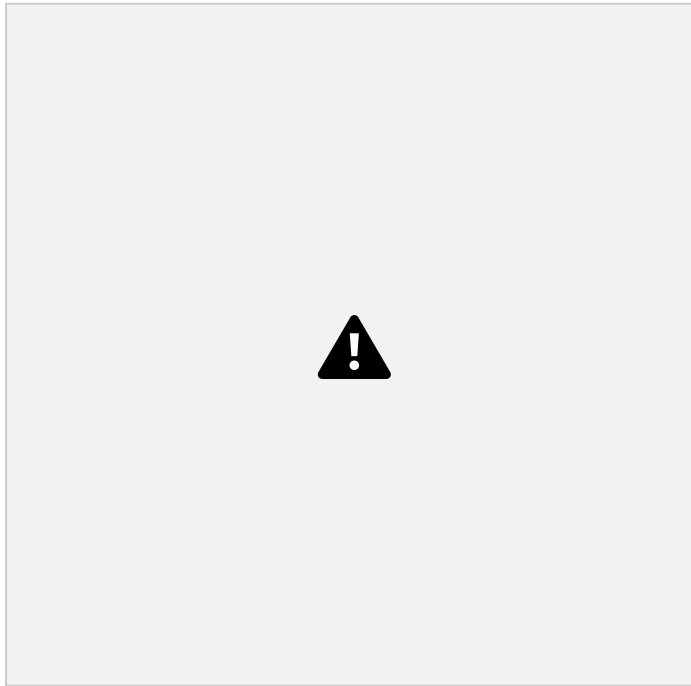
In R, a sequence of elements which share the same data type is known as vector. A vector supports logical, integer, double, character, complex, or raw data type. The elements which are contained in vector known as **components** of the vector. We can check the type of vector with the help of the **typeof()** function.



The length is an important property of a vector. A vector length is basically the number of elements in the vector, and it is calculated with the help of the **length()** function.

Vector is classified into two parts, i.e., **Atomic vectors** and **Lists**. They have three common properties, i.e., **function type**, **function length**, and **attribute function**.

Backward Skip 10sPlay VideoForward Skip 10s



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. In this section, we will discuss only the atomic vectors. We will discuss lists briefly in the next topic.

How to create a vector in R?

In R, we use `c()` function to create a vector. This function returns a one-dimensional array or simply vector. The `c()` function is a generic function which combines its argument. All arguments are restricted with a common data type which is the type of the returned value. There are various other ways to create a vector in R, which are as follows:

1) Using the colon(:) operator

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

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

1. `z<-x:y`

This operator creates a vector with elements from x to y and assigns it to z.

Example:

1. `a<-4:-10`

2. `a`

Output

```
[1] 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

2) Using the seq() function

In R, we can create a vector with the help of the seq() function. A sequence function creates a sequence of elements as a vector. The seq() function is used in two ways, i.e., by setting step size with 'by' parameter or specifying the length of the vector with the 'length.out' feature.

Example:

1. `seq_vec<-seq(1,4,by=0.5)`

2. `seq_vec`

3. `class(seq_vec)`

Output

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
```

Example:

1. `seq_vec<-seq(1,4,length.out=6)`

2. `seq_vec`

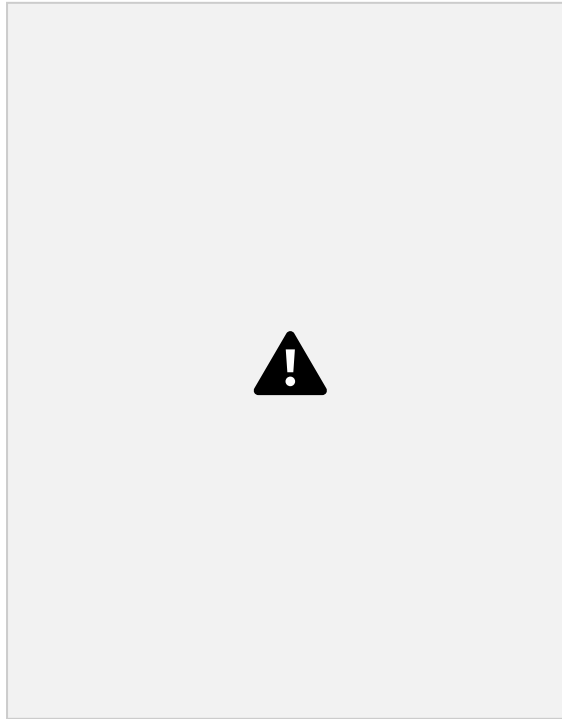
3. `class(seq_vec)`

Output

```
[1] 1.0 1.6 2.2 2.8 3.4 4.0  
[1] "numeric"
```

Atomic vectors in R

In R, there are four types of atomic vectors. Atomic vectors play an important role in Data Science. Atomic vectors are created with the help of `c()` function. These atomic vectors are as follows:



Numeric vector

The decimal values are known as numeric data types in R. If we assign a decimal value to any variable `d`, then this `d` variable will become a numeric type. A vector which contains numeric elements is known as a numeric vector.

