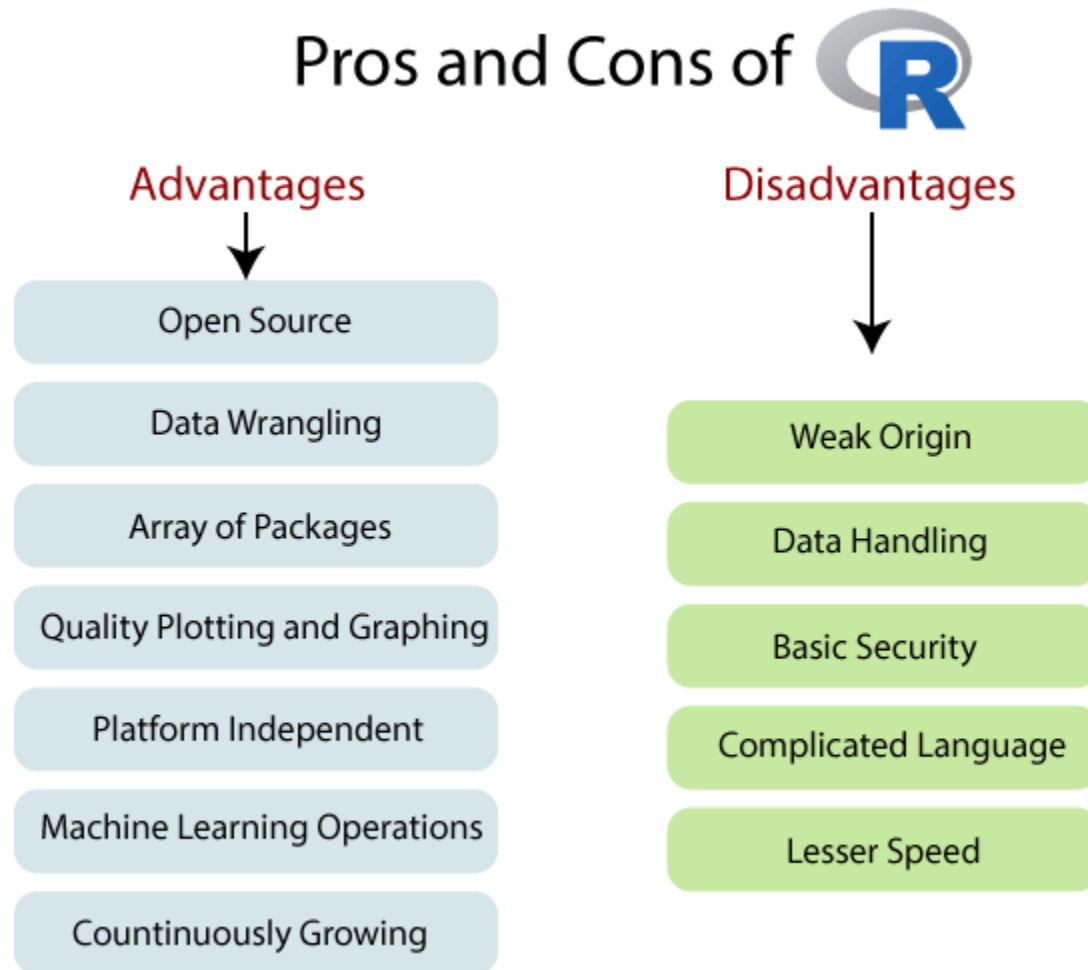


R Programming Language

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



Pros

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 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.

Cons

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.

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.

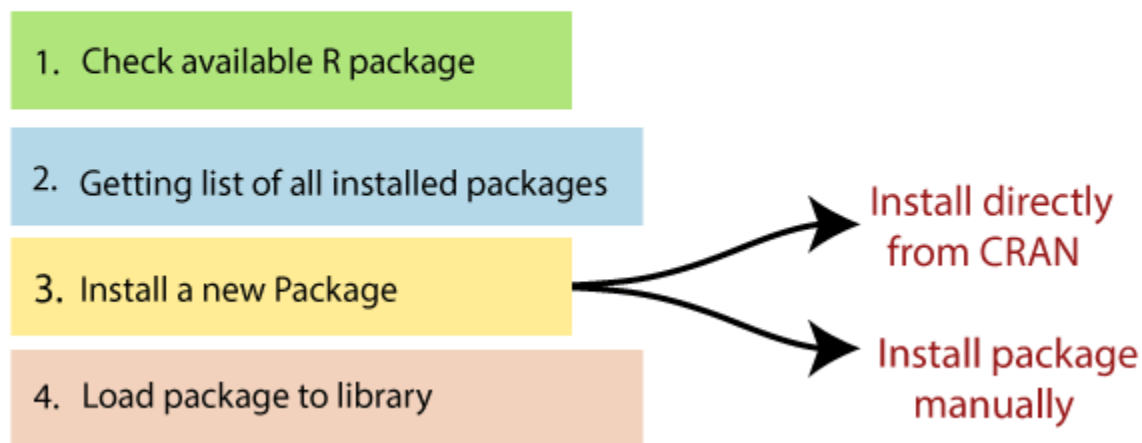
In R, algorithms are spread across different packages. The programmers who have no prior knowledge of packages may find it difficult to implement algorithms.

R Packages

R packages are the collection of R functions, sample data, and compile codes. In the R environment, these packages are stored under a directory called "**library**." During installation, R installs a set of packages. We can add packages later when they are

needed for some specific purpose. Only the default packages will be available when we start the R console. Other packages which are already installed will be loaded explicitly to be used by the R program.

There is the following list of commands to be used to check, verify, and use the R packages.



Check Available R Packages

To check the available R Packages, we have to find the library location in which R packages are contained. R provides `libPaths()` function to find the library locations.

1. `libPaths()`

When the above code executes, it produces the following project, which may vary depending on the local settings of our PCs & Laptops.

```
[1] "C:/Users/ajeet/OneDrive/Documents/R/win-library/3.6"  
[2] "C:/Program Files/R/R-3.6.1/library"
```

Getting the list of all the packages installed

R provides `library()` function, which allows us to get the list of all the installed packages.

1. library()

When we execute the above function, it produces the following result, which may vary depending on the local settings of our PCs or laptops.

Packages in library 'C:/Program Files/R/R-3.6.1/library':

Like library() function, R provides search() function to get all packages currently loaded in the R environment.

1. search()

When we execute the above code, it will produce the following result, which may vary depending on the local settings of our PCs and laptops:

```
[1] ".GlobalEnv"      "package:stats"    "package:graphics"
[4] "package:grDevices" "package:utils"    "package:datasets"
[7] "package:methods"  "Autoloads"        "package:base"
```

Install a New Package

In R, there are two techniques to add new R packages. The first technique is installing package directly from the CRAN directory, and the second one is to install it manually after downloading the package to our local system.

Install directly from CRAN

The following command is used to get the packages directly from CRAN webpage and install the package in the R environment. We may be prompted to choose the nearest mirror. Choose the one appropriate to our location.

1. install.packages("Package Name")

The syntax of installing XML package is as follows:

1. install.packages("XML")

Output

```
Rterm (64-bit)

> install.packages("XML")
Installing package into 'C:/Users/ajeet/OneDrive/Documents/R/win-library/3.6'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cloud.r-project.org/bin/windows/contrib/3.6/XML_3.98-1.20.zip'
Content type 'application/zip' length 4610084 bytes (4.4 MB)
downloaded 4.4 MB

package 'XML' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
      C:\Users\ajeet\AppData\Local\Temp\Rtmp0qpzbj\downloaded_packages
>
```

Install package manually

To install a package manually, we first have to download it from https://cran.r-project.org/web/packages/available_packages_by_name.html. The required package will be saved as a .zip file in a suitable location in the local system.

Once the downloading has finished, we will use the following command:

1. `install.packages(file_name_with_path, repos = NULL, type = "source")`

Install the package named "XML"

1. `install.packages("C:\\Users\\ajeet\\OneDrive\\Desktop\\graphics\\xml2_1.2.2.zip", repos = NULL, type = "source")`

Load Package to Library

We cannot use the package in our code until it will not be loaded into the current R environment. We also need to load a package which is already installed previously but not available in the current environment.

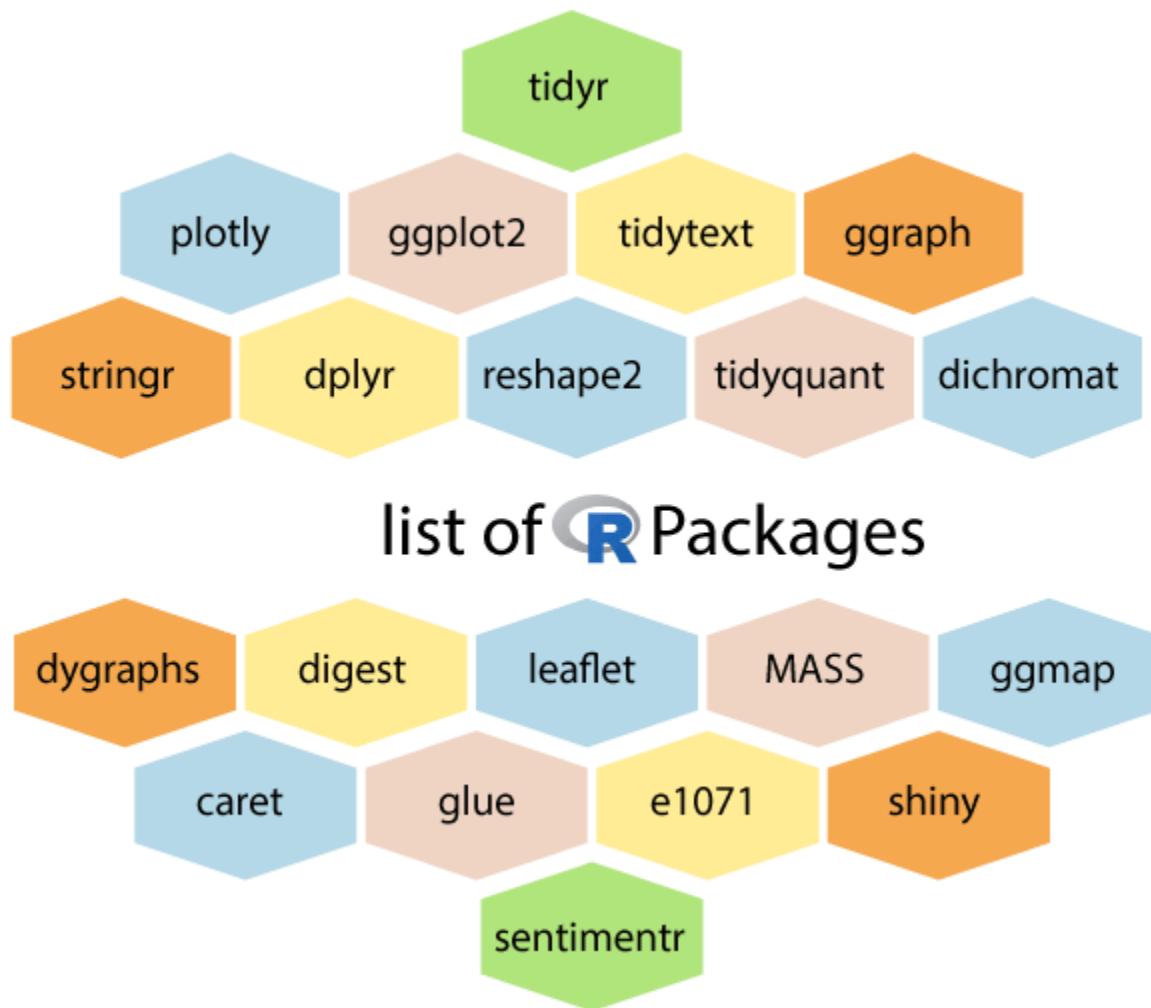
There is the following command to load a package:

1. `library("package Name", lib.loc = "path to library")`

List of R packages

R is the language of data science which includes a vast repository of packages. These packages appeal to different regions which use R for their data purposes. CRAN has 10,000 packages, making it an ocean of superlative statistical work. There are lots of packages in R, but we will discuss the important one.

There are some mostly used and popular packages which are as follows:



1) tidy

The word tidy comes from the word tidy, which means clear. So the **tidy** package is used to make the data 'tidy'. This package works well with dplyr. This package is an evolution of the reshape2 package.

2) ggplot2

R allows us to create graphics declaratively. R provides the **ggplot** package for this purpose. This package is famous for its elegant and quality graphs which sets it apart from other visualization packages.

3) ggraph

R provides an extension of ggplot known as **ggraph**. The limitation of **ggplot** is the dependency on tabular data is taken away in ggraph.

4) dplyr

R allows us to perform data wrangling and data analysis. R provides the **dplyr** library for this purpose. This library facilitates several functions for the data frame in R.

5) tidyquant

The tidyquant is a financial package which is used for carrying out quantitative financial analysis. This package adds to the **tidyverse** universe as a financial package which is used for importing, analyzing and visualizing the data.

6) dygraphs

The dygraphs package provides an interface to the main JavaScript library which we can use for charting. This package is essentially used for plotting time-series data in R.

7) leaflet

For creating interactive visualization, R provides the **leaflet** package. This package is an open-source JavaScript library. The world's popular websites like the New York Times, Github and Flickr, etc. are using leaflet. The leaflet package makes it easier to interact with these sites.

8) ggmap

For delineating spatial visualization, the **ggmap** package is used. It is a mapping package which consists of various tools for geolocating and routing.

9) glue

R provides the **glue** package to perform the operations of data wrangling. This package is used for evaluating R expressions which are present within the string.

10) shiny

R allows us to develop interactive and aesthetically pleasing web apps by providing a **shiny** package. This package provides various extensions with HTML widgets, CSS, and JavaScript.

11) plotly

The plotly package provides online interactive and quality graphs. This package extends upon the JavaScript library **-plotly.js**.

12) tidytext

The **tidytext** package provides various functions of text mining for word processing and carrying out analysis through ggplot, dplyr, and other miscellaneous tools.

13) stringr

The stringr package provides simplicity and consistency to use wrappers for the '**stringi**' package. The stringi package facilitates common string operations.

14) reshape2

This package facilitates flexible reorganization and aggregation of data using melt () and dcast () functions.

15) dichromat

The R dichromat package is used to remove Red-Green or Blue-Green contrasts from the colors.

16) digest

The digest package is used for the creation of cryptographic hash objects of R functions.

17) MASS

The **MASS** package provides a large number of statistical functions. It provides datasets that are in conjunction with the book "Modern Applied Statistics with S."

18) caret

R allows us to perform classification and regression tasks by providing the caret package. **CaretEnsemble** is a feature of caret which is used for the combination of different models.

19) e1071

The **e1071** library provides useful functions which are essential for data analysis like Naive Bayes, Fourier Transforms, SVMs, Clustering, and other miscellaneous functions.

20) sentimentr

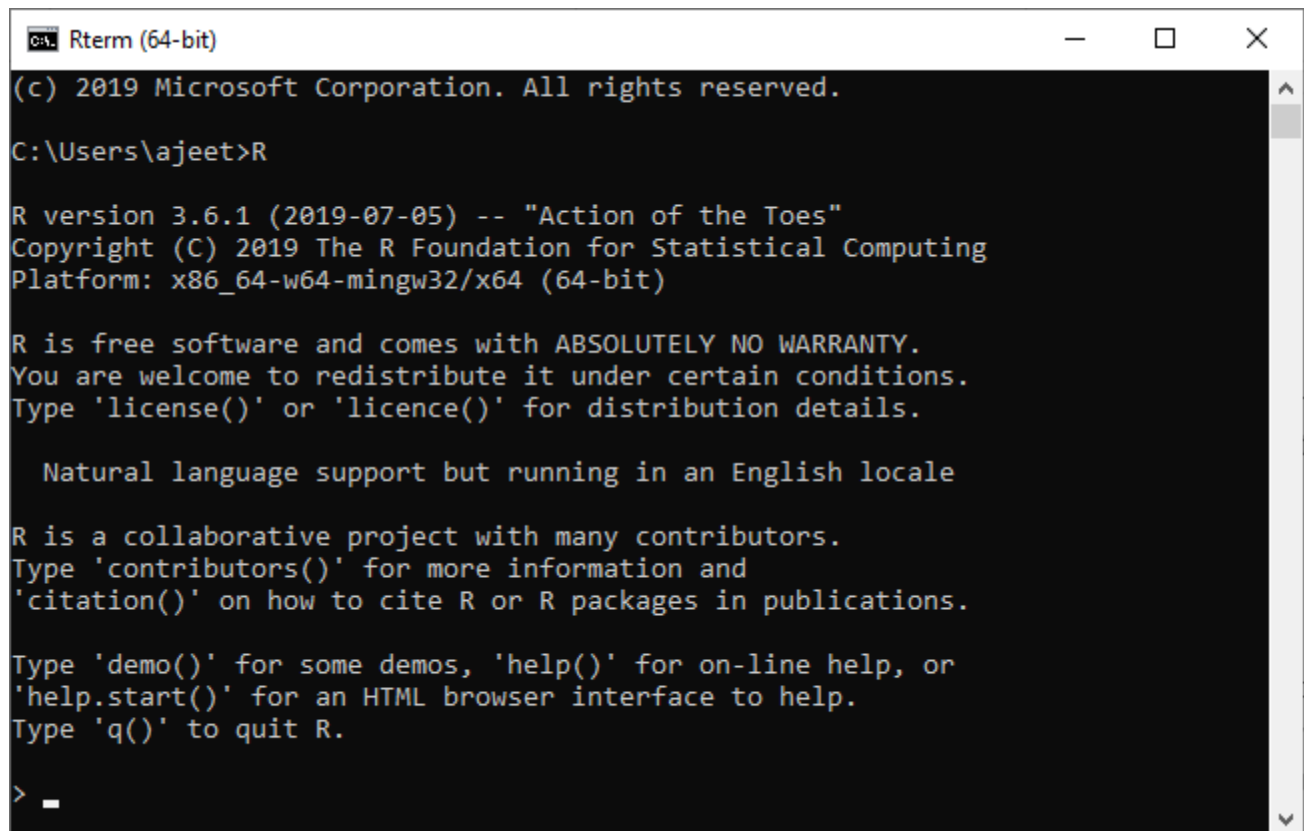
The sentiment package provides functions for carrying out sentiment analysis. It is used to calculate text polarity at the sentence level and to perform aggregation by rows or grouping variables.

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.



```
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeeet>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:



```
> 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.

