## Advance R

## DataFrame in R :

A data frame is a collection of 2 vectors means 2-dimensional array-like structure or a tabular form in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a type of the list in which each component has equal length.

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

In a simple way, it is a list of equal length vectors. A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.

DataFrame is the most important data structure in the field of Data Analytics and Data Science.

Data can be collected from various sources like Database, Cloud, CSV, etc, but is ultimately stored in a data frame.

There are following characteristics of a data frame.

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

**Creating data frames**

Data frame can be created using data.frame() function

```
v1 <- 1:5
# Letter is an inbuilt list
letters

#output
'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w'
'x' 'y' 'z'
```

```
v2 <- letters[1:5]
df <- data.frame(v1,v2)
df
```

output:

```
v1 v2
 1  a
2  b
3  c
4  d
5  e
```

***Creating dataframe with column name***

```
c1 <- c('Tendulkar','Kohli','Dohni','Bumrah','Chahal')
c2 <- c(10000,7100,5800,890,870)
c3 <- c(11,0,0,370,420)
cricket <- data.frame(players=c1,runs=c2,wickets=c3)
cricket
```

```
output :
Players       runs    wickets
Tendulkar  10000    11
Kohli          7100     0
Dohni          5800     0
```

```
Bumrah      890   370
Chahal      870   420
```

**Inbuilt data frames**

View inbuilt data frames using data() function

```
data()
state.x77
```

**Understanding data**

Use head() & tail() function to see first 6 and last 6 rows respectively

```
head(state.x77)
```

```
tail(state.x77)
```

Use summary() and str() function to get basic description of data like mean,median,quartiles etc.

```
summary(state.x77)
```

Output:

```
Population Income Illiteracy Life Exp
Min. : 365 Min. :3098 Min. :0.500 Min. :67.96
1st Qu.: 1080 1st Qu.:3993 1st Qu.:0.625 1st Qu.:70.12
Median : 2838 Median :4519 Median :0.950 Median :70.67
Mean : 4246 Mean :4436 Mean :1.170 Mean :70.88
3rd Qu.: 4968 3rd Qu.:4814 3rd Qu.:1.575 3rd Qu.:71.89
Max. :21198 Max. :6315 Max. :2.800 Max. :73.60
Murder HS Grad Frost Area
Min. : 1.400 Min. :37.80 Min. : 0.00 Min. : 1049
1st Qu.: 4.350 1st Qu.:48.05 1st Qu.: 66.25 1st Qu.: 36985
Median : 6.850 Median :53.25 Median :114.50 Median : 54277
Mean : 7.378 Mean :53.11 Mean :104.46 Mean : 70736
3rd Qu.:10.675 3rd Qu.:59.15 3rd Qu.:139.75 3rd Qu.: 81163
Max. :15.100 Max. :67.30   Max. :188.00 Max. :566432
```

```
str(state.x77)
```

Output :

```
num [1:50, 1:8] 3615 365 2212 2110 21198 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" ...
..$ : chr [1:8] "Population" "Income" "Illiteracy" "Life Exp" ...
```

```
nrow(state.x77)

#output
50
```

```
ncol(state.x77)

#output
8
```

```
rownames(state.x77)
```

```
colnames(state.x77)

#output
'Population' 'Income' 'Illiteracy' 'Life Exp' 'Murder' 'HS Grad' 'Frost' 'Area'
```

**Indexing and slicing**

**1) Selecting cells**

*Creating dataframe with column name*

```
c1 <- c('Tendulkar','Kohli','Dohni','Bumrah','Chahal')
c2 <- c(10000,7100,5800,890,870)
c3 <- c(11,0,0,370,420)
cricket <- data.frame(players=c1,runs=c2,wickets=c3)
cricket
```

```
Cricket[1,2]

#output
10000
```

```
Cricket[1:3,1:2]

#output
players       runs
Tendulkar 10000
Kohli          7100
Dohni          5800
```

```
cricket[c(1,4),c(1,3)]

#output
    players    wickets
1 Tendulkar    11
4 Bumrah    370
```

```
cricket[1:3,'wickets']

#output
11 0 0
```