Example:

1. `d<-45.5`
2. `num_vec<-c(10.1, 10.2, 33.2)`
3. `d`
4. `num_vec`
5. `class(d)`
6. `class(num_vec)`

Output

```
[1] 45.5
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

```
[1] 10.1 10.2 33.2  
[1] "numeric"  
[1] "numeric"
```

Integer vector

A non-fraction numeric value is known as integer data. This integer data is represented by "Int." The Int size is 2 bytes and long Int size of 4 bytes. There is two way to assign an integer value to a variable, i.e., by using as.integer() function and appending of L to the value.

A vector which contains integer elements is known as an integer vector.

Example:

1. `d<-as.integer(5)`
2. `e<-5L`
3. `int_vec<-c(1,2,3,4,5)`
4. `int_vec<-as.integer(int_vec)`
5. `int_vec1<-c(1L,2L,3L,4L,5L)`
6. `class(d)`
7. `class(e)`
8. `class(int_vec)`
9. `class(int_vec1)`

Output

```
[1] "integer"  
[1] "integer"  
[1] "integer"  
[1] "integer"
```

Character vector

A character is held as a one-byte integer in memory. In R, there are two different ways to create a character data type value, i.e., using as.character() function and by typing string between double quotes("") or single quotes(').

A vector which contains character elements is known as an integer vector.

Example:

1. `d<-'shubham'`

2. `e<- "Arpita"`

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

3. `f<-65`

4. `f<-as.character(f)`

5. `d`

6. `e`

7. `f`

8. `char_vec<-c(1,2,3,4,5)`

9. `char_vec<-as.character(char_vec)`

10. `char_vec1<-c("shubham","arpita","nishka","vaishali")`

11. `char_vec`

12. `class(d)`

13. `class(e)`

14. `class(f)`

15. `class(char_vec)`

16. `class(char_vec1)`

Output

```
[1] "shubham"
[1] "Arpita"
[1] "65"
[1] "1" "2" "3" "4" "5"
[1] "shubham" "arpita" "nishka" "vaishali"
[1] "character"
[1] "character"
[1] "character"
[1] "character"
[1] "character"
```

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:

1. `d<-as.integer(5)`

2. `e<-as.integer(6)`

3. `f<-as.integer(7)`

4. `g<-d>e`

5. `h<-e<f`

6. `g`

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

7. `h`

8. `log_vec<-c(d<e, d<f, e<d,e<f,f<d,f<e)`

9. `log_vec`

10. `class(g)`

11. `class(h)`

12. `class(log_vec)`

Output

```
[1] FALSE
[1] TRUE
[1] TRUE TRUE FALSE TRUE FALSE FALSE
[1] "logical"
[1] "logical"
[1] "logical"
```

Repeating Values:

You can create a vector by repeating the values using the `rep()` function. # Creates a vector with five 1s

Ex:

```
r<-rep(1, times=5)
```

```
r
```

Vector Function:

Vector Length

To find out how many items a vector has, use the `length()` function

Ex:

```
fruits <-c("banana", "apple", "orange")
```

```
length(fruits)
```

Output: `[1] 3`

Sort a Vector:

To sort items in a vector alphabetically or numerically, use the `sort()` function:

Ex:

```
fruits <-c("banana", "apple", "orange")
```

```
n<-c(5, 6, 1, 8)
```

```
sort(fruits)
```

```
sort(n)
```

Output:

```
[1] "apple" "banana" "orange"  
[1] 1 5 6 8
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA****Change an item:**

To change the value of a specific item, refer to the index number

Ex:

```
fruits <-c("banana", "apple", "orange")  
fruits[2]<-"Mango"  
fruits
```

Output:

```
[1] "banana" "Mango" "orange"
```

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. Indexing will be performed with the help of integer, character, or logic.

**1) Indexing with integer vector**

On integer vector, indexing is performed in the same way as we have applied in C, C++, and java.

There is only one difference, i.e., in C, C++, and java the indexing starts from 0, but in R, the indexing starts from 1. Like other programming languages, we perform indexing by specifying an integer value in square braces [] next to our vector.

Example:

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

1. `seq_vec<-seq(1,4,length.out=6)`
2. `seq_vec`
3. `seq_vec[2]`

Output

```
[1] 1.0 1.6 2.2 2.8 3.4 4.0  
[1] 1.6
```

2) Indexing with a character vector

In character vector indexing, we assign a unique key to each element of the vector. These keys are uniquely defined as each element and can be accessed very easily. Let's see an example to understand how it is performed.

Example:

1. `char_vec<-c("shubham"=22,"arpita"=23,"vaishali"=25)`
2. `char_vec`
3. `char_vec["arpita"]`

Output

```
shubham arpita vaishali  
22 23 25  
arpita  
23
```

3) Indexing with a logical vector

In logical indexing, it returns the values of those positions whose corresponding position has a logical vector TRUE. Let see an example to understand how it is performed on vectors.