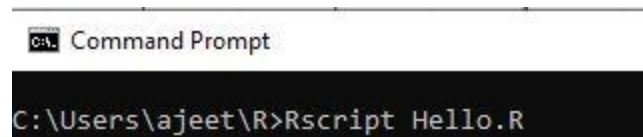
R Script File

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

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.



```
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.

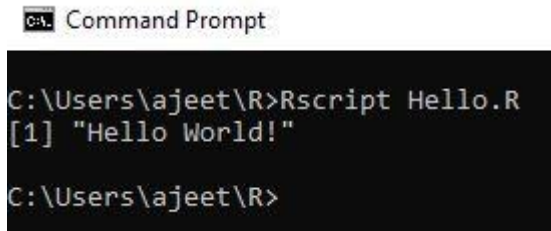
Single-line comment

1. `#My First program in R programming`
2. `string <-"Hello World!"`
3. `print(string)`

The trick for multi-line comment

1. `#Trick for multi-line comment`
2. `if(FALSE) {`
3. `"R is an interpreted computer programming language which was created by`
4. `Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand "`
5. `}`

6. #My First program in R programming
7. string <-"Hello World!"
8. **print**(string)



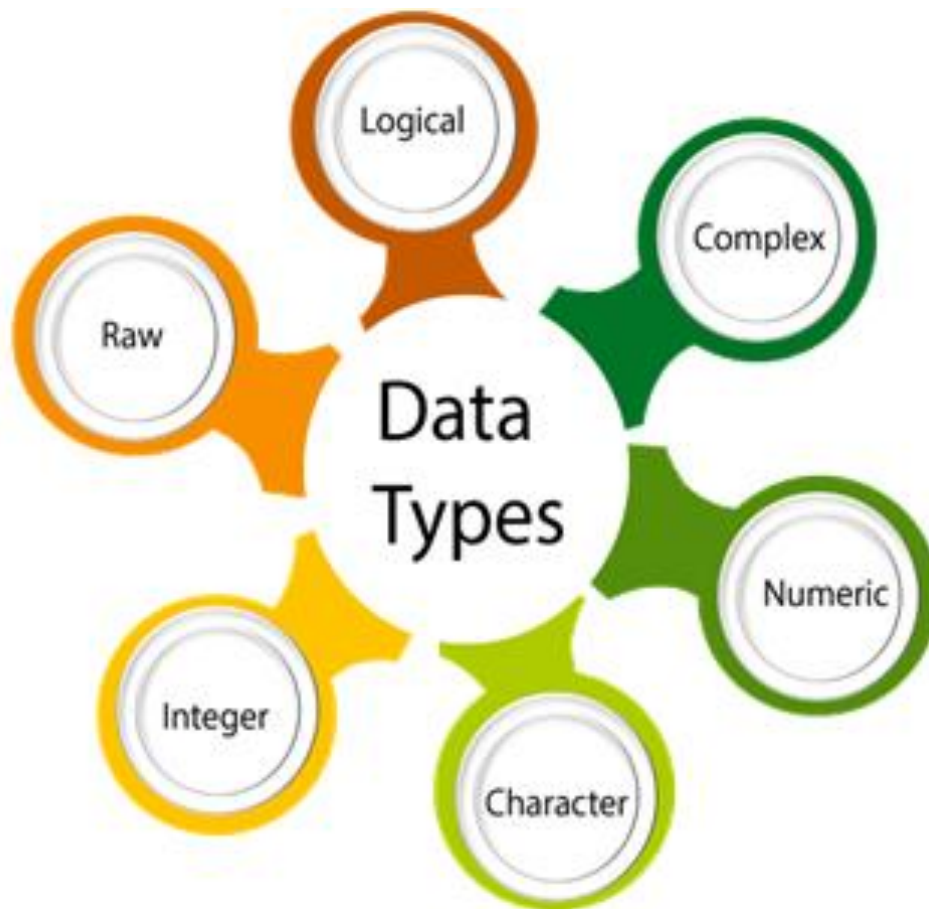
```
C:\> Command Prompt
C:\Users\ajeet\R>Rscript Hello.R
[1] "Hello World!"
C:\Users\ajeet\R>
```

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.
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 <code>as.character()</code> function.
Raw		A raw data type is used to holds raw bytes.

Let's see an example for better understanding of data types:

```
1. #Logical Data type
2. variable_logical<- TRUE
3. cat(variable_logical,"\n")
4. cat("The data type of variable_logical is ",class(variable_logical),"\n\n")
5. #Numeric Data type
6. variable_numeric<- 3532
7. cat(variable_numeric,"\n")
8. cat("The data type of variable_numeric is ",class(variable_numeric),"\n\n")
9. #Integer Data type
10.variable_integer<- 133L
11.cat(variable_integer,"\n")
12.cat("The data type of variable_integer is ",class(variable_integer),"\n\n")
13. #Complex Data type
14.variable_complex<- 3+2i
15.cat(variable_complex,"\n")
16.cat("The data type of variable_complex is ",class(variable_complex),"\n\n")
17. #Character Data type
18.variable_char<- "Learning r programming"
19.cat(variable_char,"\n")
20.cat("The data type of variable_char is ",class(variable_char),"\n\n")
21. #Raw Data type
22.variable_raw<- charToRaw("Learning r programming")
23.cat(variable_raw,"\n")
24.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:\> Command Prompt
C:\Users\ajet\>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

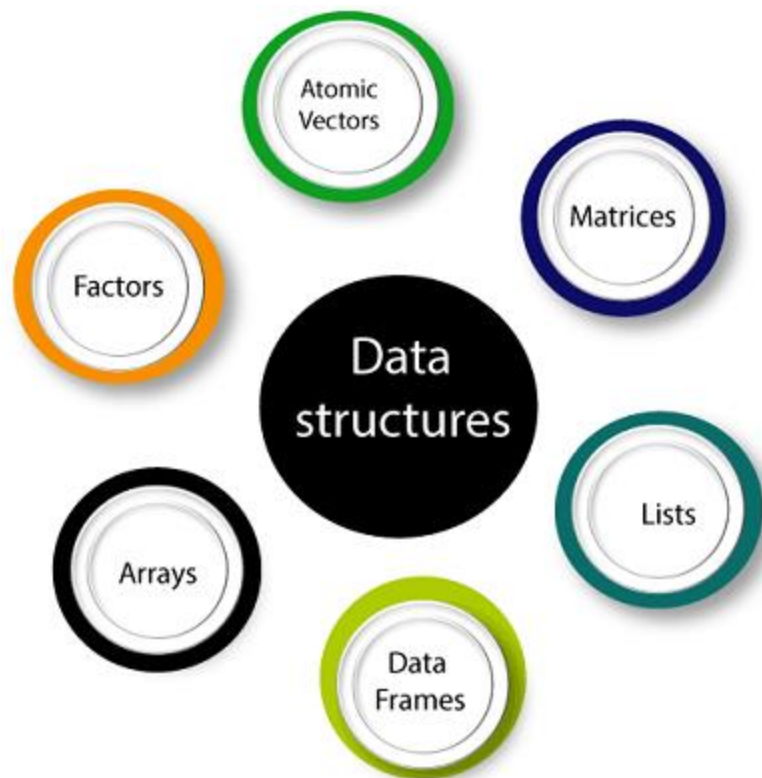
Learning r programming
The data type of variable_char is character

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:



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"** A vector can be one of the following two types:

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:

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
_var_name	Invalid	Variable name can't start with an underscore(_).
var_name, var.name	Valid	Variable can start with a dot, but dot should not be followed by a number. In this case, the variable will be invalid.
var_name%	Invalid	In R, we can't use any special character in the variable name except dot and underscore.
2var_name	Invalid	Variable name cant starts with a numeric digit.
.2var_name	Invalid	A variable name cannot start with a dot which is followed by a digit.

var_name2	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 multiples values into a continuous print output.

1. **# Assignment using equal operator.**
2. variable.1 = 124
3. **# Assignment using leftward operator.**
4. variable.2 <- "Learn R Programming"
5. **# Assignment using rightward operator.**
6. 133L -> variable.3
7. **print(variable.1)**
8. cat ("variable.1 is ", variable.1 ,"\n")
9. cat ("variable.2 is ", variable.2 ,"\n")
- 10.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

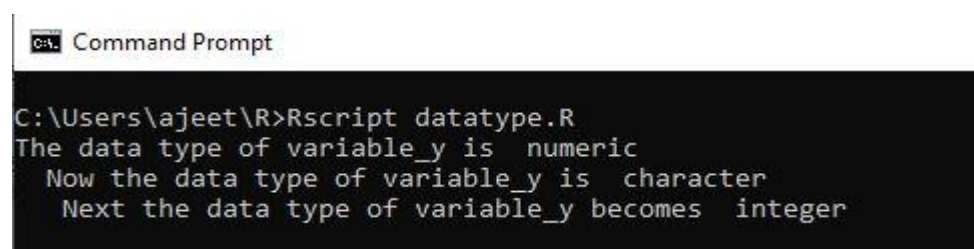
R programming is a dynamically typed language, which means that we can change the data type of the same variable again and again in our program. Because of its dynamic

nature, a variable is 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:

1. `variable_y <- 124`
2. `cat("The data type of variable_y is ", class(variable_y), "\n")`
3. `variable_y <- "Learn R Programming"`
4. `cat(" Now the data type of variable_y is ", class(variable_y), "\n")`
5. `variable_y <- 133L`
6. `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:



```
CA 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
while	function	for
next	break	TRUE
FALSE	NULL	Inf

NaN	NA	NA_integer_
NA_real_	NA_complex_	NA_character_



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:

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

Output:

```
[1] "I am lesser than 15"
>
```

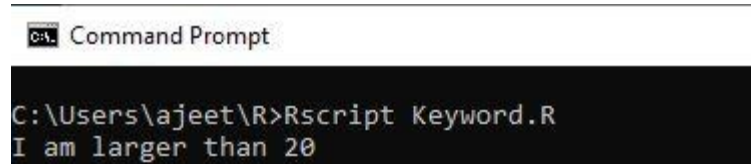
2) else

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:

```
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:



```
C:\> Command Prompt
C:\Users\ajeet\R>Rscript Keyword.R
I am larger than 20
```

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:

```
C:\ Command Prompt
C:\Users\ajeet\R>Rscript Keyword.R
12345
```

4) while

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:

```
C:\ Command Prompt
C:\Users\ajeet\R>Rscript Keyword.R
2018161412108642
```

5) function

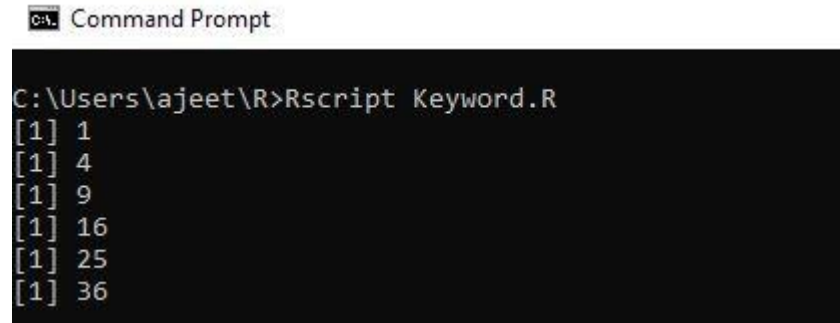
A function is an object in R programming. The keyword function is used to create a user-define function in R. R has some pre-defined functions also, such as seq, mean, and sum.

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:



```
C:\Users\ajeet\R>Rscript Keyword.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

6) for

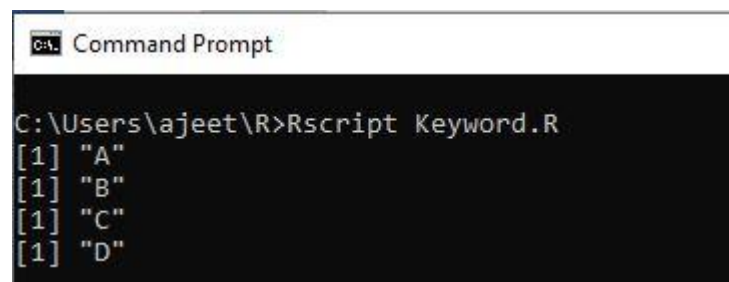
The **for** is a keyword which is used for looping or iterating over a sequence (dictionary, string, list, set or tuple).

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:



```
C:\Users\ajeet\R>Rscript Keyword.R
[1] "A"
[1] "B"
[1] "C"
[1] "D"
```

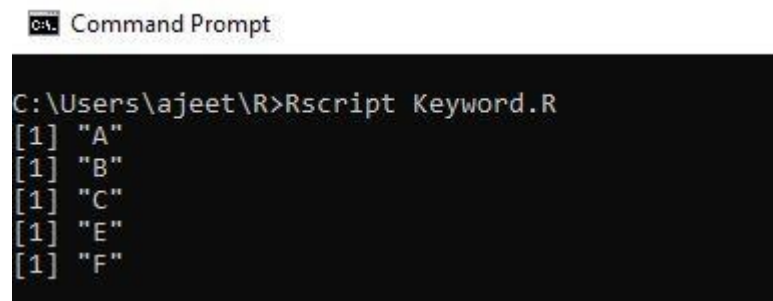
7) next

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

Output:



```
C:\Users\ajeet\R>Rscript Keyword.R
[1] "A"
[1] "B"
[1] "C"
[1] "E"
[1] "F"
```

8) break

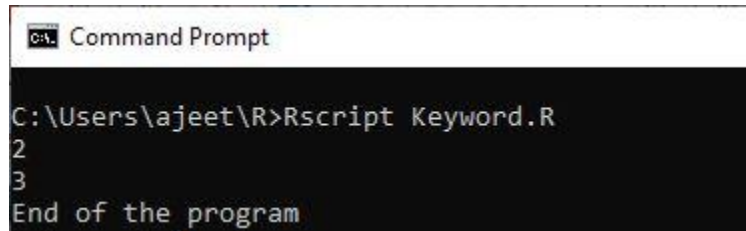
The **break** keyword is used to terminate the loop if the condition is true. The control of the program firstly passes to the outer statement then passes to the body of the break statement.

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")`

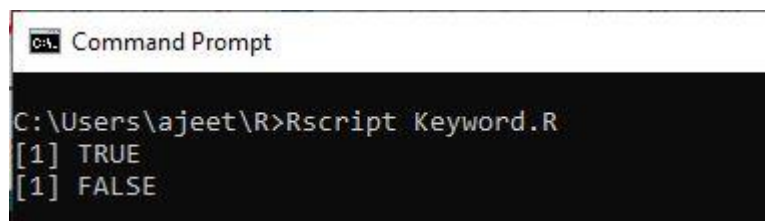
Output:



```
C:\Users\ajeet\R>Rscript Keyword.R
2
3
End of the program
```

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.



```
C:\Users\ajeet\R>Rscript Keyword.R
[1] TRUE
[1] FALSE
```

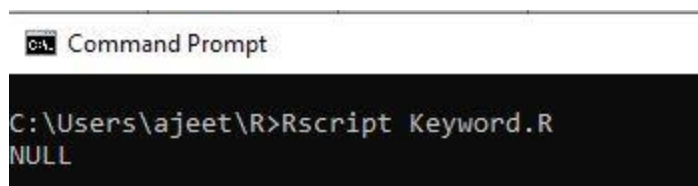
10) NULL

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:



```
C:\Users\ajeet\R>Rscript Keyword.R
NULL
```

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.

Usage

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

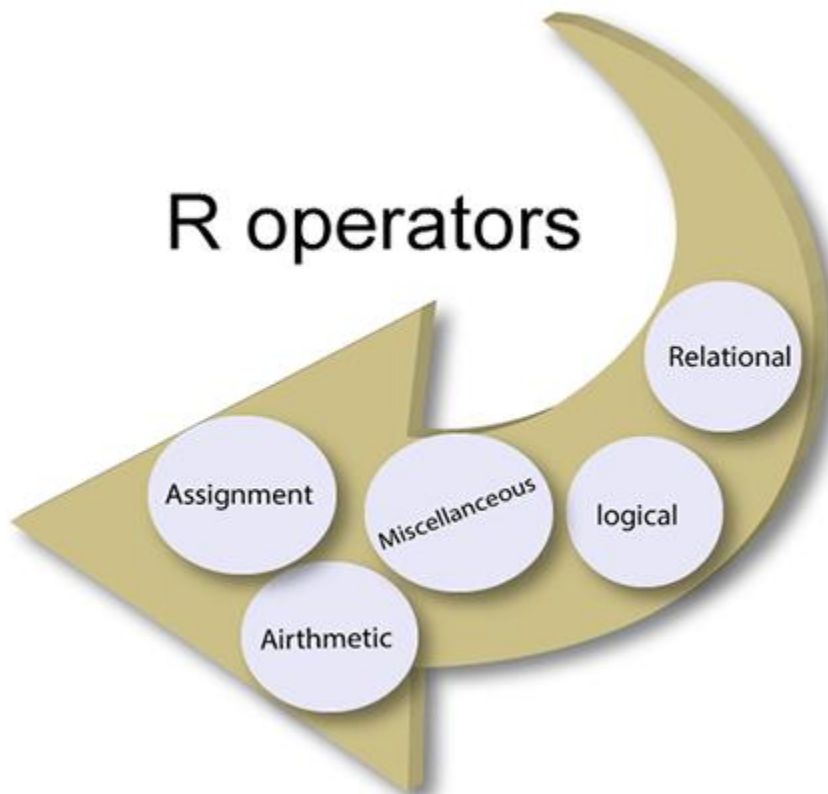
1. `NA`
2. `is.na(x)`
3. `anyNA(x, recursive = FALSE)`
4. `## S3 method for class 'data.frame'`
5. `is.na(x)`
6. `is.na(x) <- value`

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:



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.

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:

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. No	Operator	Description	Example
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 output:</p> <pre>[1] 13.0 8.3 5.0</pre>
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 output:</p> <pre>[1] -9.0 -1.7 3.0</pre>
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:</p> <pre>[1] 22.0 16.5 4.0</pre>
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:</p> <pre>[1] 0.1818182 0.6600000 4.0000000</pre>

5.	%%	This operator is used to find the remainder of the first vector with the second vector. <code>a <- c(2, 3.3, 4)</code>	<pre>b <- c(11, 5, 3) print(a%%b)</pre> <p>It will give us the following output:</p> <pre>[1] 2.0 3.3 0</pre>
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:</p> <pre>[1] 0 0 4</pre>
7.	^	This operator raised the first vector to the exponent of the second vector. <code>a <- c(2, 3.3, 4)</code>	<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>

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 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:</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> <p>It will give us the following output:</p> <pre>[1] FALSE TRUE FALSE</pre>
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 output:</p> <pre>[1] TRUE FALSE TRUE</pre>

S. No	Operator	Description	Example
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> <p>It will give us the following output:</p> <pre>[1] TRUE FALSE TRUE TRUE</pre>
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(a b)</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(a b)</pre> <p>It will give us the following output:</p> <pre>[1] TRUE</pre>

Assignment Operators

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>
--	--	--	------------------------------------------------------------------------------------------------------------------

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

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>

R if Statement

The if statement consists of the Boolean expressions followed by one or more statements. The if statement is the simplest decision-making statement which helps us to take a decision on the basis of the condition.

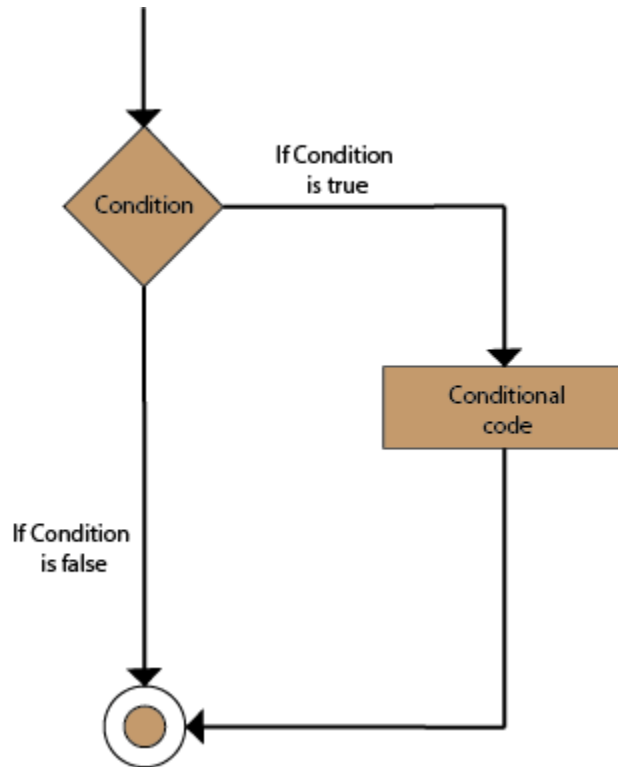
The if statement is a conditional programming statement which performs the function and displays the information if it is proved true.

The block of code inside the if statement will be executed only when the boolean expression evaluates to be true. If the statement evaluates false, then the code which is mentioned after the condition will run.

The syntax of if statement in R is as follows:

1. `if(boolean_expression) {`
2. `// If the boolean expression is true, then statement(s) will be executed.`
3. `}`

Flow Chart



Let see some examples to understand how if statements work and perform a certain task in R.

Example 1

1. `x <- 24L`
2. `y <- "shubham"`
3. `if(is.integer(x))`
4. `{`
5. `print("x is an Integer")`
6. `}`

Output:

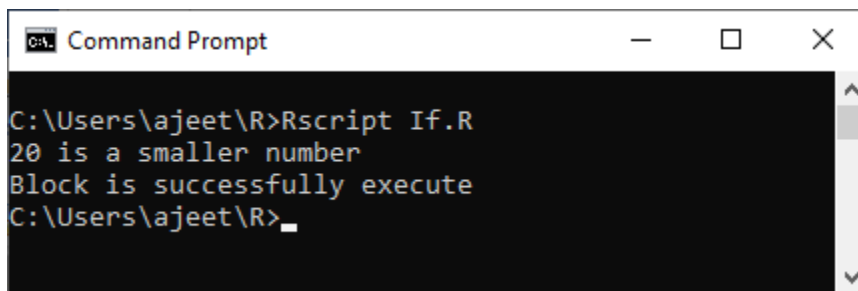
Command Prompt