**2) Selecting rows**

```
cricket[1,]

#output
players       runs     wickets
Tendulkar 10000     11
```

```
Cricket[1:3,]

#output
players       runs    wickets
Tendulkar 10000    11
Kohli          7100     0
```

```
Dohni         5800   0
```

```
cricket[c(1,3),]

#output
     players       runs      wickets
1 Tendulkar 10000            11
3 Dohni        5800             0
```

### 3) Selecting columns

```
Cricket[,3]

#output
11 0 0 370 420
```

```
Cricket[,1:2]

#output
players        runs
Tendulkar    10000
Kohli             7100
Dohni           5800
Bumrah          890
Chahal           870
```

```
cricket[,c(1,3)]
#output
players       wickets
Tendulkar     11
Kohli            0
Dohni           0
Bumrah     370
Chahal       420
```

```
Cricket[,'runs']

#output
10000 7100 5800 890 870
```

```
cricket[,c('players','wickets')]

#output
players       wickets
Tendulkar     11
Kohli          0
Dohni          0
Bumrah       370
Chahal       420
```

```
cricket[['wickets']]

#output
11 0 0 370 420
```

```
cricket$players

#output
Tendulkar Kohli Dohni Bumrah Chahal
```

**Conditional selection with subset() function**

```
subset(cricket,subset=runs>5000)

#output
Players       runs       wickets
Tendulkar   10000        11
Kohli          7100         0
Dohni         5800         0
```

```
subset(cricket,subset=wickets>300)

#output
     players    runs   wickets
4   Bumrah     890      370
5   Chahal     870      420
```

**Ordering dataframe**

Ordering is done with order() function

```
order(cricket['runs'])

#output
5 4 3 2 1
```

```
runs.order <- order(cricket['runs'])
cricket[runs.order,]

#output
   players     runs    wickets
5 Chahal       870     420
4 Bumrah      890      370
3 Dohni       5800      0
2 Kohli        7100      0
1 Tendulkar  10000    11
```

```
runs.order <- order(-cricket['runs'])
cricket[runs.order,]

#output
players      runs     wickets
Tendulkar   10000   11
Kohli            7100     0
Dohni          5800    0
Bumrah         890  370
Chahal          870  420
```

**Renaming Names**

```
mat <- matrix(1:20,nrow=5)
print(mat)
```

```
df <- data.frame(mat)
Df

#output
X1 X2 X3 X4
1    6   11  16
```

```
2   7 12 17
3  8  13 18
4  9  14 19
5 10 15 20
```

```
# renaming single column
colnames(df)[1] <- 'index'
df

#output
index   X2   X3   X4
1          6     11   16
2          7     12   17
3          8     13 18
4          9     14 19
5          10     15 20
```

```
# renaming multiple column
colnames(df) <- c('A','B','C','D')
df

#output
A   B   C  D
1   6  11 16
2   7 12 17
3  8  13 18
4  9  14 19
5 10 15 20
```

*Adding new rows and columns*

```
# Adding row
df2 <- data.frame(A=20,B=34,C=67,D=56)
df <- rbind(df,df2)
df

#output
A    B   C  D
1    6  11  16
2    7 12   17
3   8  13  18
```

```
4   9  14  19
5  10  15  20
20 34 67  56
```

```
# adding columns using replicate function
df$E <- rep(NA,nrow(df))
df

#output
A    B   C  D    E
1    6  11  16   NA
2    7 12   17   NA
3    8  13  18   NA
4    9  14  19   NA
5   10  15  20   NA
20 34 67  56    NA
```

```
df$F <- 10:15
df

#output
A    B   C  D    E    F
1    6  11  16   NA  10
2    7 12   17   NA  11
3    8  13  18   NA  12
4    9  14  19   NA  13
5   10  15  20   NA  14
20 34 67  56    NA  15
```

```
#copying another column
df$G = df$E
df

#output
A    B   C  D    E    F   G
1    6  11  16   NA  10  NA
2    7 12   17   NA  11  NA
3    8  13  18   NA  12 NA
4    9  14  19   NA  13 NA
5   10  15  20   NA  14 NA
20 34 67  56    NA  15 NA
```

```
# cbind function
V <- c(10,20,30,40,50,60)
df <- cbind(df,V)
df
#output
A    B   C  D    E    F   G    V
1    6  11  16   NA 10  NA  10
2    7 12   17   NA 11  NA   20
3    8  13  18   NA 12 NA   30
4    9  14  19   NA 13 NA   40
5   10 15 20   NA 14 NA   50
20 34 67 56   NA 15 NA   60
```