Example:

1. `a<-c(1,2,3,4,5,6)`
2. `a[c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE)]`

Output

```
[1] 1 3 4 6
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

Vector Operation

In R, there are various operation which is performed on the vector. We can add, subtract, multiply or divide two or more vectors from each other. In data science, R plays an important role, and operations are required for data manipulation. There are the following types of operation which are performed on the vector.



1) Combining vectors

The `c()` function is not only used to create a vector, but also it is also used to combine two vectors. By combining one or more vectors, it forms a new vector which contains all the elements of each vector. Let see an example to see how `c()` function combines the vectors.

Example:

1. `p<-c(1,2,4,5,7,8)`
2. `q<-c("shubham","arpita","nishka","gunjan","vaishali","sumit")`
3. `r<-c(p,q)`

Output

```
[1] "1" "2" "4" "5" "7" "8"
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

```
[7] "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. We can add, subtract, multiply, or divide two vectors. Let see an example to understand how arithmetic operations are performed on vectors.

Example:

1. `a<-c(1,3,5,7)`
2. `b<-c(2,4,6,8)`
3. `a+b`
4. `a-b`
5. `a/b`
6. `a%%b`

Output

```
[1] 3 7 11 15
[1] -1 -1 -1 -1
[1] 2 12 30 56
[1] 0.5000000 0.7500000 0.8333333 0.8750000
[1] 1 3 5 7
```

3) Logical Index vector

With the help of the logical index vector in R, we can form a new vector from a given vector. This vector has the same length as the original vector. The vector members are TRUE only when the corresponding members of the original vector are included in the slice; otherwise, it will be false. Let see an example to understand how a new vector is formed with the help of logical index vector.

Example:

1. `a<-c("Shubham","Arpita","Nishka","Vaishali","Sumit","Gunjan")`
2. `b<-c(TRUE,FALSE,TRUE,TRUE,FALSE,FALSE)`
3. `a[b]`

Output

```
[1] "Shubham" "Nishka" "Vaishali"
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

4) Numeric Index

In R, we specify the index between square braces [] for indexing a numerical value. If our index is negative, it will return us all the values except for the index which we have specified. For example, specifying [-3] will prompt R to convert -3 into its absolute value and then search for the value which occupies that index.

Example:

1. `q<-c("shubham","arpita","nishka","gunjan","vaishali","sumit")`
2. `q[2]`
3. `q[-4]`
4. `q[15]`

Output

```
[1] "arpita"  
[1] "shubham" "arpita" "nishka" "vaishali" "sumit"  
[1] NA
```

5) Duplicate Index

An index vector allows duplicate values which means we can access one element twice in one operation. Let see an example to understand how duplicate index works.

Example:

1. `q<-c("shubham","arpita","nishka","gunjan","vaishali","sumit")`
2. `q[c(2,4,4,3)]`

Output

```
[1] "arpita" "gunjan" "gunjan" "nishka"
```

6) Range Indexes

Range index is used to slice our vector to form a new vector. For slicing, we used colon(:) operator. Range indexes are very helpful for the situation involving a large operator. Let see an example to understand how slicing is done with the help of the colon operator to form a new vector.

Example:

```
1. q<-c("shubham","arpita","nishka","gunjan","vaishali","sumit")
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

```
2. b<-q[2:5]
```

```
3. b
```

Output

```
[1] "arpita" "nishka" "gunjan" "vaishali"
```

7) Out-of-order Indexes

In R, the index vector can be out-of-order. Below is an example in which a vector slice with the order of first and second values reversed.

Example:

```
1. q<-c("shubham","arpita","nishka","gunjan","vaishali","sumit")b<-q[2:5]
```

```
2. q[c(2,1,3,4,5,6)]
```

Output

```
[1] "arpita" "shubham" "nishka" "gunjan" "vaishali" "sumit"
```

8) Named vectors members

We first create our vector of characters as:

```
1. z=c("TensorFlow","PyTorch")
```

```
2. z
```

Output

```
[1] "TensorFlow" "PyTorch"
```

Once our vector of characters is created, we name the first vector member as "Start" and the second member as "End" as:

1. `names(z)=c("Start","End")`
2. `z`

Output

```
Start End  
"TensorFlow" "PyTorch"
```

We retrieve the first member by its name as follows:

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

1. `z["Start"]`

Output

```
Start  
"TensorFlow"
```

We can reverse the order with the help of the character string index vector.

1. `z[c("Second","First")]`

Output

```
Second First  
"PyTorch" "TensorFlow"
```

Applications of vectors

1. In machine learning for principal component analysis vectors are used. They are extended to eigenvalues and eigenvector and then used for performing decomposition in vector spaces.
2. The inputs which are provided to the deep learning model are in the form of vectors. These vectors consist of standardized data which is supplied to the input layer of the neural network.
3. In the development of support vector machine algorithms, vectors are used.
4. Vector operations are utilized in neural networks for various operations like image recognition and text processing.

R Lists

In R, lists are the second type of vector. Lists are the objects of R which contain elements of different types such as number, vectors, string and another list inside it. It can also contain a function or a matrix as its elements. A list is a data structure which has components of mixed data types. We can say, a list is a generic vector which contains other objects.

Example

1. `vec <- c(3,4,5,6)`
2. `char_vec<-c("shubham","nishka","gunjan","sumit")`
3. `logic_vec<-c(TRUE,FALSE,FALSE,TRUE)`
4. `out_list<-list(vec,char_vec,logic_vec)`
5. `out_list`

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

Output:

```
[[1]]  
[1] 3 4 5 6  
[[2]]  
[1] "shubham" "nishka" "gunjan" "sumit"  
[[3]]  
[1] TRUE FALSE FALSE TRUE
```



List Functions:

R provides various functions for working with lists, including:

`length()`: Returns the number of elements in a list.

`names()`: Returns or sets the names of the elements in a list.

`str()`: Displays the structure of a list, showing its elements and data types.

`unlist()`: Converts a list to a vector by flattening it.

Lists creation

The process of creating a list is the same as a vector. In R, the vector is created with the help of `c()` function. Like `c()` function, there is another function, i.e., `list()` which is used to create a list in R. A list avoids the drawback of the vector which is data type. We can add the elements in the list of different data types.

Syntax

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA

1. `list()`

Example 1: Creating list with same data type

1. `list_1<-list(1,2,3)`
2. `list_2<-list("Shubham","Arpita","Vaishali")`
3. `list_3<-list(c(1,2,3))`
4. `list_4<-list(TRUE,FALSE,TRUE)`
5. `list_1`
6. `list_2`
7. `list_3`
8. `list_4`

Output:

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

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

```
[[1]]
```



```
[1] "Shubham"  
[[2]]  
[1] "Arpita"  
[[3]]  
[1] "Vaishali"
```

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

```
[[1]]  
[1] TRUE  
[[2]]  
[1] FALSE  
[[3]]  
[1] TRUE
```

Example 2: Creating the list with different data type

1. `list_data<-list("Shubham","Arpita",c(1,2,3,4,5),TRUE,FALSE,22.5,12L)`
2. `print(list_data)`

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

In the above example, the list function will create a list with character, logical, numeric, and vector element. It will give the following output

Output:

```
[[1]]  
[1] "Shubham"  
[[2]]  
[1] "Arpita"  
[[3]]  
[1] 1 2 3 4 5  
[[4]]  
[1] TRUE  
[[5]]  
[1] FALSE  
[[6]]  
[1] 22.5  
[[7]]  
[1] 12
```

Giving a name to list elements

R provides a very easy way for accessing elements, i.e., by giving the name to each element of a list. By assigning names to the elements, we can access the element easily. 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.

Let see an example to understand how we can give the names to the list elements.

Example

1. # Creating a list containing a vector, a matrix and a list.
2. list_data <- list(c("Shubham", "Nishka", "Gunjan"), matrix(c(40,80,60,70,90,80), nrow = 2),
3. list("BCA", "MCA", "B.tech"))
- 4.
5. # Giving names to the elements in the list.
6. names(list_data) <- c("Students", "Marks", "Course")
- 7.
8. # Show the list.
9. print(list_data)

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

Output:

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

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

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

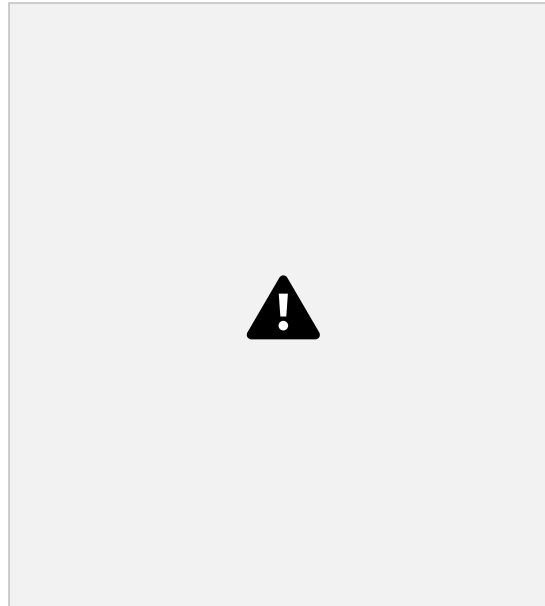
```
$Course[[2]]  
[1] "MCA"
```

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

Accessing List Elements

R provides two ways through which we can access the elements of a list. 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.; we cannot access the elements of a list using names if the list is normal.



Let see an example of both methods to understand how they are used in the list to access elements. **Example 1:** Accessing elements using index

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

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

2. `list_data <- list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80), nrow = 2),`
3. `list("BCA","MCA","B.tech"))`
4. # Accessing the first element of the list.
5. `print(list_data[1])`
- 6.
7. # Accessing the third element. The third element is also a list, so all its elements will be printed.
8. `print(list_data[3])`

Output:

```
[[1]]  
[1] "Shubham" "Arpita" "Nishka"
```

```
[[1]]  
[[1]][1]  
[1] "BCA"
```

```
[[1]][2]  
[1] "MCA"
```

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

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("BCA", "MCA", "B.tech"))`
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)`

Output:

```
$Student  
[1] "Shubham" "Arpita" "Nishka"
```

```
[,1] [,2] [,3]  
[1,] 40 60 90  
[2,] 80 70 80
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

```
$Student  
[1] "Shubham" "Arpita" "Nishka"
```

```
$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."
```

Manipulation of list elements

R allows us to add, delete, or update elements in the list. We can update an element of a list from anywhere, but elements can add or delete only at the end of the list. To remove an element from a specified index, we will assign it a null value. We can update the element of a list by overriding it

from the new value. Let see an example to understand how we can add, delete, or update the elements in the list.