```
C:\Users\ajeet\R>Rscript Keyword.R  
24 is an Integer
```

Example 2

1. `x <-20`
2. `y<-24`
3. `count=0`
4. `if(x<y)`
5. `{`
6. `cat(x,"is a smaller number\n")`
7. `count=1`
8. `}`
9. `if(count==1){`
10. `cat("Block is successfully execute")`
11. `}`

Output:



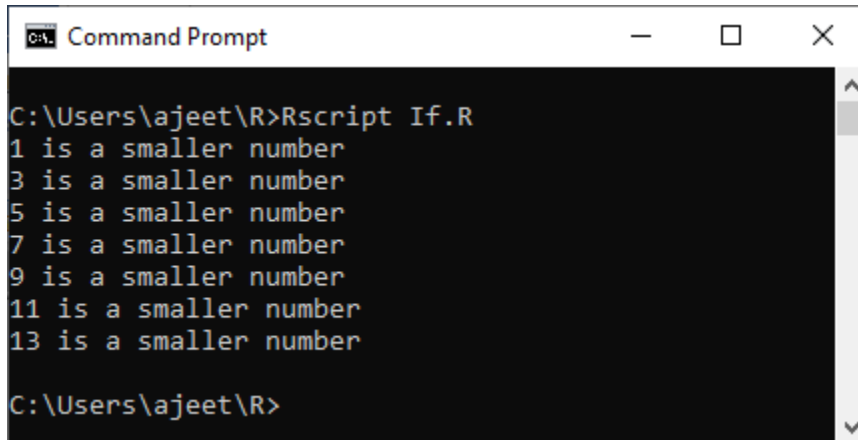
```
Command Prompt  
C:\Users\ajeet\R>Rscript If.R  
20 is a smaller number  
Block is successfully execute  
C:\Users\ajeet\R>
```

Example 3

1. `x <-1`
2. `y<-24`
3. `count=0`
4. `while(x<y){`
5. `cat(x,"is a smaller number\n")`
6. `x=x+2`

7. `if(x==15)`
8. `break`
9. `}`

Output:

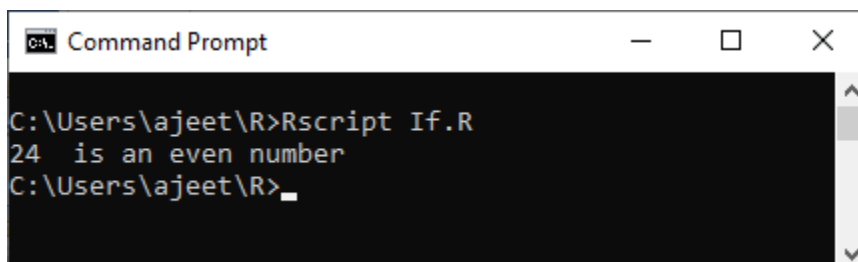


```
Command Prompt
C:\Users\ajeet\R>Rscript If.R
1 is a smaller number
3 is a smaller number
5 is a smaller number
7 is a smaller number
9 is a smaller number
11 is a smaller number
13 is a smaller number
C:\Users\ajeet\R>
```

Example 4

1. `x <-24`
2. `if(x%%2==0){`
3. `cat(x," is an even number")`
4. `}`
5. `if(x%%2!=0){`
6. `cat(x," is an odd number")`
7. `}`

Output:

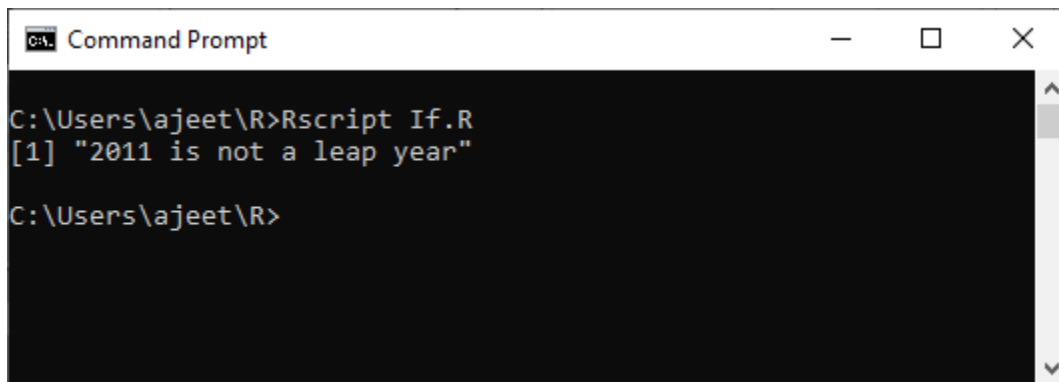


```
Command Prompt
C:\Users\ajeet\R>Rscript If.R
24 is an even number
C:\Users\ajeet\R>
```

Example 5

```
1. year
2. 1 = 2011
3. if(year1 %% 4 == 0) {
4.   if(year1 %% 100 == 0) {
5.     if(year1 %% 400 == 0) {
6.       cat(year,"is a leap year")
7.     } else {
8.       cat(year,"is not a leap year")
9.     }
10.  } else {
11.    cat(year,"is a leap year")
12.  }
13.} else {
14. cat(year,"is not a leap year")
15.}
```

Output:



```
CA Command Prompt
C:\Users\ajeet\R>Rscript If.R
[1] "2011 is not a leap year"
C:\Users\ajeet\R>
```

If-else statement

In the if statement, the inner code is executed when the condition is true. The code which is outside the if block will be executed when the if condition is false.

There is another type of decision-making statement known as the if-else statement. An if-else statement is the if statement followed by an else statement. An if-else statement, else statement will be executed when the boolean expression will false. In

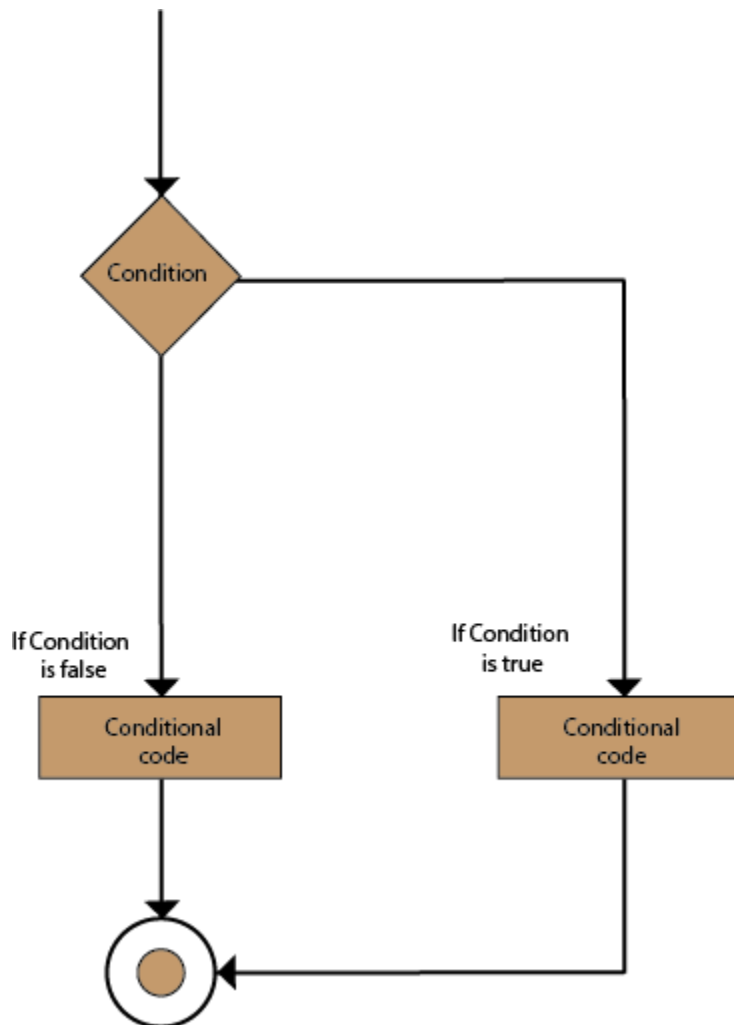
simple words, If a Boolean expression will have true value, then the if block gets executed otherwise, the else block will get executed.

R programming treats any non-zero and non-null values as true, and if the value is either zero or null, then it treats them as false.

The basic syntax of If-else statement is as follows:

1. **if**(boolean_expression) {
2. // statement(s) will be executed if the boolean expression is true.
3. } **else** {
4. // statement(s) will be executed if the boolean expression is false.
5. }

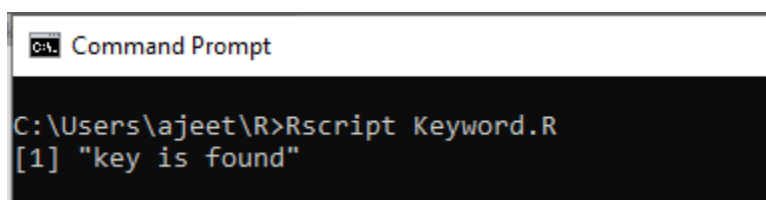
Flow Chart



Example 1

```
1. # local variable definition
2. a<- 100
3. #checking boolean condition
4. if(a<20){
5.   # if the condition is true then print the following
6.   cat("a is less than 20\n")
7. }else{
8.   # if the condition is false then print the following
9.   cat("a is not less than 20\n")
10.}
11.cat("The value of a is", a)
```

Output:



```
C:\Users\ajeet\R>Rscript Keyword.R
[1] "key is found"
```

Example 2

```
1. x <- c("Hardwork","is","the","key","of","success")
2.
3. if("key" %in% x) {
4.   print("key is found")
5. } else {
6.   print("key is not found")
7. }
```

Output:

```
Command Prompt
C:\Users\ajeet\R>Rscript Keyword.R
[1] "key is found"
```

Example 3

```
1. a<- 100
2. #checking boolean condition
3. if(a<20){
4.   cat("a is less than 20")
5.   if(a%%2==0){
6.     cat(" and an even number\n")
7.   }
8.   else{
9.     cat(" but not an even number\n")
10.  }
11.}else{
12.  cat("a is greater than 20")
13.  if(a%%2==0){
14.    cat(" and an even number\n")
15.  }
16.  else{
17.    cat(" but not an even number\n")
18.  }
19.}
```

Output:

```
Command Prompt
C:\Users\ajeet\R>Rscript if-else.R
a is greater than 20 and an even number
The value of a is 100
C:\Users\ajeet\R>_
```

Example 4

1. `a<- 'u'`
2. `if(a=='a' || a=='e' || a=='i' || a=='o' || a=='u' || a=='A' || a=='E' || a=='I' || a=='O' || a=='U'){`
3. `cat("character is a vowel\n")`
4. `}else{`
5. `cat("character is a constant")`
6. `}`
7. `cat("character is =",a)`
8. `}`

Output:

```
Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

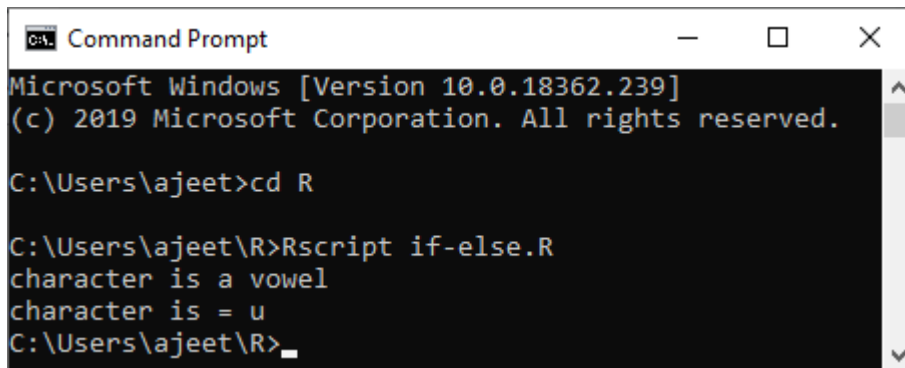
C:\Users\ajeet\R>Rscript if-else.R
character is a vowel
character is = u
C:\Users\ajeet\R>_
```

Example 5

1. `a<- 'u'`
2. `if(a=='a' || a=='e' || a=='i' || a=='o' || a=='u' || a=='A' || a=='E' || a=='I' || a=='O' || a=='U'){`

```
3.   cat("character is a vowel\n")
4. }else{
5.   cat("character is a constant")
6. }
7. cat("character is =",a)
8. }
```

Output:



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript if-else.R
character is a vowel
character is = u
C:\Users\ajeet\R>
```

R else if statement

This statement is also known as nested if-else statement. The if statement is followed by an optional else if..... else statement. This statement is used to test various condition in a single if.....else if statement. There are some key points which are necessary to keep in mind when we are using the if.....else if.....else statement. These points are as follows:

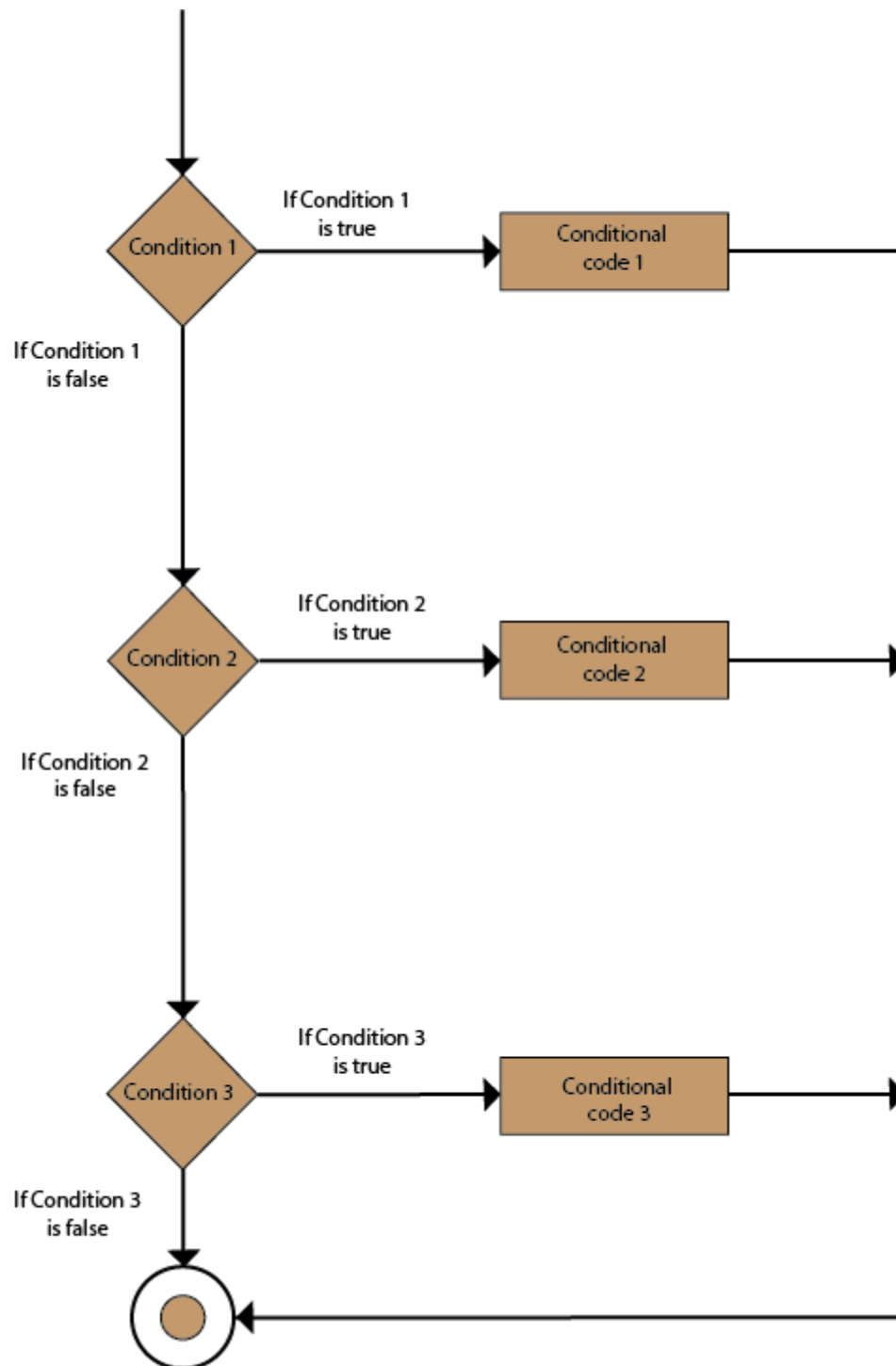
1. **if** statement can have either zero or one **else** statement and it must come after any **else if's** statement.
2. **if** statement can have many **else if's** statement and they come before the else statement.
3. Once an **else if** statement succeeds, none of the remaining **else if's** or **else's** will be tested.