**Handling missing values**

check presence of missing values with any() and is.na() function

```
# Check in specific column
any(is.na(df$A))

#output
FALSE
```

```
any(is.na(df$E))

#output
TRUE
```

```
# Check entire dataframe
any(is.na(df))

#output
TRUE
```

```
#Replacing missing values
df$G[is.na(df$G)] <- 0
df

#output
A    B   C  D    E    F   G   V
```

```
1    6  11  16   NA 10  0  10
2    7 12   17   NA 11  0   20
3    8  13  18   NA 12 0   30
4    9  14  19   NA 13 0   40
5   10 15 20   NA 14 0   50
20 34 67 56   NA 15  0  60
```

```
df[is.na(df)] <- -1
df

#output
A    B   C  D    E    F  G  V
1    6  11  16  -1 10  0   10
2    7 12   17   -1 11  0   20
3    8  13  18   -1 12 0   30
4    9  14  19   -1 13 0   40
5   10 15 20   -1 14 0   50
20 34 67 56   -1 15  0  60
```

Note : Missing data must be replaced with mean, median or mode.

## List in R

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

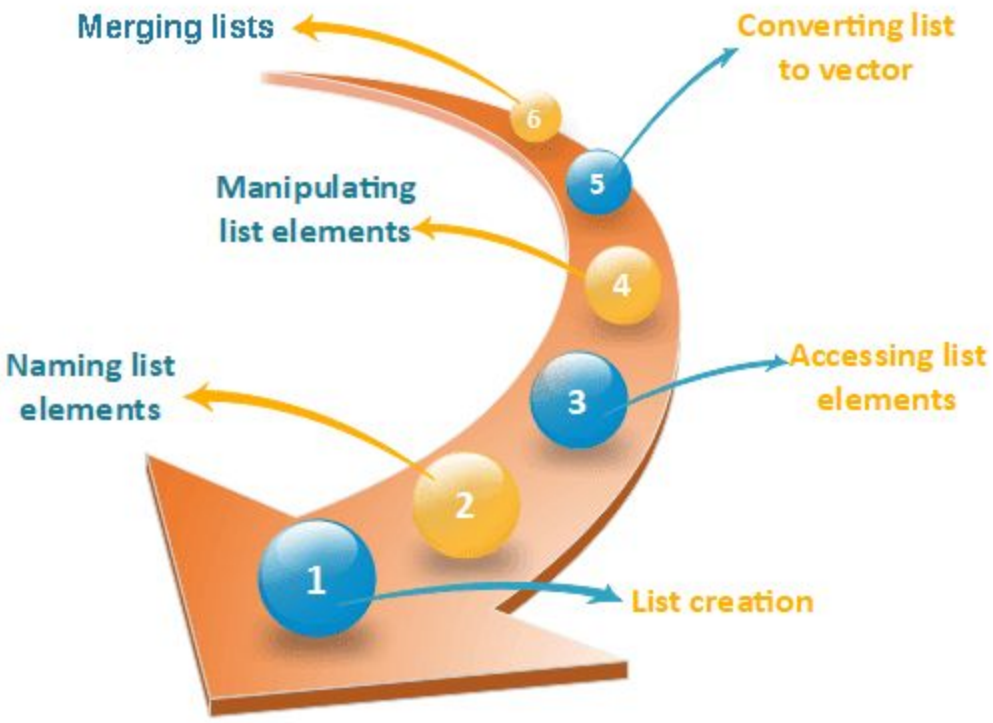
Example

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

Output:

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

# Lists in R programming




**Lists creation**

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

Syntax

list()

Example 1: Creating list with same data type

```
list_1<-list(1,2,3)
list_2<-list("Shubham","Arpita","Vaishali")
list_3<-list(c(1,2,3))
list_4<-list(TRUE,FALSE,TRUE)
list_1
list_2
list_3
list_4
```

Output:

```
[[1]]
```

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

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

[[1]]
[1] 1 2 3

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

Example 2: Creating the list with different data type

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

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

Output:

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

Giving a name to list elements

R provides a very easy way for accessing elements, i.e., by giving the name to each element of a list. By assigning names to the elements, we can access the element easily. There are only three steps to print the list data corresponding to the name:

Creating a list.