ADVERTISEMENT

Example

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),`
3. `list("BCA", "MCA", "B.tech"))`
- 4.
5. # Giving names to the elements in the list.
6. `names(list_data) <- c("Student", "Marks", "Course")`
- 7.
8. # Adding element at the end of the list.
9. `list_data[4] <- "Moradabad"`
10. `print(list_data[4])`
- 11.
12. # Removing the last element.
13. `list_data[4] <- NULL`
- 14.
15. # Printing the 4th Element.
16. `print(list_data[4])`

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

- 17.
18. # Updating the 3rd Element.
19. `list_data[3] <- "Masters of computer applications"`
20. `print(list_data[3])`

Output:

```
[[1]]  
[1] "Moradabad"
```

```
$<NA>  
NULL
```

```
$Course  
[1] "Masters of computer applications"
```

Check if Item Exists:

To find out if a specified item is present in a list, use `TRUE` the `%in%` function

Ex:

```
lst<-list("apple", "banana", "cherry")  
"apple" %in% lst
```

Output: [1] TRUE

Remove List Items:

You can also remove list items. The following example creates a new, updated list without an "apple" item:

Ex:

```
lst<-list("apple", "banana", "cherry")  
nl<-lst[-1]  
nl
```

Output:

```
[[1]]  
[1] "banana"  
[[2]]  
[1]"cherry"
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

Converting list to vector

There is a drawback with the list, i.e., we cannot perform all the arithmetic operations on list elements. To remove this, drawback R provides `unlist()` function. This function converts the list into vectors. In some cases, it is required to convert a list into a vector so that we can use the elements of the vector for further manipulation.

The `unlist()` function takes the list as a parameter and change into a vector. Let see an example to understand how to `unlist()` function is used in R.

Example

1. # Creating lists.

```

2. list1 <- list(10:20)
3. print(list1)
4.
5. list2 <-list(5:14)
6. print(list2)
7.
8. # Converting the lists to vectors.
9. v1 <- unlist(list1)
10. v2 <- unlist(list2)
11.
12. print(v1)
13. print(v2)
14.
15. adding the vectors
16. result <- v1+v2
17. print(result)

```

Output:

```

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

[[1]]
[1] 10 11 12 13 14

[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 19

```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

Merging Lists

R allows us to merge one or more lists into one list. Merging is done with the help of the `list()` function also. 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. Let see an example to understand how the merging process is done.

Example

```

1. # Creating two lists.
2. Even_list <- list(2,4,6,8,10)

```

```

3. Odd_list <- list(1,3,5,7,9)
4.
5. # Merging the two lists.
6. merged.list <- list(Even_list,Odd_list)
7.
8. # Printing the merged list.
9. print(merged.list)

```

Output:

```

[[1]]
[[1]][[1]]
[1] 2

```

```

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

```

```

[[1]][[3]]
[1] 6

```

```

[[1]][[4]]
[1] 8

```

```

[[1]][[5]]
[1] 10

```

```

[[2]]
[[2]][[1]]
[1] 1

```

```

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

```

```

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

```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

```

[[2]][[4]]
[1] 7

```

```

[[2]][[5]]
[1] 9

```

R 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.

R Array Syntax

There is the following syntax of R arrays:

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

data

The data is the first argument in the array() function. It is an input vector which is given to the array.

matrices

In R, the array consists of multi-dimensional matrices.

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, columns, layers and blocks.

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA



How to create?

In R, array creation is quite simple. We can easily create an array using vector and array() function. In array, data is stored in the form of the matrix. 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.

Let see an example to understand how we can implement an array with the help of the vectors and array() function.

Example

1. #Creating two vectors of different lengths
2. `vec1 <-c(1,3,5)`
3. `vec2 <-c(10,11,12,13,14,15)`
- 4.
5. #Taking these vectors as input to the array
6. `res <- array(c(vec1,vec2),dim=c(3,3,2))`
7. `print(res)`

Output

```
,, 1
[,1] [,2] [,3]
[1,] 1 10 13
[2,] 3 11 14
```

R PROGRAMMING V - SEM BCA

```
[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.

It is not necessary to give the name to the rows and columns. It is only used to differentiate the row and column for better understanding.

Below is an example, in which we create two arrays and giving names to the rows, columns, and matrices.

Example

1. #Creating two vectors of different lengths
2. `vec1 <-c(1,3,5)`
3. `vec2 <-c(10,11,12,13,14,15)`
- 4.
5. #Initializing names for rows, columns and matrices
6. `col_names <- c("Col1","Col2","Col3")`
7. `row_names <- c("Row1","Row2","Row3")`
8. `matrix_names <- c("Matrix1","Matrix2")`
- 9.
10. #Taking the vectors as input to the array
11. `res <- array(c(vec1,vec2),dim=c(3,3,2),dimnames=list(row_names,col_names,matrix_names))`
12. `print(res)`

Output

```
, , Matrix1  
  
Col1 Col2 Col3  
Row1 1 10 13  
Row2 3 11 14  
Row3 5 12 15  
  
, , Matrix2
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

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

Accessing array elements

Like C or C++, we can access the elements of the array. The elements are accessed with the help of the index. Simply, we can access the elements of the array with the help of the indexing method. Let see an example to understand how we can access the elements of the array using the indexing method.

Example

1. , , Matrix1
2. Col1 Col2 Col3
3. Row1 1 10 13
4. Row2 3 11 14
5. Row3 5 12 15
- 6.
7. , , Matrix2
8. Col1 Col2 Col3
9. Row1 1 10 13
10. Row2 3 11 14
11. Row3 5 12 15
- 12.
13. Col1 Col2 Col3
14. 5 12 15
- 15.
16. [1] 13
- 17.
18. Col1 Col2 Col3
19. Row1 1 10 13
20. Row2 3 11 14
21. Row3 5 12 15

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.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

Example