The basic syntax of If-else statement is as follows:

1. **if**(boolean_expression **1**) {


```
2. // This block executes when the boolean expression 1 is true.
3. } else if( boolean_expression 2) {
4. // This block executes when the boolean expression 2 is true.
5. } else if( boolean_expression 3) {
6. // This block executes when the boolean expression 3 is true.
7. } else {
8. // This block executes when none of the above condition is true.
9. }
```

Flow Chart



Example 1

1. `age <- readline(prompt="Enter age: ")`
2. `age <- as.integer(age)`

3. **if**(age<18)
4. print("You are child")
5. **else if**(age>30)
6. print("You are old guy")
7. **else**
8. print("You are adult")

Output:

```
> age <- readline(prompt="Enter your age: ")
Enter your age: 21
> age <- as.integer(age)
> if(age<18){
+ print("You are child")
+ }else if(age>30){
+ print("You are old guy")
+ }else{
+ print("You are Adult")
+ }
[1] "You are Adult"
>
```

Example 2

1. marks=83;
2. **if**(marks>75){
3. print("First class")
4. }**else if**(marks>65){
5. print("Second class")
6. }**else if**(marks>55){
7. print("Third class")
8. }**else**{
9. print("Fail")
10. }

Output:

```
Rterm (64-bit)
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.

[Previously saved workspace restored]

> marks=83;
> if(marks>75){
+ print("First class")
+ }else if(marks>65){
+ print("Second class")
+ }else if(marks>55){
+ print("Third class")
+ }else{
+ print("Fail")
+ }
[1] "First class"
> _
```

Example 3

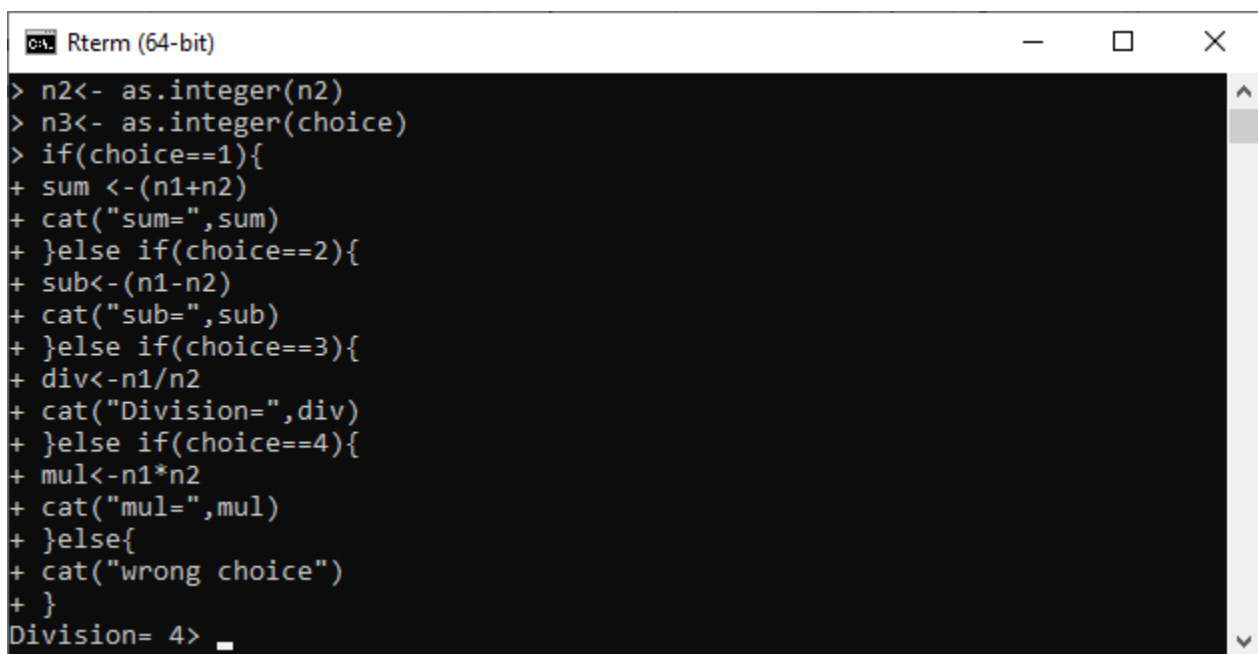
1. `cat("1) For Addition\n")`
2. `cat("2) For Subtraction\n")`
3. `cat("3) For Division\n")`
4. `cat("4) For multiplication\n")`
5. `n1<-readline(prompt="Enter first number:")`
6. `n2<-readline(prompt="Enter second number:")`
7. `choice<-readline(prompt="Enter your choice:")`
8. `n1<- as.integer(n1)`
9. `n2<- as.integer(n2)`
10. `choice<- as.integer(choice)`
11. `if(choice==1){`
12. `sum <-(n1+n2)`
13. `cat("sum=",sum)`
14. `}else if(choice==2){`
15. `sub<-(n1-n2)`
16. `cat("sub=",sub)`
17. `}else if(choice==3){`

```

18.  div<-n1/n2
19.  cat("Division=",div)
20. }else if(choice==4){
21.  mul<-n1*n2
22.  cat("mul=",mul)
23. }else{
24.  cat("wrong choice")
25. }

```

Output:



```

Rterm (64-bit)
> n2<- as.integer(n2)
> n3<- as.integer(choice)
> if(choice==1){
+ sum <- (n1+n2)
+ cat("sum=",sum)
+ }else if(choice==2){
+ sub<-(n1-n2)
+ cat("sub=",sub)
+ }else if(choice==3){
+ div<-n1/n2
+ cat("Division=",div)
+ }else if(choice==4){
+ mul<-n1*n2
+ cat("mul=",mul)
+ }else{
+ cat("wrong choice")
+ }
Division= 4>

```

Example 4

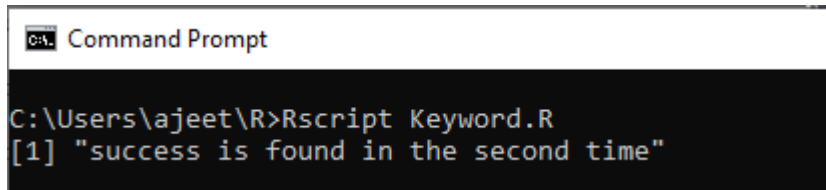
```

1. x <- c("Hardwork","is","the","key","of","success")
2. if("Success" %in% x) {
3.   print("success is found in the first time")
4. } else if ("success" %in% x) {
5.   print("success is found in the second time")
6. } else {
7.   print("No success found")

```

8. }

Output:

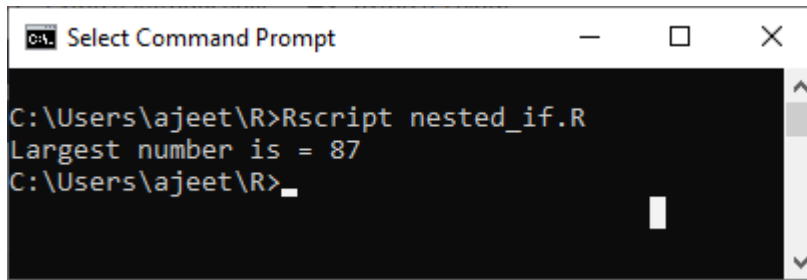


```
Command Prompt
C:\Users\ajeet\R>Rscript Keyword.R
[1] "success is found in the second time"
```

Example 5

```
1. n1=4
2. n2=87
3. n3=43
4. n4=74
5. if(n1>n2){
6.     if(n1>n3&& n1>n4){
7.         largest=n1
8.     }
9. }else if(n2>n3){
10.     if(n2>n1&& n2>n4){
11.         largest=n2
12.     }
13. }else if(n3>n4){
14.     if(n3>n1&& n3>n2){
15.         largest=n3
16.     }
17. }else{
18.     largest=n4
19. }
20. cat("Largest number is =",largest)
```

Output:



```
C:\Users\ajeet\R>Rscript nested_if.R
Largest number is = 87
C:\Users\ajeet\R>
```

R Switch Statement

A switch statement is a selection control mechanism that allows the value of an expression to change the control flow of program execution via map and search.

The switch statement is used in place of long if statements which compare a variable with several integral values. It is a multi-way branch statement which provides an easy way to dispatch execution for different parts of code. This code is based on the value of the expression.

This statement allows a variable to be tested for equality against a list of values. A switch statement is a little bit complicated. To understand it, we have some key points which are as follows:

- If expression type is a character string, the string is matched to the listed cases.
- If there is more than one match, the first match element is used.
- No default case is available.
- If no case is matched, an unnamed case is used.

There are basically two ways in which one of the cases is selected:

1) Based on Index

If the cases are values like a character vector, and the expression is evaluated to a number than the expression's result is used as an index to select the case.

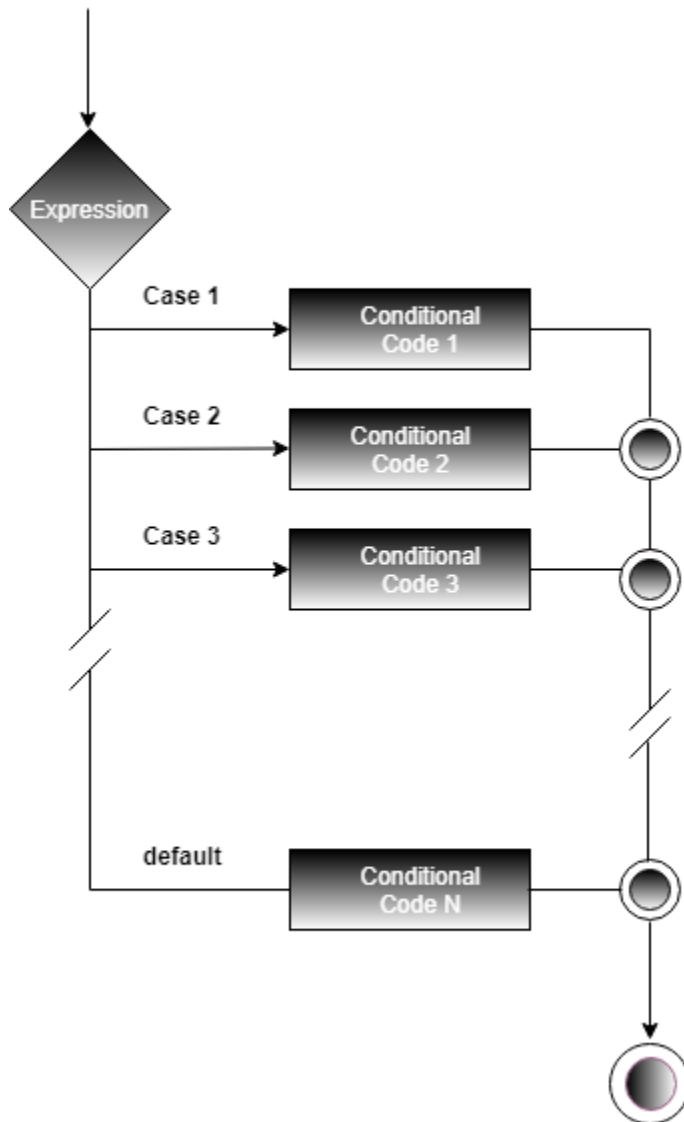
2) Based on Matching Value

When the cases have both case value and output value like ["case_1"="value1"], then the expression value is matched against case values. If there is a match with the case, the corresponding value is the output.

The basic syntax of If-else statement is as follows:

1. **switch**(expression, case1, case2, case3....)

Flow Chart

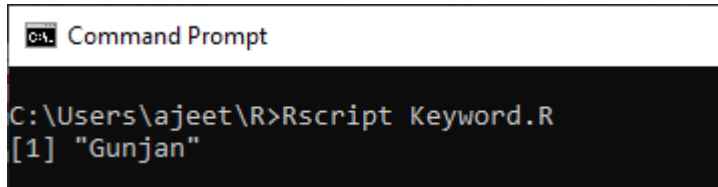


Example 1

1. `x <- switch(`
2. `3,`
3. `"Shubham",`

4. "Nishka",
5. "Gunjan",
6. "Sumit"
7.)
8. print(x)

Output:

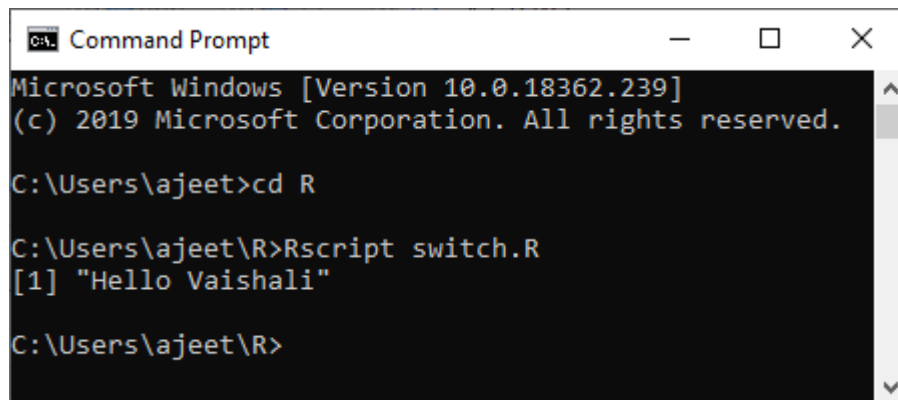


```
C:\Users\ajeet\R>Rscript Keyword.R  
[1] "Gunjan"
```

Example 2

1. ax= 1
2. bx = 2
3. y = switch(
4. ax+bx,
5. "Hello, Shubham",
6. "Hello Arpita",
7. "Hello Vaishali",
8. "Hello Nishka"
9.)
10. print (y)

Output:



```
Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

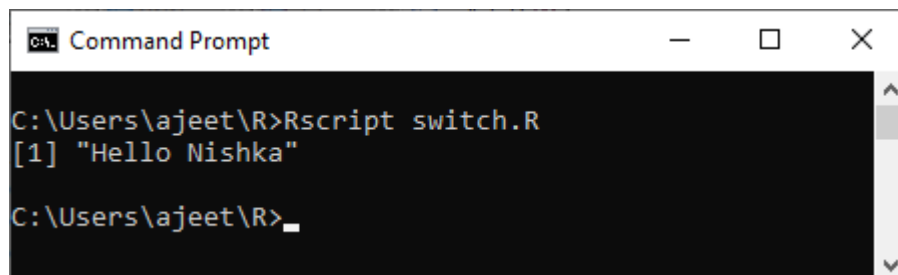
C:\Users\ajeet\R>Rscript switch.R
[1] "Hello Vaishali"

C:\Users\ajeet\R>
```

Example 3

1. `y = "18"`
2. `x = switch(`
3. `y,`
4. `"9"="Hello Arpita",`
5. `"12"="Hello Vaishali",`
6. `"18"="Hello Nishka",`
7. `"21"="Hello Shubham"`
8. `)`
- 9.
10. `print (x)`

Output:



```
Command Prompt

C:\Users\ajeet\R>Rscript switch.R
[1] "Hello Nishka"

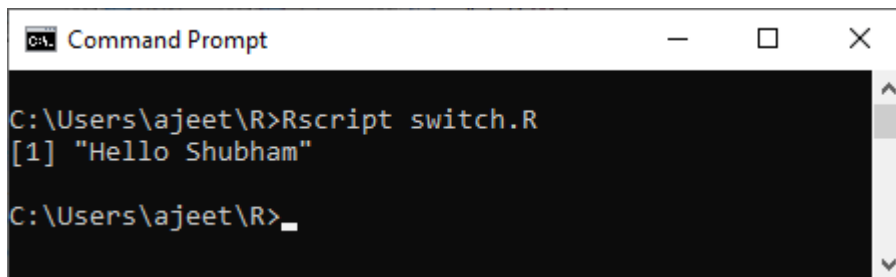
C:\Users\ajeet\R>_
```

Example 4

1. `x= "2"`
2. `y="1"`

```
3. a = switch(  
4.   paste(x,y,sep=""),  
5.   "9"="Hello Arpita",  
6.   "12"="Hello Vaishali",  
7.   "18"="Hello Nishka",  
8.   "21"="Hello Shubham"  
9. )  
10.  
11. print (a)
```

Output:

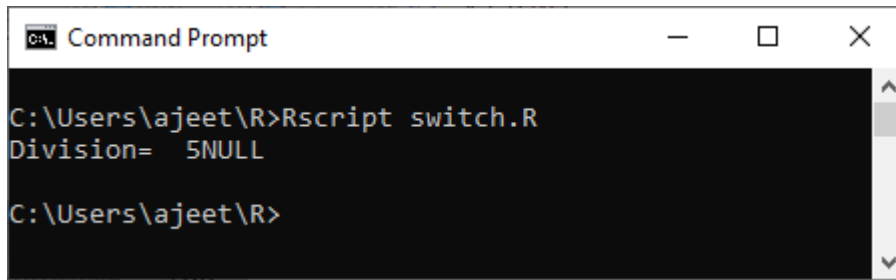


```
C:\Users\ajeet\R>Rscript switch.R  
[1] "Hello Shubham"  
C:\Users\ajeet\R>
```

Example 5

```
1. y = "18"  
2. a=10  
3. b=2  
4. x = switch(  
5.   y,  
6.   "9"=cat("Addition=",a+b),  
7.   "12"=cat("Subtraction =",a-b),  
8.   "18"=cat("Division= ",a/b),  
9.   "21"=cat("multiplication =",a*b)  
10.)  
11.  
12. print (x)
```

Output:

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the user at the C:\Users\ajet\R directory. They have run the command "Rscript switch.R", which has produced the output "Division= 5NULL". The prompt is now waiting for the next command.

```
C:\Users\ajet\R>Rscript switch.R
Division= 5NULL
C:\Users\ajet\R>
```

R next Statement

The next statement is used to skip any remaining statements in the loop and continue executing. In simple words, a next statement is a statement which skips the current iteration of a loop without terminating it. When the next statement is encountered, the R parser skips further evaluation and starts the next iteration of the loop.

This statement is mostly used with for loop and while loop.