Assign a name to the list elements with the help of names() function.

Print the list data.

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

Example

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

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


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

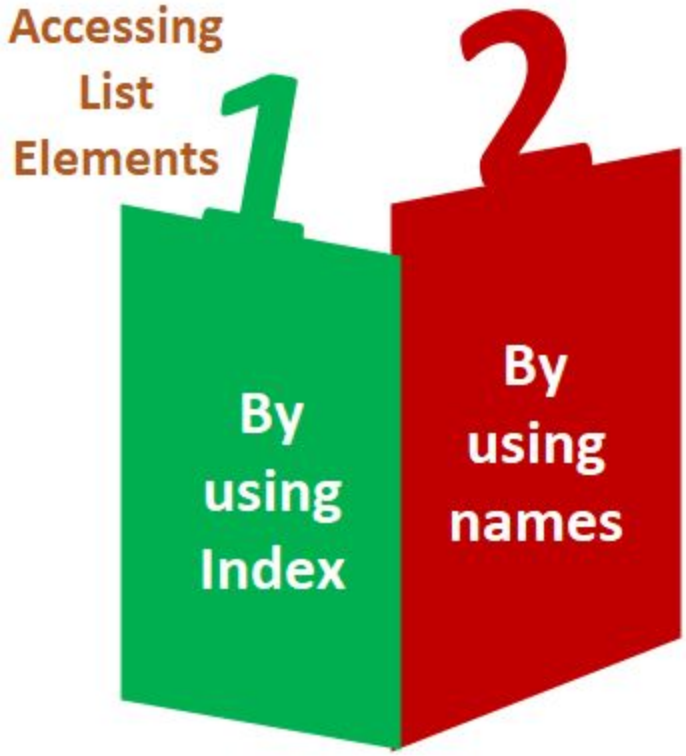
Output:

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

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

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

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

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

Accessing List Elements

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

with the named list.; we cannot access the elements of a list using names if the list is normal.




Let see an example of both methods to understand how they are used in the list to access elements.

Example 1:

**Accessing elements using index**

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

Output:

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

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

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

[1] "MCA"

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

Example 2:

**Accessing elements using names**

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

Output:

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

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

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

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

$Course
$Course[[1]]
[1] "BCA"
$Course[[2]]
[1] "MCA"
$Course[[3]]
[1] "B. tech."
```

**Manipulation of list elements**

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

Example

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

# Giving names to the elements in the list.
names(list_data) <- c("Student", "Marks", "Course")
  # Adding element at the end of the list.
list_data[4] <- "Moradabad"
print(list_data[4])
  # Removing the last element.
list_data[4] <- NULL
  # Printing the 4th Element.
print(list_data[4])
  # Updating the 3rd Element.
list_data[3] <- "Masters of computer applications"
print(list_data[3])
```

Output:

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

$<NA>
NULL

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

**Converting list to vector**

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

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

Example

```
# Creating lists.
list1 <- list(10:20)
```

```
print(list1)
 list2 <-list(5:14)
print(list2)
 # Converting the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
 print(v1)
print(v2)
 #adding the vectors
result <- v1+v2
print(result)
```
Output:
```
[[1]]
[1] 1 2 3 4 5

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

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

**Merging Lists**

R allows us to merge one or more lists into one list. Merging is done with the help of the list() function also. To merge the lists, we have to pass all the lists into list function as a parameter, and it returns a list which contains all the elements which are present in the lists. Let see an example to understand how the merging process is done.

Example
```
# Creating two lists.
Even_list <- list(2,4,6,8,10)
Odd_list <- list(1,3,5,7,9)
 # Merging the two lists.
merged.list <- list(Even_list,Odd_list)
 # Printing the merged list.
print(merged.list)
```
Output:
```
[[1]]
[[1]][[1]]
[1] 2

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

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

```
[1] 6

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

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


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

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

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

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

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

## R Built-in Features

seq() - creates a sequence

```
seq(1,10)