```
1. #Creating two vectors of different lengths
2. vec1 <-c(1,3,5)
3. vec2 <-c(10,11,12,13,14,15)
4.
5. #Taking the vectors as input to the array1
6. res1 <- array(c(vec1,vec2),dim=c(3,3,2))
7. print(res1)
8.
9. #Creating two vectors of different lengths
10. vec1 <-c(8,4,7)
11. vec2 <-c(16,73,48,46,36,73)
12.
13. #Taking the vectors as input to the array2
14. res2 <- array(c(vec1,vec2),dim=c(3,3,2))
15. print(res2)
16.
17. #Creating matrices from these arrays
18. mat1 <- res1[,2]
19. mat2 <- res2[,2]
20. res3 <- mat1+mat2
21. print(res3)
```

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
```

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

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

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

```
, , 2
[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
```

Calculations across array elements

For calculation purpose, R provides **apply()** function. This apply function contains three parameters i.e., x, margin, and function.

This function takes the array on which we have to perform the calculations. The basic syntax of the **apply()** function is as follows:

1. **apply(x, margin, fun)**

Here, x is an array, and a margin is the name of the dataset which is used and fun is the function which is to be applied to the elements of the array.

Example

1. #Creating two vectors of different lengths
2. `vec1 <-c(1,3,5)`
3. `vec2 <-c(10,11,12,13,14,15)`
- 4.
5. #Taking the vectors as input to the array1
6. `res1 <- array(c(vec1,vec2),dim=c(3,3,2))`
7. `print(res1)`

- 8.
9. #using apply function
10. result <- apply(res1,c(1),sum)
11. print(result)

Output

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

```
., 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
```

```
[1] 48 56 64
```

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. On R matrices, we can perform addition, subtraction, multiplication, and division operation.

In the R matrix, elements are arranged in a fixed number of rows and columns. The matrix elements are the real numbers. In R, we use matrix function, which can easily reproduce the memory representation of the matrix. In the R matrix, all the elements must share a common basic type.

Example

1. matrix1<-matrix(c(11, 13, 15, 12, 14, 16),nrow =2, ncol =3, byrow = TRUE)
2. matrix1

Output

```
[1,] [2,] [3,]
[1,] 11 13 15
[2,] 12 14 16
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**



History of matrices in R

The word "Matrix" is the Latin word for womb which means a place where something is formed or produced. Two authors of historical importance have used the word "Matrix" for unusual ways. They proposed this axiom as a means to reduce any function to one of the lower types so that at the "bottom" (0order) the function is identical to its extension.

Any possible function other than a matrix from the matrix holds true with the help of the process of generalization. It will be true only when the proposition (which asserts function in question) is true. It will hold true for all or one of the value of argument only when the other argument is undetermined.

How to create a matrix in R?

Like vector and list, R provides a function which creates a matrix. R provides the `matrix()` function to create a matrix. This function plays an important role in data analysis. There is the following syntax of the matrix in R:

1. `matrix(data, nrow, ncol, byrow, dim_name)`

data

The first argument in matrix function is data. It is the input vector which is the data elements of the matrix.

nrow

The second argument is the number of rows which we want to create in the matrix.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

ncol

The third argument 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

The dim_name parameter is the name assigned to the rows and columns.

Let's see an example to understand how matrix function is used to create a matrix and arrange the elements sequentially by row or column.

Example

1. #Arranging elements sequentially by row.
2. `P <- matrix(c(5:16), nrow = 4, byrow = TRUE)`
3. `print(P)`
- 4.
5. # Arranging elements sequentially by column.
6. `Q <- matrix(c(3:14), nrow = 4, byrow = FALSE)`
7. `print(Q)`

8.

9. # Defining the column and row names.

10. `row_names = c("row1", "row2", "row3", "row4")`

11. `col_names = c("col1", "col2", "col3")`

12.

13. `R <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))`

14. `print(R)`

Output

```
[,1] [,2] [,3]
[1,] 5 6 7
[2,] 8 9 10
[3,] 11 12 13
[4,] 14 15 16
```

```
[,1] [,2] [,3]
[1,] 3 7 11
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

```
[2,] 4 8 12
[3,] 5 9 13
[4,] 6 10 14
```

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

Accessing matrix elements in R

Like C and C++, we can easily access the elements of our matrix by using the index of the element. 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.

Let see an example to understand how elements are accessed from the matrix present on nth row mth column, nth row, or mth column.

Example

1. # Defining the column and row names.

```

2. row_names = c("row1", "row2", "row3", "row4")
3. col_names = c("col1", "col2", "col3")
4. #Creating matrix
5. R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
6. print(R)
7.
8. #Accessing element present on 3rd row and 2nd column
9. print(R[3,2])
10.
11. #Accessing element present in 3rd row
12. print(R[3,])
13.
14. #Accessing element present in 2nd column
15. print(R[,2])

```

Output

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

```

col1 col2 col3
row1 5 6 7
row2 8 9 10
row3 11 12 13
row4 14 15 16

```

```
[1] 12
```

```

col1 col2 col3
11 12 13

```

```

row1 row2 row3 row4
6 9 12 15

```

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. By assigning a new value to that position, the old value will get replaced with the new one. This modification technique is quite simple to perform matrix modification. The basic syntax for it is as follows:

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

1. `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.