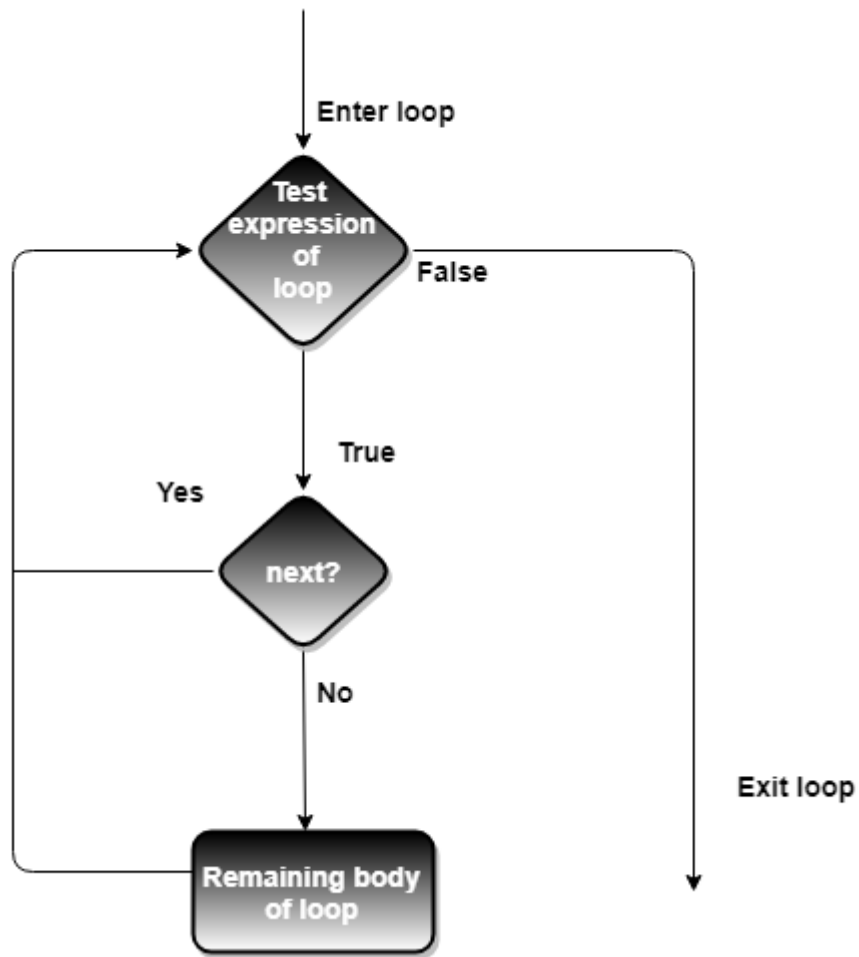
Note: In else branch of the if-else statement, the next statement can also be used.

Syntax

There is the following syntax for creating the next statement in R

1. next

Flowchart



Example 1: next in repeat loop

```
1. a <- 1
2. repeat {
3.   if(a == 10)
4.     break
5.   if(a == 5){
6.     next
7.   }
8.   print(a)
9.   a <- a+1
10. }
```

Output:

```
Command Prompt - Rscript next.R

C:\Users\ajeet\R>Rscript next.R
[1] 1
[1] 2
[1] 3
[1] 4
```

Example 2: next in while loop

1. `a<-1`
2. `while (a < 10) {`
3. `if(a==5)`
4. `next`
5. `print(a)`
6. `a = a + 1`
7. `}`

Output:

```
Command Prompt - Rscript next.R

Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

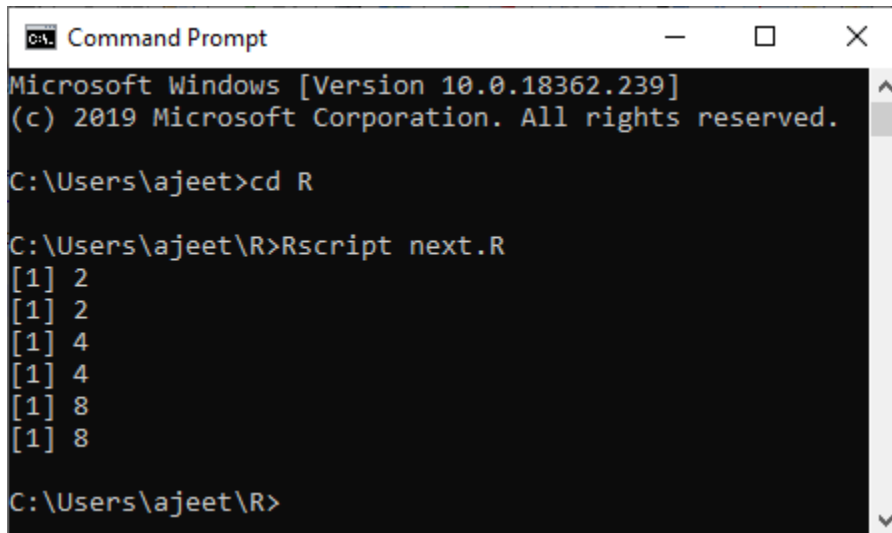
C:\Users\ajeet\R>Rscript next.R
[1] 1
[1] 2
[1] 3
[1] 4
```

Example 3: next in for loop

1. `x <- 1:10`
2. `for (val in x) {`
3. `if (val == 3){`
4. `next`

5. }
6. print(val)
7. }

Output:



```
Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript next.R
[1] 2
[1] 2
[1] 4
[1] 4
[1] 8
[1] 8

C:\Users\ajeet\R>
```

Example 4

1. a1<- c(10L,-11L,12L,-13L,14L,-15L,16L,-17L,18L)
2. sum<-0
3. for(i in a1){
4. if(i<0){
5. next
6. }
7. sum=sum+i
8. }
9. cat("The sum of all positive numbers in array is=",sum)

Output:

```
Command Prompt
C:\Users\ajeet\R>Rscript nested_if.R
The sum of all positive numbers in array is= 70
C:\Users\ajeet\R>_
```

Example 5

1. `j<-0`
2. `while(j<10){`
3. `if (j==7){`
4. `j=j+1`
5. `next`
6. `}`
7. `cat("\nnumber is =",j)`
8. `j=j+1`
9. `}`

Output:

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

number is = 0
number is = 1
number is = 2
number is = 3
number is = 4
number is = 5
number is = 6
number is = 8
number is = 9
C:\Users\ajeet\R>
```

R Break Statement

In the R language, the break statement is used to break the execution and for an immediate exit from the loop. In nested loops, break exits from the innermost loop only and control transfer to the outer loop.

It is useful to manage and control the program execution flow. We can use it to various loops like: for, repeat, etc.

There are basically two usages of break statement which are as follows:

1. When the break statement is inside the loop, the loop terminates immediately and program control resumes on the next statement after the loop.
2. It is also used to terminate a case in the switch statement.

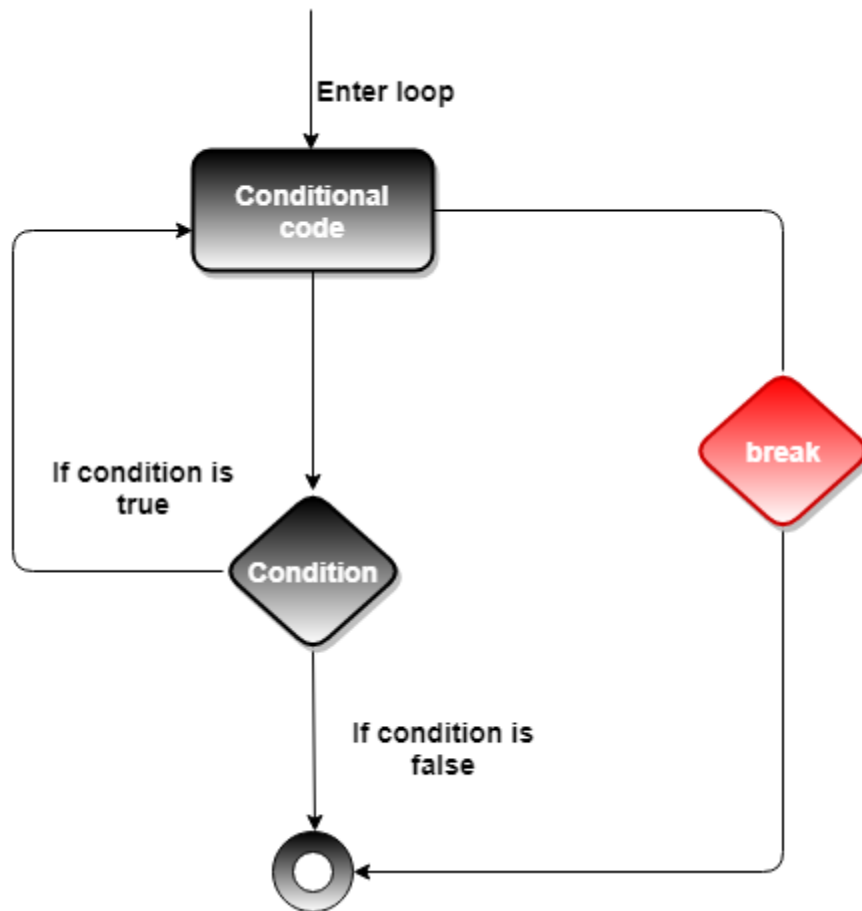
Note: *We can also use break statement inside the else branch of if...else statement.*

Syntax

There is the following syntax for creating a break statement in R

1. **break**

Flowchart



Example 1: Break in repeat loop

```
1. a <- 1
2. repeat {
3.   print("hello");
4.   if(a >= 5)
5.     break
6.   a <- a + 1
7. }
```

Output:

```
Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript break.R
[1] "hello"
[1] "hello"
[1] "hello"
[1] "hello"
[1] "hello"

C:\Users\ajeet\R>
```

Example 2

1. `v <- c("Hello","loop")`
2. `count <- 2`
3. `repeat {`
4. `print(v)`
5. `count <- count + 1`
6. `if(count > 5) {`
7. `break`
8. `}`
9. `}`

Output:

```
Command Prompt

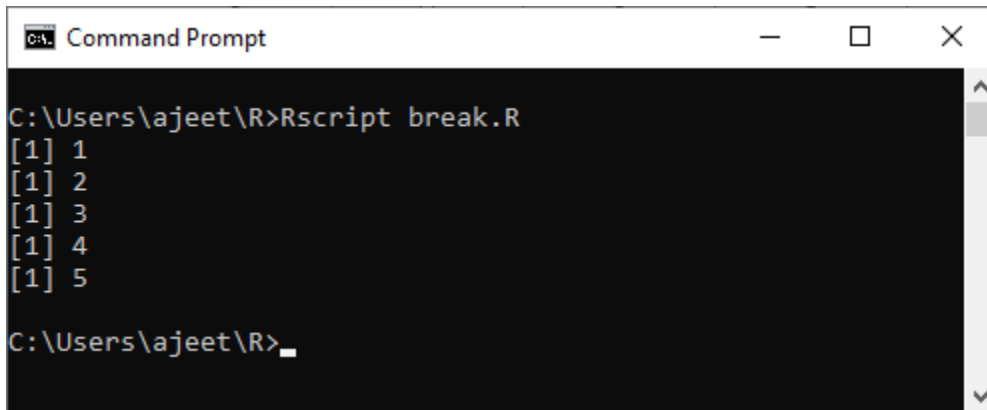
C:\Users\ajeet\R>Rscript break.R
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"

C:\Users\ajeet\R>
```

Example 3: Break in while loop

```
1. a<-1
2. while (a < 10) {
3.   print(a)
4.   if(a==5)
5.     break
6.   a = a + 1
7. }
```

Output:

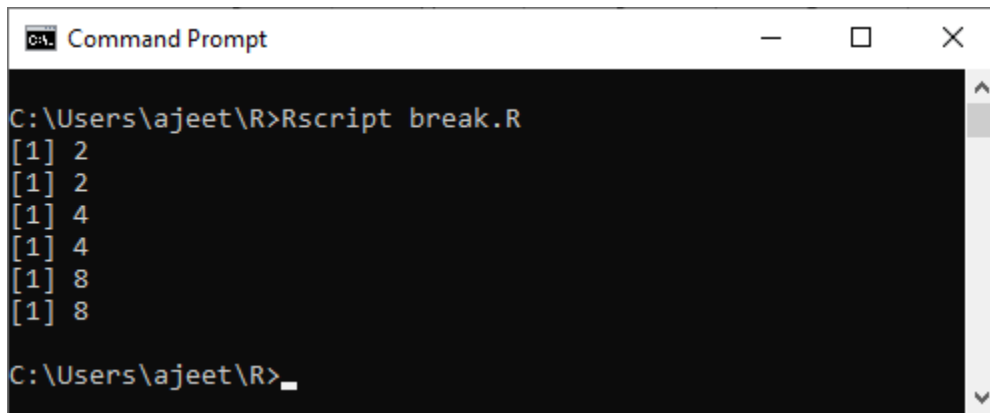


```
Command Prompt
C:\Users\ajet\R>Rscript break.R
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
C:\Users\ajet\R>
```

Example 4: Break in for loop

```
1. for (i in c(2,4,6,8)) {
2.   for (j in c(1,3)) {
3.     if (i==6)
4.       break
5.     print(i)
6.   }
7. }
```

Output:

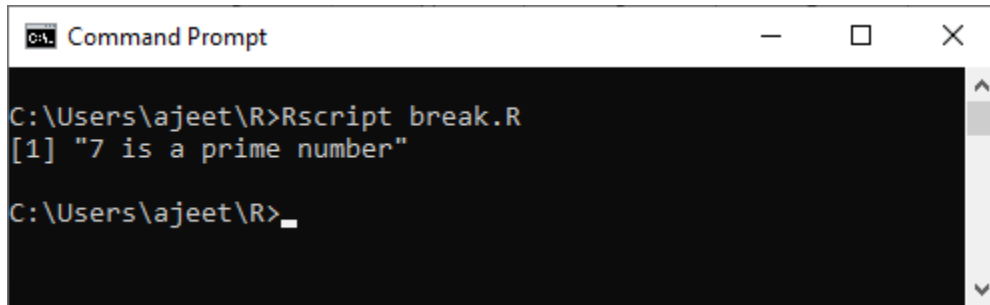


```
Command Prompt
C:\Users\ajeet\R>Rscript break.R
[1] 2
[1] 2
[1] 4
[1] 4
[1] 8
[1] 8
C:\Users\ajeet\R>
```

Example 5

```
1. num=7
2. flag = 0
3. if(num> 1) {
4.     flag = 1
5.     for(i in 2:(num-1)) {
6.         if ((num %% i) == 0) {
7.             flag = 0
8.             break
9.         }
10.    }
11. }
12. if(num == 2)  flag = 1
13. if(flag == 1) {
14.     print(paste(num,"is a prime number"))
15. } else {
16.     print(paste(num,"is not a prime number"))
17. }
```

Output:



```
Command Prompt
C:\Users\ajet\R>Rscript break.R
[1] "7 is a prime number"
C:\Users\ajet\R>
```

R For Loop

A for loop is the most popular control flow statement. A for loop is used to iterate a vector. It is similar to the while loop. There is only one difference between for and while, i.e., in while loop, the condition is checked before the execution of the body, but in for loop condition is checked after the execution of the body.

There is the following syntax of For loop in C/C++:

1. **for** (initialization_Statement; test_Expression; update_Statement)
2. {
3. // statements inside the body of the loop
4. }

How For loop works in C/C++?

The for loop in C and C++ is executed in the following way:

- The initialization statement of for loop is executed only once.
- After the initialization process, the test expression is evaluated. The for loop is terminated when the test expression is evaluated to false.
- The statements inside the body of for loop are executed, and expression is updated if the test expression is evaluated to true.
- The test expression is again evaluated.
- The process continues until the test expression is false. The loop terminates when the test expression is false.

For loop in R Programming

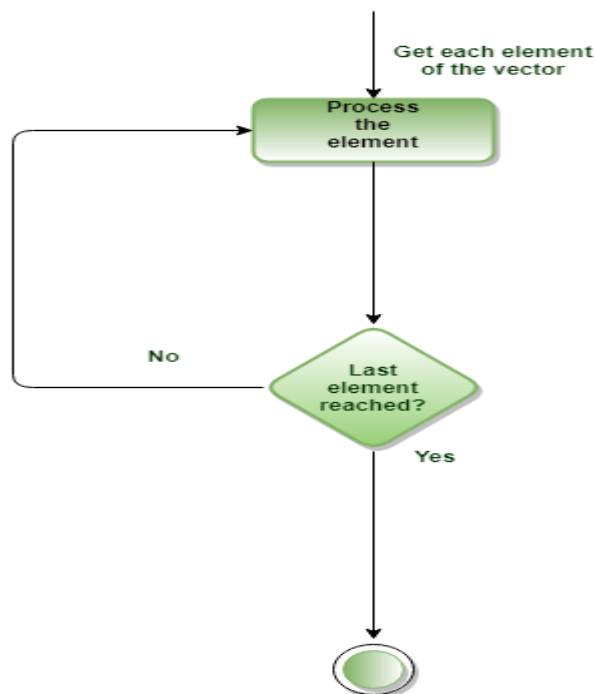
In R, a for loop is a way to repeat a sequence of instructions under certain conditions. It allows us to automate parts of our code which need repetition. In simple words, a for loop is a repetition control structure. It allows us to efficiently write the loop that needs to execute a certain number of time.

In R, a for loop is defined as :

1. It starts with the keyword `for` like C or C++.
2. Instead of initializing and declaring a loop counter variable, we declare a variable which is of the same type as the base type of the vector, matrix, etc., followed by a colon, which is then followed by the array or matrix name.
3. In the loop body, use the loop variable rather than using the indexed array element.
4. There is a following syntax of for loop in R:

1. **for** (value in vector) {
2. statements
3. }

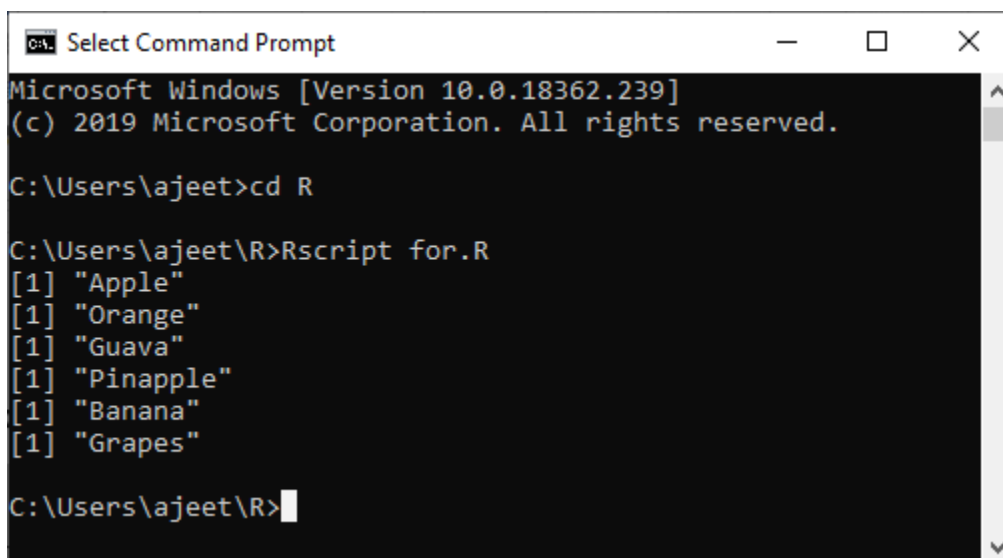
Flowchart



Example 1: We iterate all the elements of a vector and print the current value.

1. # Create fruit vector
2. fruit <- c('Apple', 'Orange', 'Guava', 'Pinapple', 'Banana', 'Grapes')
3. # Create the **for** statement
4. **for** (i in fruit){
5. print(i)
6. }

Output



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

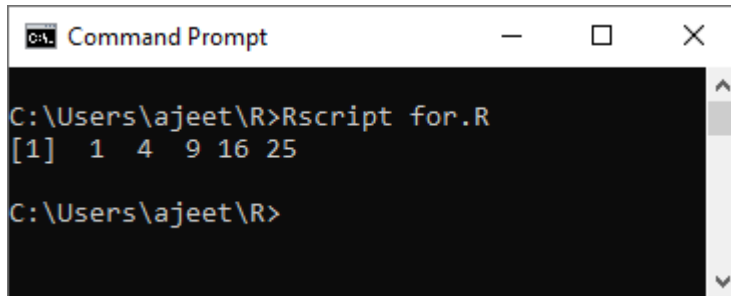
C:\Users\ajeet\R>Rscript for.R
[1] "Apple"
[1] "Orange"
[1] "Guava"
[1] "Pinapple"
[1] "Banana"
[1] "Grapes"

C:\Users\ajeet\R>
```

Example 2: creates a non-linear function with the help of the polynomial of x between 1 and 5 and store it in a list.