Let see an example to understand how modification will be done:

Example

1. # Defining the column and row names.
2. `row_names = c("row1", "row2", "row3", "row4")`
3. `col_names = c("col1", "col2", "col3")`
- 4.

5. `R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))`
6. `print(R)`
- 7.
8. #Assigning value 20 to the element at 3d row and 2nd column
9. `R[3,2]<-20`
10. `print(R)`

Output

```
col1 col2 col3
row1 5 6 7
row2 8 9 10
row3 11 12 13
row4 14 15 16
```

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

Use of Relational Operator

R provides another way to perform matrix modification. In this method, we used some relational operators like `>`, `<`, `==`. Like the first method, the second method is quite simple to use. Let see an example to understand how this method modifies the matrix.

Example 1

1. # Defining the column and row names.

SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA

2. `row_names = c("row1", "row2", "row3", "row4")`
3. `col_names = c("col1", "col2", "col3")`
- 4.
5. `R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))`
6. `print(R)`
- 7.
8. #Replacing element that equal to the 12
9. `R[R==12]<-0`
10. `print(R)`

Output

```
col1 col2 col3
row1 5 6 7
row2 8 9 10
row3 11 12 13
row4 14 15 16
```

```
col1 col2 col3
row1 5 6 7
row2 8 9 10
row3 11 0 13
row4 14 15 16
```

Example 2

1. # Defining the column and row names.
2. `row_names = c("row1", "row2", "row3", "row4")`
3. `col_names = c("col1", "col2", "col3")`
- 4.
5. `R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))`
6. `print(R)`
- 7.
8. #Replacing elements whose values are greater than 12
9. `R[R>12]<-0`
10. `print(R)`

Output

```
col1 col2 col3
row1 5 6 7
row2 8 9 10
```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

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

```
col1 col2 col3
row1 5 6 7
row2 8 9 10
row3 11 12 0
row4 0 0 0
```

Addition of Rows and Columns

The third method of matrix modification is through the addition of rows and columns using the `cbind()` and `rbind()` function. The `cbind()` and `rbind()` function are used to add a column and a row respectively. Let see an example to understand the working of `cbind()` and `rbind()` functions.

Example 1

1. # Defining the column and row names.
2. `row_names = c("row1", "row2", "row3", "row4")`
3. `col_names = c("col1", "col2", "col3")`
- 4.
5. `R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))`
6. `print(R)`
- 7.
8. # Adding row
9. `rbind(R, c(17, 18, 19))`
- 10.
11. # Adding column
12. `cbind(R, c(17, 18, 19, 20))`
- 13.
14. # transpose of the matrix using the `t()` function:
15. `t(R)`
- 16.
17. # Modifying the dimension of the matrix using the `dim()` function
18. `dim(R) <- c(1, 12)`
19. `print(R)`

Output

```
col1 col2 col3
row1 5 6 7
```

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

```
row2 8 9 10
row3 11 12 13
row4 14 15 16
```

```
col1 col2 col3
row1 5 6 7
row2 8 9 10
row3 11 12 13
row4 14 15 16
```

```
17 18 19
```

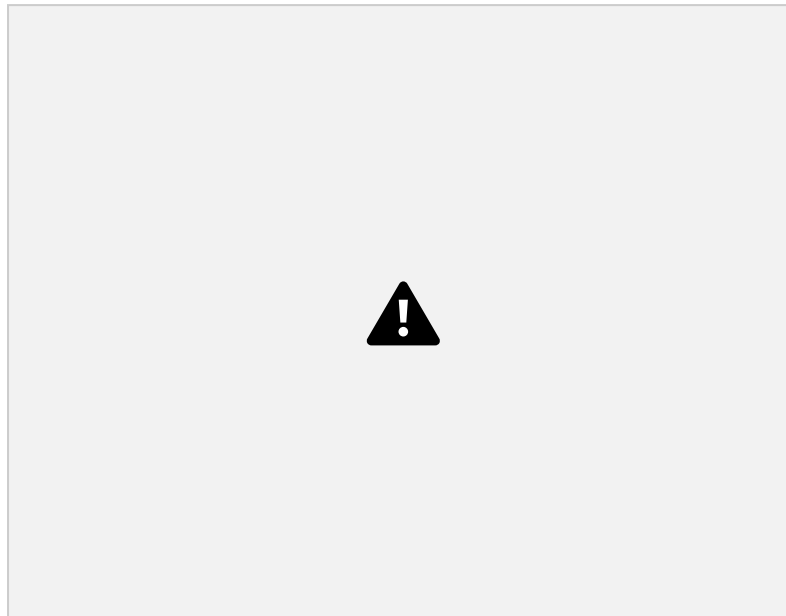
```
col1 col2 col3  
row1 5 6 7 17  
row2 8 9 10 18  
row3 11 12 13 19  
row4 14 15 16 20
```

```
row1 row2 row3 row4  
col1 5 8 11 14  
col2 6 9 12 15  
col3 7 10 13 16
```

```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]  
[1,] 5 8 11 14 6 9 12 15 7 10 13 16
```

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.



**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

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.


```

4. #Addition
5. sum<-R+S
6. print(sum)
7.
8. #Subtraction
9. sub<-R-S
10. print(sub)
11.
12. #Multiplication
13. mul<-R*S
14. print(mul)
15.
16. #Multiplication by constant
17. mul1<-R*12
18. print(mul1)
19.
20. #Division
21. div<-R/S
22. 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

```

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

```

[,1] [,2] [,3]
[1,] 5 45 117
[2,] 12 60 140
[3,] 21 77 165
[4,] 32 96 192

```

```

[,1] [,2] [,3]
[1,] 60 108 156

```

```
[2,] 72 120 168  
[3,] 84 132 180  
[4,] 96 144 192
```

```
[,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
```

Matrix Transposition:

Transpose of a matrix using the t() function

```
result<-t(matrix1)
```

Matrix Inversion:

To find the inverse of a square matrix, use the solve() function

```
inverse<-solve(matrix1)
```

Applications of matrix

1. In geology, Matrices takes surveys and plot graphs, statistics, and used to study in different fields.
2. Matrix is the representation method which helps in plotting common survey things.
3. In robotics and automation, Matrices have the topmost elements for the robot movements.
4. Matrices are mainly used in calculating the gross domestic products in Economics, and it also helps in calculating the capability of goods and products.
5. In computer-based application, matrices play a crucial role in the creation of realistic seeming motion.

R 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.

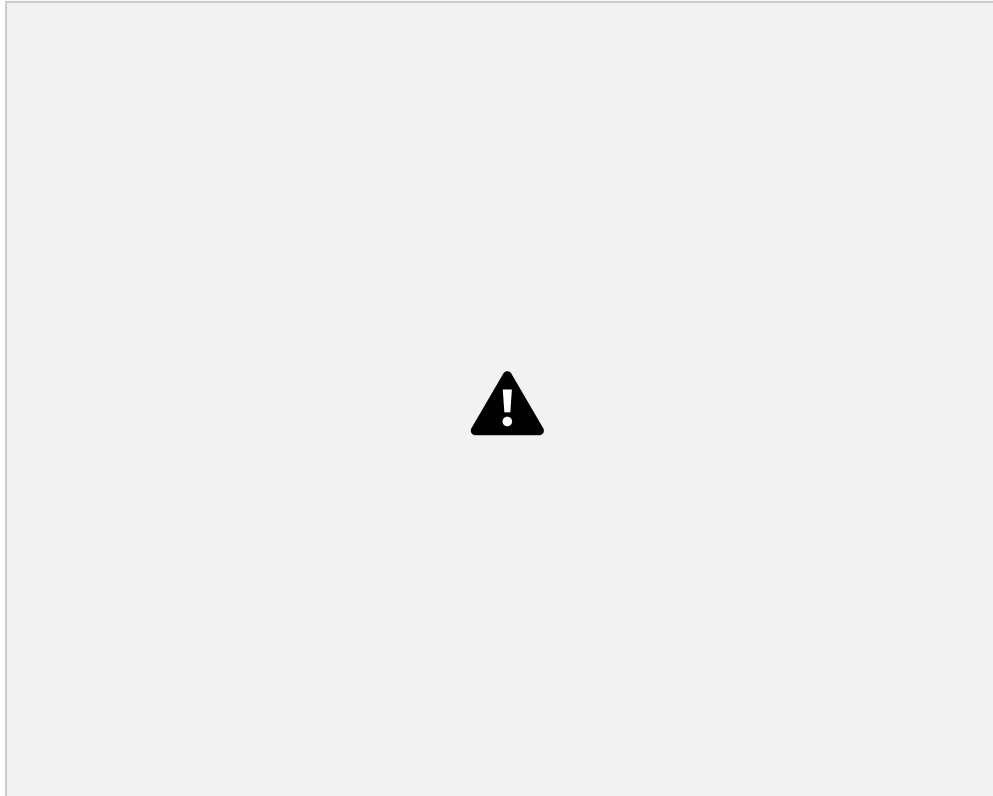
A data frame is used to store data table and the vectors which are present in the form of a list in a data frame, are of equal length.

SHREE MEDHA DEGREE COLLEGE MANJESH M R PROGRAMMING V - SEM BCA

In a simple way, it is a list of equal length vectors. 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.

- **Rectangular structure:** Data frames are two dimensional structures with rows and columns forming a rectangular shape. All columns must have same number of rows, making them suitable for structured datasets.
- Column Names: 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. In below example, we create a data frame that contains employee id (integer vector), employee name(character vector), salary(numeric vector), and starting date(Date vector).

Example

**SHREE MEDHA DEGREE COLLEGE MANJESH M
R PROGRAMMING V - SEM BCA**

1. # Creating the data frame.
2. `emp.data<- data.frame(`

```

3. employee_id = c (1:5),
4. employee_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,915.2,611.0,729.0,843.25),
6.
7. starting_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. # Printing the data frame.
12. 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

```

Getting the structure of R Data Frame

In R, we can find the structure of our data frame. R provides an in-build function called `str()` which returns the data with its complete structure. In below example, we have created a frame using a vector of different data type and extracted the structure of it.

Example

```

1. # Creating the data frame.
2. emp.data<- data.frame(
3. employee_id = c (1:5),
4. employee_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
6.
7. starting_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),

```