1. # Creating an empty list
2. list <- c()
3. # Creating a **for** statement to populate the list
4. **for** (i in seq(1, 5, by=1)) {
5. list[[i]] <- i*i
6. }
7. print(list)

Output



```
C:\Users\ajeet\R>Rscript for.R
[1] 1 4 9 16 25
C:\Users\ajeet\R>
```

Example 3: For loop over a matrix

1. # Creating a matrix
2. `mat <- matrix(data = seq(10, 21, by=1), nrow = 6, ncol = 2)`
3. # Creating the loop with r and c to iterate over the matrix
4. `for (r in 1:nrow(mat))`
5. `for (c in 1:ncol(mat))`
6. `print(paste("mat[", r, ", ", c, "]= ", mat[r,c]))`
7. `print(mat)`

Output

```
Command Prompt
C:\Users\ajeet\R>Rscript for.R
[1] "mat[ 1 , 1 ]= 10"
[1] "mat[ 1 , 2 ]= 16"
[1] "mat[ 2 , 1 ]= 11"
[1] "mat[ 2 , 2 ]= 17"
[1] "mat[ 3 , 1 ]= 12"
[1] "mat[ 3 , 2 ]= 18"
[1] "mat[ 4 , 1 ]= 13"
[1] "mat[ 4 , 2 ]= 19"
[1] "mat[ 5 , 1 ]= 14"
[1] "mat[ 5 , 2 ]= 20"
[1] "mat[ 6 , 1 ]= 15"
[1] "mat[ 6 , 2 ]= 21"
      [,1] [,2]
[1,]   10  16
[2,]   11  17
[3,]   12  18
[4,]   13  19
[5,]   14  20
[6,]   15  21

C:\Users\ajeet\R>
```

Example 4: For loop over a list

1. # Create a list with three vectors
2. `fruit <- list(Basket = c('Apple', 'Orange', 'Guava', 'Pinapple', 'Banana', 'Grapes'),`
3. `Money = c(10, 12, 15), purchase = TRUE)`
4. `for (p in fruit)`
5. `{`
6. `print(p)`
7. `}`

Output

```

C:\Users\ajeet\R>Rscript for.R
[1] "Apple" "Orange" "Guava" "Pinapple" "Banana" "Grapes"
[1] 10 12 15
[1] TRUE
C:\Users\ajeet\R>_

```

Example 5: count the number of even numbers in a vector. # Create a list with three vectors.

1. `x <- c(2,5,3,9,8,11,6,44,43,47,67,95,33,65,12,45,12)`
2. `count <- 0`
3. `for (val in x) {`
4. `if(val %% 2 == 0) count = count+1`
5. `}`
6. `print(count)`

Output

```

C:\Users\ajeet\R>Rscript for.R
[1] 6
C:\Users\ajeet\R>_

```

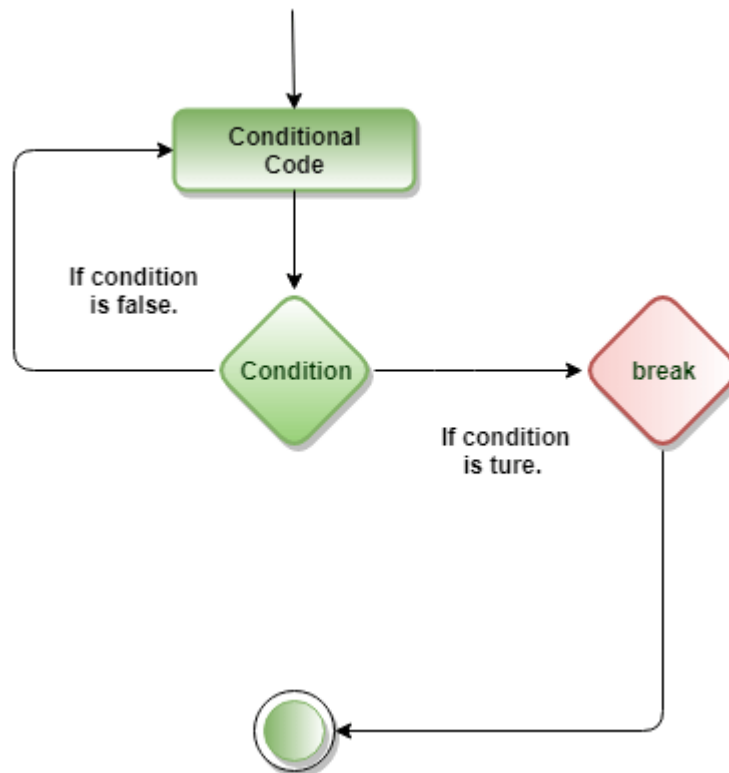
R repeat loop

A repeat loop is used to iterate a block of code. It is a special type of loop in which there is no condition to exit from the loop. For exiting, we include a break statement with a user-defined condition. This property of the loop makes it different from the other loops.

A repeat loop constructs with the help of the repeat keyword in R. It is very easy to construct an infinite loop in R.

The basic syntax of the repeat loop is as follows:

```
1. repeat {  
2.   commands  
3.   if(condition) {  
4.     break  
5.   }  
6. }
```



Flowchart

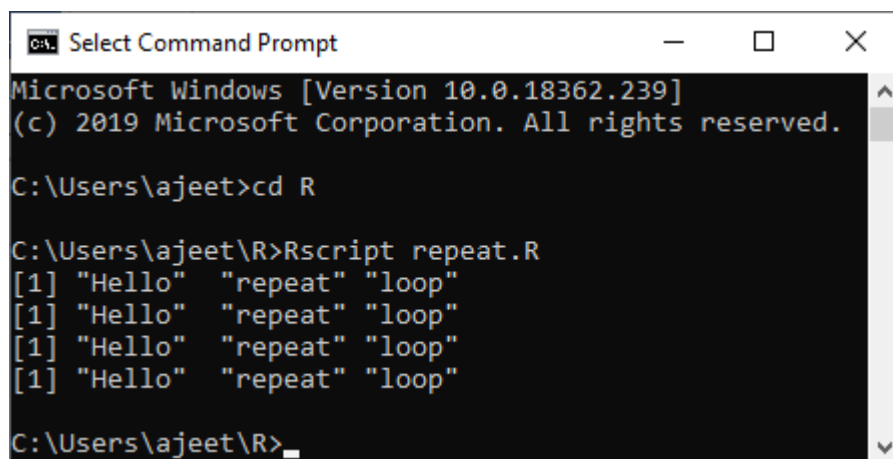
1. First, we have to initialize our variables than it will enter into the Repeat loop.
2. This loop will execute the group of statements inside the loop.
3. After that, we have to use any expression inside the loop to exit.
4. It will check for the condition. It will execute a break statement to exit from the loop
5. If the condition is true.

6. The statements inside the repeat loop will be executed again if the condition is false.

Example 1:

```
1. v <- c("Hello","repeat","loop")
2. cnt <- 2
3. repeat {
4.   print(v)
5.   cnt <- cnt+1
6.
7.   if(cnt > 5) {
8.     break
9.   }
10.}
```

Output



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript repeat.R
[1] "Hello" "repeat" "loop"
[1] "Hello" "repeat" "loop"
[1] "Hello" "repeat" "loop"
[1] "Hello" "repeat" "loop"

C:\Users\ajeet\R>
```

Example 2:

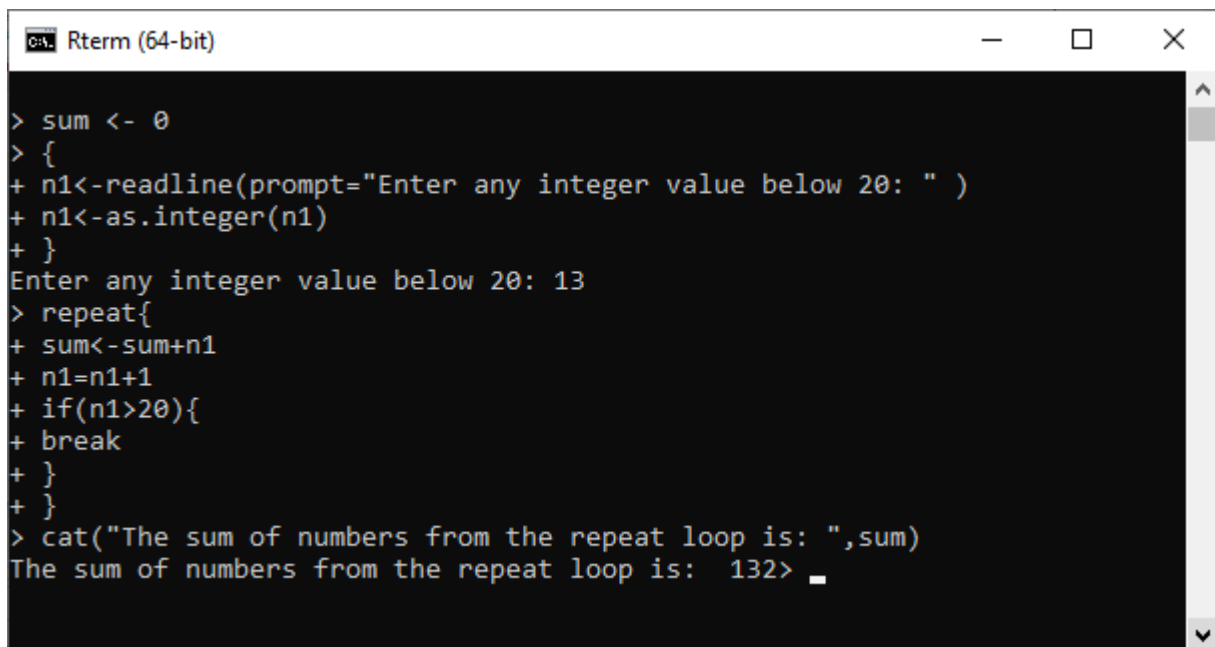
```
1. sum <- 0
2. {
3.   n1<-readline(prompt="Enter any integer value below 20: ")
4.   n1<-as.integer(n1)
5. }
```

```

6. repeat{
7.   sum<-sum+n1
8.   n1n1=n1+1
9.   if(n1>20){
10.    break
11.  }
12.}
13.cat("The sum of numbers from the repeat loop is: ",sum)

```

Output



```

> sum <- 0
> {
+ n1<-readline(prompt="Enter any integer value below 20: " )
+ n1<-as.integer(n1)
+ }
Enter any integer value below 20: 13
> repeat{
+ sum<-sum+n1
+ n1=n1+1
+ if(n1>20){
+ break
+ }
+ }
> cat("The sum of numbers from the repeat loop is: ",sum)
The sum of numbers from the repeat loop is: 132> 

```

Example 3: Infinity repeat loop

```

1. total<-0
2. number<-readline(prompt="please enter any integer value: ")
3. repeat{
4.  totaltotal=total+number
5.  numbernumber=number+1
6.  cat("sum is =",total)
7. }

```

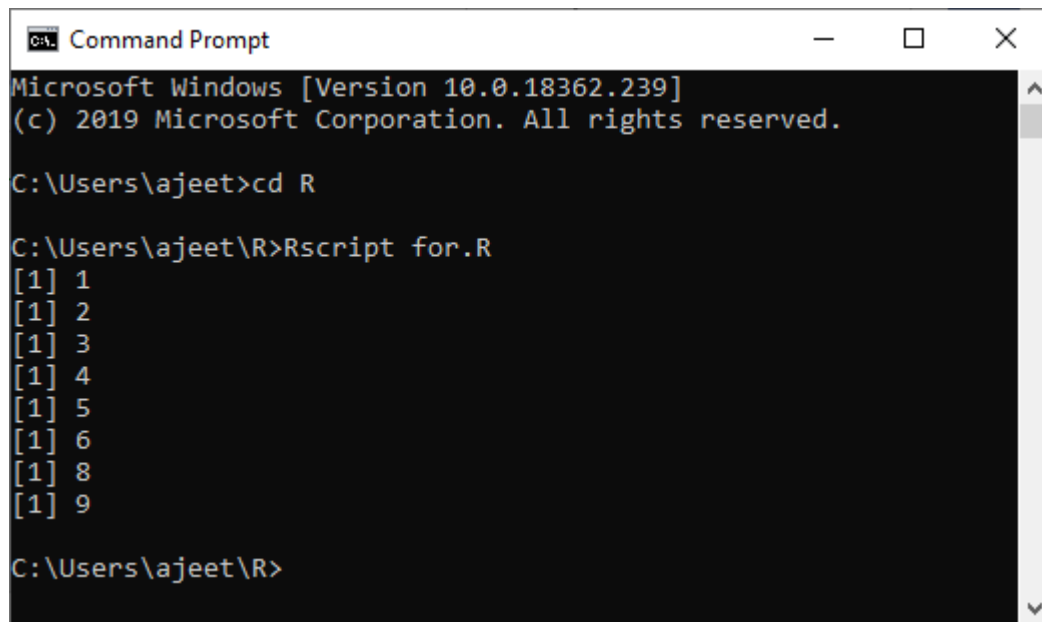
Output

```
Select Rterm (64-bit)
> total<-0
> number<-readline(prompt="please enter any integer value: ")
please enter any integer value: 21
> number<-as.integer(number)
> repeat{
+ total=total+number
+ number=number+1
+
+ cat("sum is =",total)
+ }
sum is = 21sum is = 43sum is = 66sum is = 90sum is = 115sum is = 141sum i
s = 168sum is = 196sum is = 225sum is = 255sum is = 286sum is = 318sum is
= 351sum is = 385sum is = 420sum is = 456sum is = 493sum is = 531sum is
= 570sum is = 610sum is = 651sum is = 693sum is = 736sum is = 780sum is =
825sum is = 871sum is = 918sum is = 966sum is = 1015sum is = 1065sum is
= 1116sum is = 1168sum is = 1221sum is = 1275sum is = 1330sum is = 1386su
m is = 1443sum is = 1501sum is = 1560sum is = 1620sum is = 1681sum is = 1
743sum is = 1806sum is = 1870sum is = 1935sum is = 2001sum is = 2068sum i
```

Example 4: repeat loop with next

1. `a <- 1`
2. `repeat {`
3. `if(a == 10)`
4. `break`
5. `if(a == 7){`
6. `a=a+1`
7. `next`
8. `}`
9. `print(a)`
10. `a <- a+1`
11. `}`

Output

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The window content shows the following text: "Microsoft Windows [Version 10.0.18362.239] (c) 2019 Microsoft Corporation. All rights reserved. C:\Users\ajeet>cd R C:\Users\ajeet\R>Rscript for.R [1] 1 [1] 2 [1] 3 [1] 4 [1] 5 [1] 6 [1] 8 [1] 9 C:\Users\ajeet\R>". The output consists of nine lines, each starting with "[1]" followed by a number. The numbers are 1, 2, 3, 4, 5, 6, 8, and 9. There is a jump in the sequence from 6 to 8. The window has a scrollbar on the right side.

```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R
C:\Users\ajeet\R>Rscript for.R
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 8
[1] 9
C:\Users\ajeet\R>
```

Example 5:

1. `terms<-readline(prompt="How many terms do you want ?")`
2. `terms<-as.integer(terms)`
3. `i<-1`
4. `repeat{`
5. `print(paste("The cube of number",i,"is =",(i*i*i)))`
6. `if(i==terms)`
7. `break`
8. `i<-i+1`
9. `}`

Output


```
Rterm (64-bit)
> terms<-readline(prompt="How many terms do you want ?")
How many terms do you want ?6
> terms<-as.integer(terms)
> i<-1
> repeat{
+ print(paste("The cube of number",i,"is =",(i*i*i)))
+ if(i==terms)
+ break
+ i<-i+1
+ }
[1] "The cube of number 1 is = 1"
[1] "The cube of number 2 is = 8"
[1] "The cube of number 3 is = 27"
[1] "The cube of number 4 is = 64"
[1] "The cube of number 5 is = 125"
[1] "The cube of number 6 is = 216"
>
```

R while loop

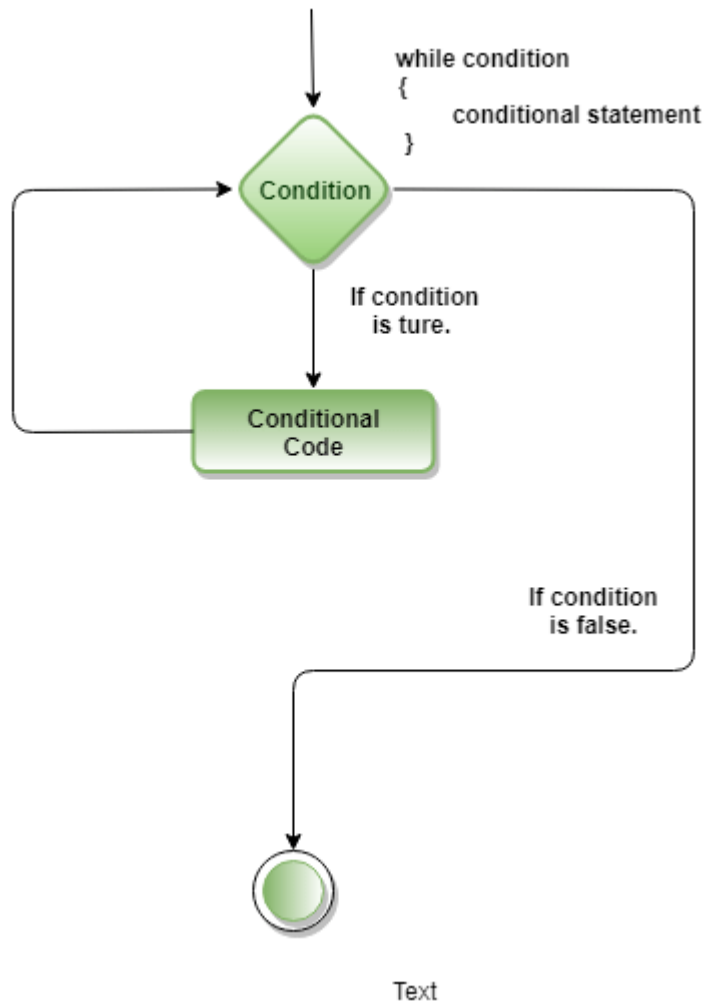
A while loop is a type of control flow statements which is used to iterate a block of code several numbers of times. The while loop terminates when the value of the Boolean expression will be false.

In while loop, firstly the condition will be checked and then after the body of the statement will execute. In this statement, the condition will be checked $n+1$ time, rather than n times.

The basic syntax of while loop is as follows:

1. while (test_expression) {
2. statement
3. }

Flowchart



Example 1:

1. `v <- c("Hello","while loop","example")`
2. `cnt <- 2`
3. `while (cnt < 7) {`
4. `print(v)`
5. `cnt = cnt + 1`
6. `}`

Output

```
Command Prompt
C:\Users\ajeet\R>Rscript while.R
[1] "Hello"      "while loop" "example"
[1] "Hello"      "while loop" "example"
[1] "Hello"      "while loop" "example"
[1] "Hello"      "while loop" "example"
[1] "Hello"      "while loop" "example"
C:\Users\ajeet\R>
```

Example 2: Program to find the sum of the digits of the number.

1. `n<-readline(prompt="please enter any integer value: ")`
2. please enter any integer value: 12367906
3. `n <- as.integer(n)`
4. `sum<-0`
5. `while(n!=0){`
6. `sumsum=sum+(n%%10)`
7. `n=as.integer(n/10)`
8. `}`
9. `cat("sum of the digits of the numbers is=",sum)`

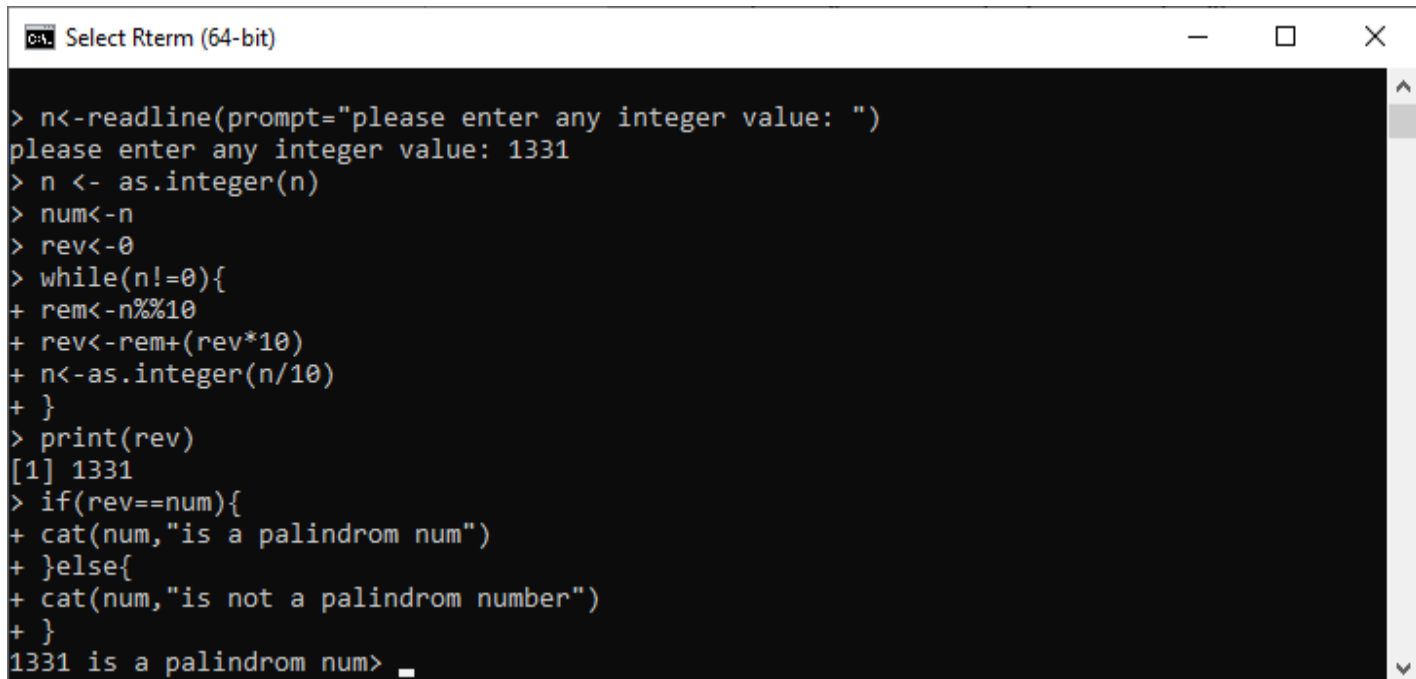
Output

```
Rterm (64-bit)
please enter any integer value: 12367906
> n <- as.integer(n)
> sum<-0
> while(n!=0){
+ sum=sum+(n%%10)
+ n=as.integer(n/10)
+ }
> cat("sum of the digits of the numbers is=",sum)
sum of the digits of the numbers is= 34>
```

Example 3: Program to check a number is palindrome or not.

```
1. n <- readline(prompt="Enter a four digit number please: ")
2. n <- as.integer(n)
3. num<-n
4. rev<-0
5. while(n!=0){
6.   rem<-n%%10
7.   rev<-rem+(rev*10)
8.   n<-as.integer(n/10)
9. }
10. print(rev)
11. if(rev==num){
12.   cat(num,"is a palindrome num")
13. }else{
14.   cat(num,"is not a palindrome number")
15. }
```

Output

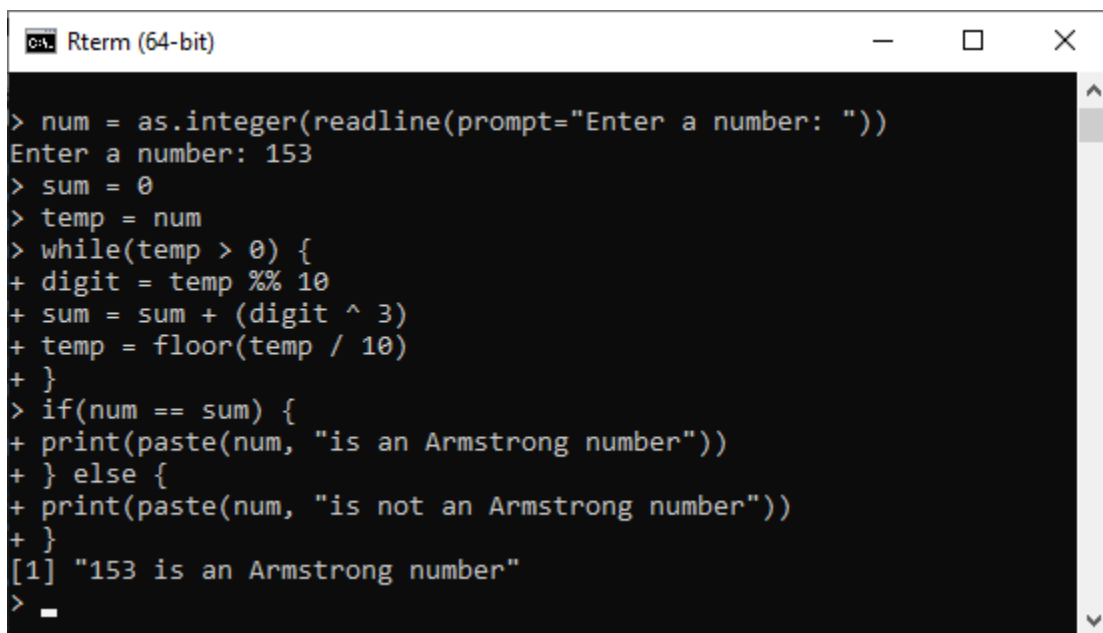
A screenshot of an R console window titled "Select Rterm (64-bit)". The window has a black background with white text. The R prompt is ">". The user enters the code from the example, and the output shows the number 1331 being entered, the reversed number 1331 being printed, and the message "1331 is a palindrom num" being displayed. The cursor is at the end of the last line of output.

```
> n<-readline(prompt="please enter any integer value: ")
please enter any integer value: 1331
> n <- as.integer(n)
> num<-n
> rev<-0
> while(n!=0){
+ rem<-n%%10
+ rev<-rem+(rev*10)
+ n<-as.integer(n/10)
+ }
> print(rev)
[1] 1331
> if(rev==num){
+ cat(num,"is a palindrom num")
+ }else{
+ cat(num,"is not a palindrom number")
+ }
1331 is a palindrom num> _
```

Example 4: Program to check a number is Armstrong or not.

```
1. num = as.integer(readline(prompt="Enter a number: "))
2. sum = 0
3. temp = num
4. while(temp > 0) {
5.     digit = temp %% 10
6.     sumsum = sum + (digit ^ 3)
7.     temp = floor(temp / 10)
8. }
9. if(num == sum) {
10.    print(paste(num, "is an Armstrong number"))
11.} else {
12.    print(paste(num, "is not an Armstrong number"))
13.}
```

Output

A screenshot of an Rterm window titled "Rterm (64-bit)". The window has a black background with white text. The code from the previous block is pasted into the terminal. The user has entered "153" in response to the prompt "Enter a number: ". The output shows the calculation of the sum of the cubes of the digits (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153) and the final result: "[1] \"153 is an Armstrong number\"".

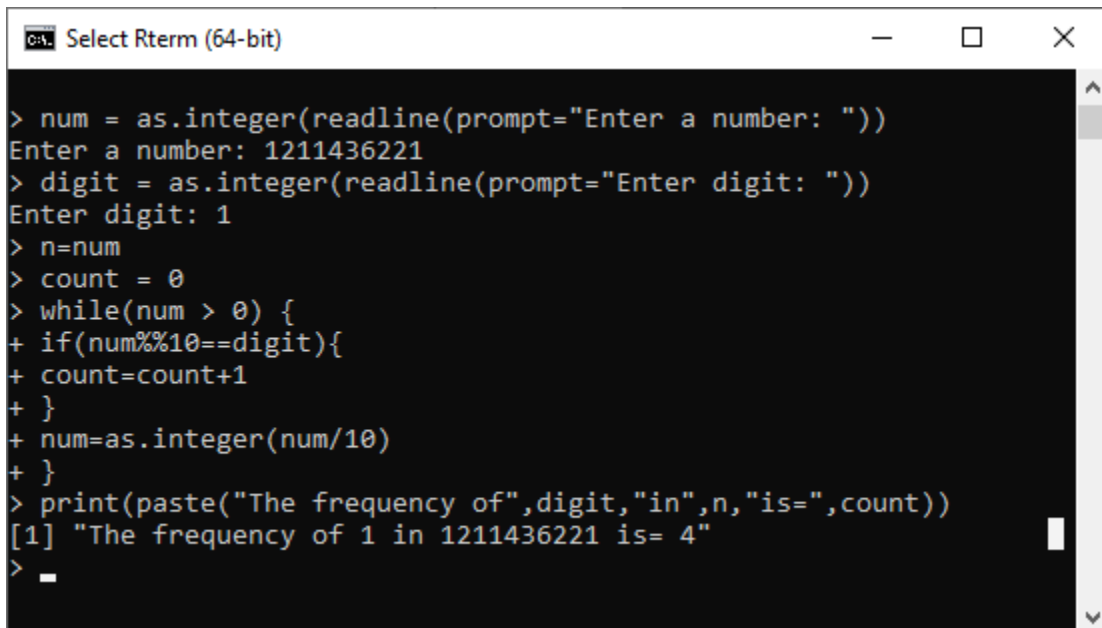
```
> num = as.integer(readline(prompt="Enter a number: "))
Enter a number: 153
> sum = 0
> temp = num
> while(temp > 0) {
+ digit = temp %% 10
+ sum = sum + (digit ^ 3)
+ temp = floor(temp / 10)
+ }
> if(num == sum) {
+ print(paste(num, "is an Armstrong number"))
+ } else {
+ print(paste(num, "is not an Armstrong number"))
+ }
[1] "153 is an Armstrong number"
> _
```

Example 5: program to find the frequency of a digit in the number.

```
1. num = as.integer(readline(prompt="Enter a number: "))
```

```
2. digit = as.integer(readline(prompt="Enter digit: "))
3. n=num
4. count = 0
5. while(num > 0) {
6.     if(num%%10==digit){
7.         countcount=count+1
8.     }
9.     num=as.integer(num/10)
10.}
11. print(paste("The frequency of",digit,"in",n,"is=",count))
```

Output



```
> num = as.integer(readline(prompt="Enter a number: "))
Enter a number: 1211436221
> digit = as.integer(readline(prompt="Enter digit: "))
Enter digit: 1
> n=num
> count = 0
> while(num > 0) {
+ if(num%%10==digit){
+ count=count+1
+ }
+ num=as.integer(num/10)
+ }
> print(paste("The frequency of",digit,"in",n,"is=",count))
[1] "The frequency of 1 in 1211436221 is= 4"
> _
```

R Functions

A set of statements which are organized together to perform a specific task is known as a function. R provides a series of in-built functions, and it allows the user to create their own functions. Functions are used to perform tasks in the modular approach.

Functions are used to avoid repeating the same task and to reduce complexity. To understand and maintain our code, we logically break it into smaller parts using the function. A function should be

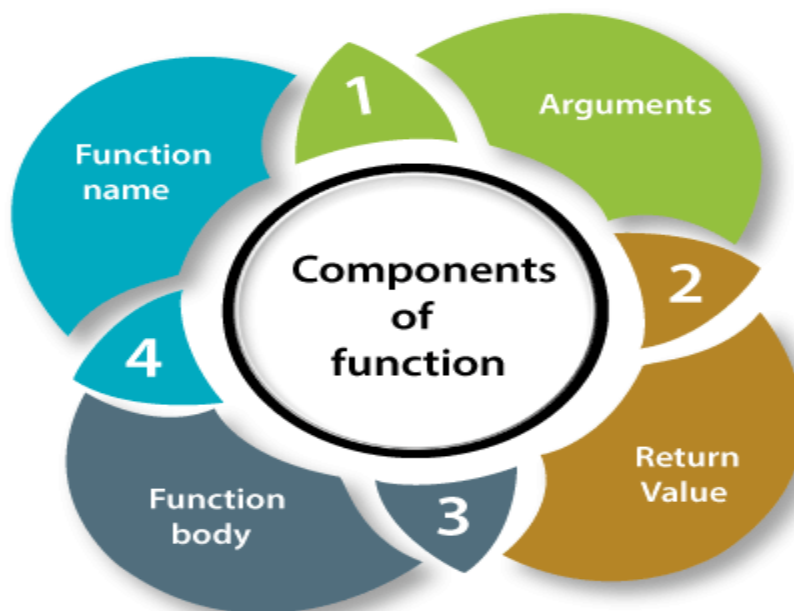
1. Written to carry out a specified task.
2. May or may not have arguments
3. Contain a body in which our code is written.
4. May or may not return one or more output values.

"An R function is created by using the keyword function." There is the following syntax of R function:

1. `func_name <- function(arg_1, arg_2, ...) {`
2. Function body
3. `}`

Components of Functions

There are four components of function, which are as follows:



Function Name

The function name is the actual name of the function. In R, the function is stored as an object with its name.

Arguments

In R, an argument is a placeholder. In function, arguments are optional means a function may or may not contain arguments, and these arguments can have default values also. We pass a value to the argument when a function is invoked.

Function Body

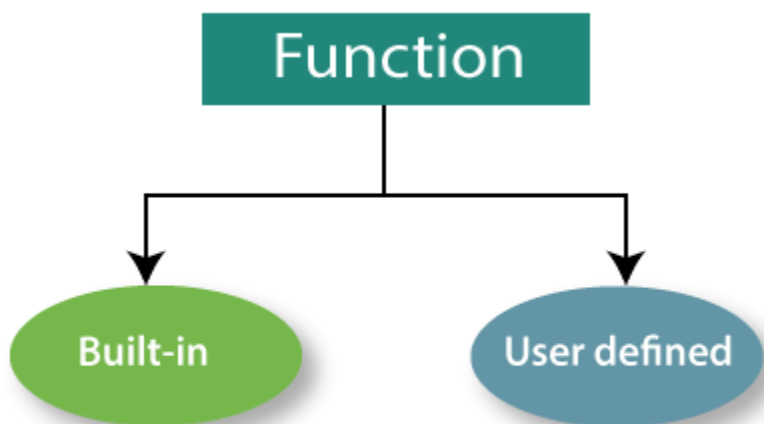
The function body contains a set of statements which defines what the function does.

Return value

It is the last expression in the function body which is to be evaluated.

Function Types

Similar to the other languages, R also has two types of function, i.e. **Built-in Function** and **User-defined Function**. In R, there are lots of built-in functions which we can directly call in the program without defining them. R also allows us to create our own functions.

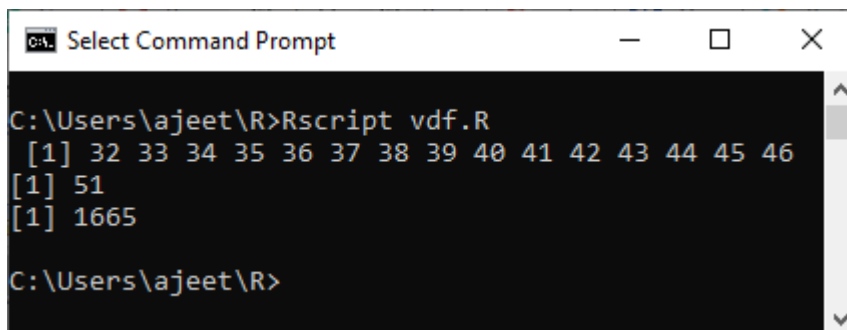


Built-in function

The functions which are already created or defined in the programming framework are known as built-in functions. User doesn't need to create these types of functions, and these functions are built into an application. End-users can access these functions by simply calling it. R have different types of built-in functions such as seq(), mean(), max(), and sum(x) etc.

1. # Creating sequence of numbers from 32 to 46.
2. print(seq(32,46))
- 3.
4. # Finding the mean of numbers from 22 to 80.
5. print(mean(22:80))
- 6.
7. # Finding the sum of numbers from 41 to 70.
8. print(sum(41:70))

Output:



```
C:\Users\ajeet\R>Rscript vdf.R
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
[1] 51
[1] 1665
C:\Users\ajeet\R>
```

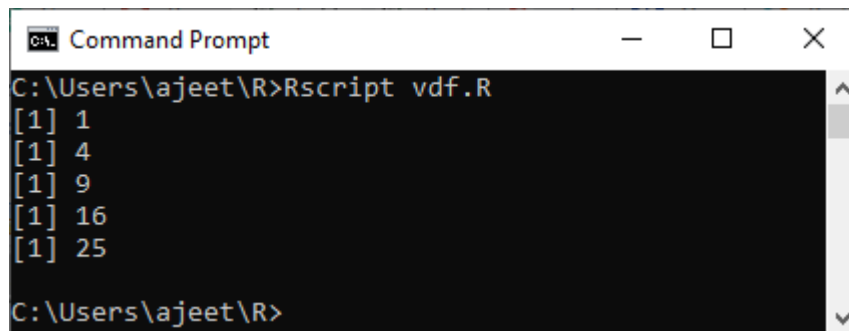
User-defined function

R allows us to create our own function in our program. A user defines a user-defined function to fulfill the requirement of user. Once these functions are created, we can use these functions like in-built function.

1. # Creating a function without an argument.
2. new.function <- function() {
3. for(i in 1:5) {
4. print(i^2)

5. }
6. }
- 7.
8. `new.function()`

Output:



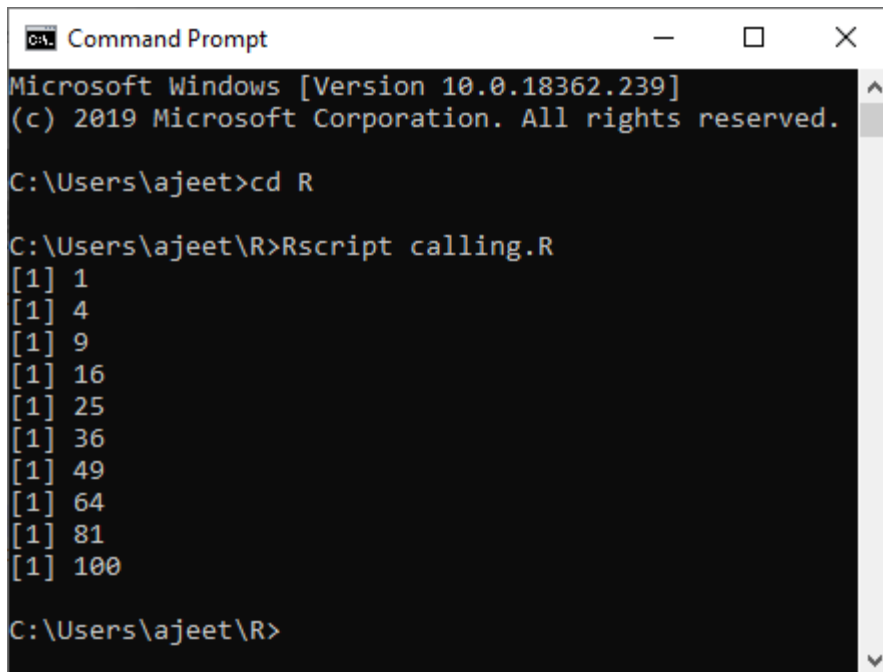
```
Command Prompt
C:\Users\ajeet\R>Rscript vdf.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
C:\Users\ajeet\R>
```

Function calling with an argument

We can easily call a function by passing an appropriate argument in the function. Let see an example to see how a function is called.

1. # Creating a function to print squares of numbers in sequence.
2. `new.function <- function(a) {`
3. `for(i in 1:a) {`
4. `b <- i^2`
5. `print(b)`
6. `}`
- 7.
8. # Calling the function `new.function` supplying `10` as an argument.
9. `new.function(10)`

Output:

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The text inside shows the Windows version and copyright information, followed by the user navigating to the R directory and running an R script. The output of the script is a list of squares from 1 to 100.

```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript calling.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100

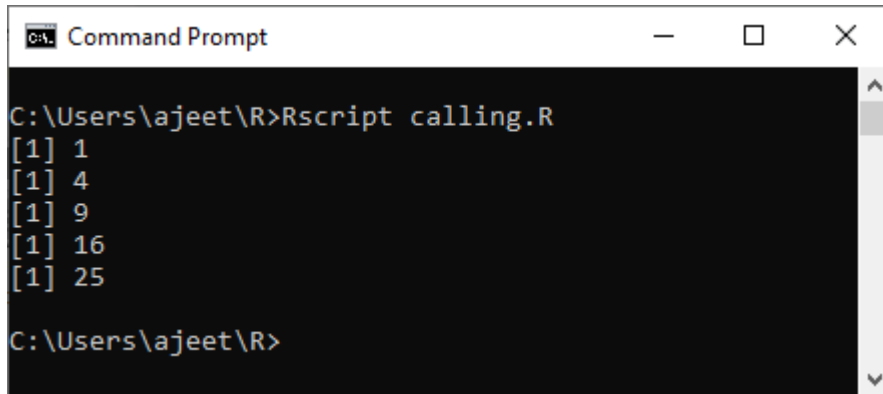
C:\Users\ajeet\R>
```

Function calling with no argument

In R, we can call a function without an argument in the following way

1. # Creating a function to print squares of numbers in sequence.
2. `new.function <- function() {`
3. `for(i in 1:5) {`
4. `a <- i^2`
5. `print(a)`
6. `}`
7. `}`
- 8.
9. # Calling the function `new.function` with no argument.
10. `new.function()`

Output:



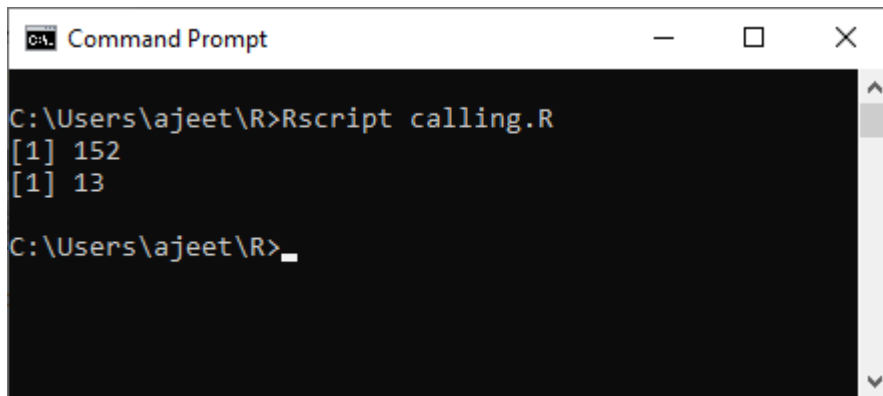
```
Command Prompt
C:\Users\ajet\R>Rscript calling.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
C:\Users\ajet\R>
```

Function calling with Argument Values

We can supply the arguments to a function call in the same sequence as defined in the function or can supply in a different sequence but assigned them to the names of the arguments.

1. # Creating a function with arguments.
2. `new.function <- function(x,y,z) {`
3. `result <- x * y + z`
4. `print(result)`
5. `}`
- 6.
7. # Calling the function by position of arguments.
8. `new.function(11,13,9)`
- 9.
10. # Calling the function by names of the arguments.
11. `new.function(x = 2, y = 5, z = 3)`

Output:



```
Command Prompt
C:\Users\ajeet\R>Rscript calling.R
[1] 152
[1] 13
C:\Users\ajeet\R>_
```

Function calling with default arguments

To get the default result, we assign the value to the arguments in the function definition, and then we call the function without supplying argument. If we pass any argument in the function call, then it will get replaced with the default value of the argument in the function definition.

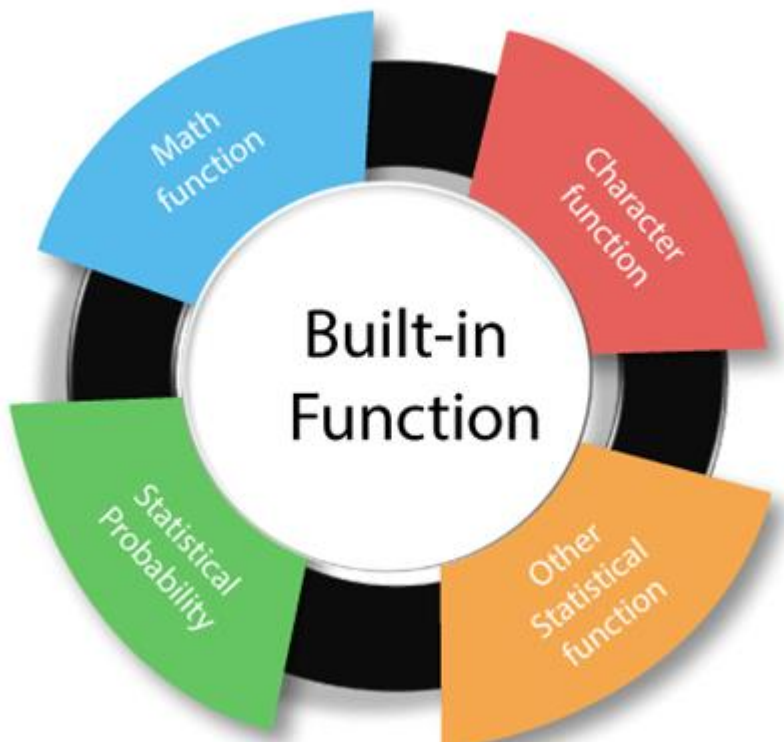
1. # Creating a function with arguments.
2. **new**.function <- function(x = 11, y = 24) {
3. result <- x * y
4. print(result)
5. }
- 6.
7. # Calling the function without giving any argument.
8. **new**.function()
- 9.
10. # Calling the function with giving **new** values of the argument.
11. **new**.function(4,6)

Output:

```
Command Prompt
C:\Users\ajet\R>Rscript calling.R
[1] 264
[1] 24
C:\Users\ajet\R>_
```

R Built-in Functions

The functions which are already created or defined in the programming framework are known as a built-in function. R has a rich set of functions that can be used to perform almost every task for the user. These built-in functions are divided into the following categories based on their functionality.



Math Functions

S. No	Function	Description	Example
1.	abs(x)	It returns the absolute value of input x.	<pre>x<- -4 print(abs(x))</pre> Output <pre>[1] 4</pre>
2.	sqrt(x)	It returns the square root of input x.	<pre>x<- 4 print(sqrt(x))</pre> Output <pre>[1] 2</pre>
3.	ceiling(x)	It returns the smallest integer which is larger than or equal to x.	<pre>x<- 4.5 print(ceiling(x))</pre> Output <pre>[1] 5</pre>
4.	floor(x)	It returns the largest integer, which is smaller than or equal to x.	<pre>x<- 2.5 print(floor(x))</pre> Output <pre>[1] 2</pre>
5.	trunc(x)	It returns the truncate value of input x.	<pre>x<- c(1.2,2.5,8.1) print(trunc(x))</pre> Output <pre>[1] 1 2 8</pre>
6.	round(x, digits=n)	It returns round value of input x.	<pre>x<- -4 print(abs(x))</pre> Output <pre>4</pre>
7.	cos(x), sin(x), tan(x)	It returns cos(x), sin(x) value of input x.	<pre>x<- 4 print(cos(x)) print(sin(x)) print(tan(x))</pre> Output <pre>[1] -0.6536436 [2] -0.7568025 [3] 1.157821</pre>
8.	log(x)	It returns natural logarithm of input x.	<pre>x<- 4 print(log(x))</pre> Output <pre>[1] 1.386294</pre>

9.	log10(x)	It returns common logarithm of input x.	<pre>x<- 4 print(log10(x))</pre> Output <pre>[1] 0.60206</pre>
10.	exp(x)	It returns exponent.	<pre>x<- 4 print(exp(x))</pre> Output <pre>[1] 54.59815</pre>

R provides the various mathematical functions to perform the mathematical calculation. These mathematical functions are very helpful to find absolute value, square value and much more calculations. In R, there are the following functions which are used:

String Function

R provides various string functions to perform tasks. These string functions allow us to extract sub string from string, search pattern etc. There are the following string functions in R:

S. No	Function	Description	Example
1.	substr(x, start=n1, stop=n2)	It is used to extract substrings in a character vector.	<pre>a <- "987654321" substr(a, 3, 3)</pre> Output <pre>[1] "3"</pre>
2.	grep(pattern, x, ignore.case=FALSE, fixed=FALSE)	It searches for pattern in x.	<pre>st1 <- c('abcd', 'bdcd', 'abcdabcd') pattern<- '^abc' print(grep(pattern, st1))</pre> Output <pre>[1] 1 3</pre>
3.	sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)	It finds pattern in x and replaces it with replacement (new) text.	<pre>st1<- "England is beautiful but no the part of EU" sub("England", "UK", st1)</pre> Output <pre>[1] "UK is beautiful but not a part of EU"</pre>

4.	<code>paste(..., sep="")</code>	It concatenates strings after using sep string to separate them.	<code>paste('one',2,'three',4,'five')</code> Output [1] one 2 three 4 five
5.	<code>strsplit(x, split)</code>	It splits the elements of character vector x at split point.	<code>a<-"Split all the character"</code> <code>print(strsplit(a, ""))</code> Output [[1]] [1] "split" "all" "the" "character"
6.	<code>tolower(x)</code>	It is used to convert the string into lower case.	<code>st1<- "shuBHAm"</code> <code>print(tolower(st1))</code> Output [1] shubham
7.	<code>toupper(x)</code>	It is used to convert the string into upper case.	<code>st1<- "shuBHAm"</code> <code>print(toupper(st1))</code> Output [1] SHUBHAM

Statistical Probability Functions

R provides various statistical probability functions to perform statistical task. These statistical functions are very helpful to find normal density, normal quantile and many more calculation. In R, there are following functions which are used:

Other Statistical Function

Apart from the functions mentioned above, there are some other useful functions which helps for statistical purpose. There are the following functions:

S. No	Function	Description	Example
1.	<code>dnorm(x, m=0, sd=1, log=False)</code>	It is used to find the height of the probability distribution at each point to a given mean and standard deviation	<pre>a <- seq(-7, 7, by=0.1) b <- dnorm(a, mean=2.5, sd=0.5) png(file="dnorm.png") plot(x,y) dev.off()</pre>
2.	<code>pnorm(q, m=0, sd=1, lower.tail=TRUE, log.p=FALSE)</code>	it is used to find the probability of a normally distributed random numbers which are less than the value of a given number.	<pre>a <- seq(-7, 7, by=0.2) b <- dnorm(a, mean=2.5, sd=2) png(file="pnorm.png") plot(x,y) dev.off()</pre>
3.	<code>qnorm(p, m=0, sd=1)</code>	It is used to find a number whose cumulative value matches with the probability value.	<pre>a <- seq(1, 2, by=0.02) b <- qnorm(a, mean=2.5, sd=0.5) png(file="qnorm.png") plot(x,y) dev.off()</pre>
4.	<code>rnorm(n, m=0, sd=1)</code>	It is used to generate random numbers whose distribution is normal.	<pre>y <- rnorm(40) png(file="rnorm.png") hist(y, main="Normal Distribution") dev.off()</pre>
5.	<code>dbinom(x, size, prob)</code>	It is used to find the probability density distribution at each point.	<pre>a<-seq(0, 40, by=1) b<- dbinom(a, 40, 0.5) png(file="pnorm.png") plot(x,y) dev.off()</pre>
6.	<code>pbinom(q, size, prob)</code>	It is used to find the cumulative probability (a single value representing the probability) of an event.	<pre>a <- pbinom(25, 40,0.5) print(a) Output [1] 0.9596548</pre>
7.	<code>qbinom(p, size, prob)</code>	It is used to find a number whose cumulative value matches the probability value.	<pre>a <- qbinom(0.25, 40,0.5) print(a) Output [1] 18</pre>

8.	<code>rbinom(n, size, prob)</code>	It is used to generate required number of random values of a given probability from a given sample.	<pre>a <- rbinom(6, 140, 0.4) print(a)</pre> Output <pre>[1] 55 61 46 56 58 49</pre>
9.	<code>dpois(x, lamba)</code>	it is the probability of x successes in a period when the expected number of events is lambda (λ)	<pre>dpois(a=2, lambda=3)+dpois(a=3, lambda=3)+dpois(z=4, labda=4)</pre> Output <pre>[1] 0.616115</pre>
10.	<code>ppois(q, lamba)</code>	It is a cumulative probability of less than or equal to q successes.	<pre>ppois(q=4, lambda=3, lower.tail=TRUE)-ppois(q=1, lambda=3, lower.tail=TRUE)</pre> Output <pre>[1] 0.6434504</pre>
11.	<code>rpois(n, lamba)</code>	It is used to generate random numbers from the poisson distribution.	<pre>rpois(10, 10)</pre> <pre>[1] 6 10 11 3 10 7 7 8 14 12</pre>
12.	<code>dunif(x, min=0, max=1)</code>	This function provide information about the uniform distribution on the interval from min to max. It gives the density.	<code>dunif(x, min=0, max=1, log=FALSE)</code>
13.	<code>punif(q, min=0, max=1)</code>	It gives the distributed function	<pre>punif(q, min=0, max=1, lower.tail=TRUE, log.p=FALSE)</pre>
14.	<code>qunif(p, min=0, max=1)</code>	It gives the quantile function.	<pre>qunif(p, min=0, max=1, lower.tail=TRUE, log.p=FALSE)</pre>
15.	<code>runif(x, min=0, max=1)</code>	It generates random deviates.	<code>runif(x, min=0, max=1)</code>

S. No	Function	Description	Example
1.	mean(x, trim=0, na.rm=FALSE)	It is used to find the mean for x object	<pre>a<-c(0:10, 40) xm<-mean(a) print(xm)</pre> Output <pre>[1] 7.916667</pre>
2.	sd(x)	It returns standard deviation of an object.	<pre>a<-c(0:10, 40) xm<-sd(a) print(xm)</pre> Output <pre>[1] 10.58694</pre>
3.	median(x)	It returns median.	<pre>a<-c(0:10, 40) xm<-median(a) print(xm)</pre> Output <pre>[1] 5.5</pre>
4.	quantile(x, probs)	It returns quantile where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0, 1]	
5.	range(x)	It returns range.	<pre>a<-c(0:10, 40) xm<-range(a) print(xm)</pre> Output <pre>[1] 0 40</pre>
6.	sum(x)	It returns sum.	<pre>a<-c(0:10, 40) xm<-sum(a) print(xm)</pre> Output <pre>[1] 95</pre>

7.	diff(x, lag=1)	It returns differences with lag indicating which lag to use.	<pre>a<-c(0:10, 40) xm<-diff(a) print(xm)</pre> Output <pre>[1] 1 1 1 1 1 1 1 1 1 1 30</pre>
8.	min(x)	It returns minimum value.	<pre>a<-c(0:10, 40) xm<-min(a) print(xm)</pre> Output <pre>[1] 0</pre>
9.	max(x)	It returns maximum value	<pre>a<-c(0:10, 40) xm<-max(a) print(xm)</pre> Output <pre>[1] 40</pre>
10.	scale(x, center=TRUE, scale=TRUE)	Column center or standardize a matrix.	<pre>a <- matrix(1:9,3,3) scale(x)</pre> Output <pre>[,1] [1,] -0.747776547 [2,] -0.653320562 [3,] -0.558864577 [4,] -0.464408592 [5,] -0.369952608 [6,] -0.275496623 [7,] -0.181040638 [8,] -0.086584653 [9,] 0.007871332 [10,] 0.102327317 [11,] 0.196783302 [12,] 3.030462849 attr(,"scaled:center") [1] 7.916667 attr(,"scaled:scale") [1] 10.58694</pre>