#output
1 2 3 4 5 6 7 8 9 10
```

```
# by argument for stepsize
seq(1,10,by=2)

#output
1 3 5 7 9
```

```
seq(1,10,2)

#output
```

```
1 3 5 7 9
```

sort() to sort a vector

```
v1 <- c(10,2,11,4,6,5,12,7)
sort(v1)

#output
2 4 5 6 7 10 11 12
```

```
sort(v1, decreasing=T)

#output
12 11 10 7 6 5 4 2
```

rev() to reverse elements of an object

```
rev(v1)
#output
7 12 5 6 4 11 2 10
```

str() describes structure of an object

```
str(v1)

#output
num [1:8] 10 2 11 4 6 5 12 7
```

```
mat <- matrix(1:15,nrow=3)
mat
#output
1 4 7 10  13
2 5 8 11 14
3 6 9 12 15
```

```
str(mat)

#output
int [1:3, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
```

append() merges objects together

```
append(v1,90)
```

```
#output
10 2 11 4 6 5 12 7 90
```

```
v2 <- c(1,2,3,4)
append(v1,v2)

#output
10 2 11 4 6 5 12 7 1 2 3 4
```

sample() gives a random value from a sequence

```
sample(1:20,1)

#output
17
```

```
sample(1:20,2)

#output
1  17
```

```
sample(v1,1)

#output
10
```

is.* checks class of an object

```
is.vector(v1)

#output
TRUE
```

```
is.list(v1)

#output
FALSE
```

as.* converts object type

```
as.list(v1)
```

```
as.matrix(v1)
```

*Regular expression*

grepl() returns logical output

```
text <- "I am data scientist"
grepl("data",text)

#output
TRUE
```

```
grepl("outlier",text)

#output
FALSE
```

grep() returns index

```
v <- c(10,20,30)
v

#output
10 20 30
```

```
grep(20,v)

#output
2
```

```
grep(40,v)

#output
# No output
```

**Date Time**

Sys.Date() get system's date

```
Sys.Date()

#output
2019-09-06
```

as.Date() convert string to date object

```
as.Date('1997-02-19')
#output
1997-02-19
```

**Formatting**

| Code | Value |
|---|---|
| %d | Day of the month |
| %m | Month |
| %b | Abbreviated month |
| %B | Full month |
| %y | 2 digit year |
| %Y | 4 digit year |

```
as.Date('28-November-1992',format="%d-%B-%Y")

#output
1992-11-28
```

```
as.Date('12-Jan-86',format="%d-%b-%y")

#output
1986-01-12
```

Sys.time() to get time

```
Sys.time()

#output
"2019-09-06 16:49:38 IST"
```

as.POSIXct to convert string to date and time

```
as.POSIXct("08:30:03",format="%H:%M:%S")

#output
"2019-09-06 08:30:03 IST"
```

```
as.POSIXct("28-November-1992 09:25:04",format="%d-%B-%Y %H:%M:%S")

#output
"1992-11-28 09:25:04 IST"
```

### Math functions

### abs()

In [29]:
```
abs(-10)
```

10

### round()

In [30]:
```
round(10.23)
```

10

In [31]:
```
round(10.232343,2)
```

10.23

### sqrt()

In [33]:
```
sqrt(144)
```

12

### exp()

In [35]:
```
exp(1)
```

2.71828182845905

**Logarithm**

In [43]:
```
log(10)
```

2.30258509299405

In [44]:
```
log10(12)
```

1.07918124604762

**Trigonometry**

In [47]:
```
sin(90)
```

0.893996663600558

In [48]:
```
cos(90)
```

-0.448073616129l7

In [49]:
```
tan(90)
```

## apply function in R

apply() is a inbuilt function which takes matrix or Data Frame as an input and provides the output in list, array or vector. apply() Function is used to avoid explicit uses of constructs of loop. It is the most basic of all collections can be used over a matrice. The simplest example is to sum a matrice over all the columns.

This function takes 3 arguments:
apply(X, MARGIN, FUN)
Here:
-x: an array or matrix
-MARGIN: take a value or range between 1 and 2 to define where to apply the function:
-MARGIN=1`: the manipulation is performed on rows
-MARGIN=2`: the manipulation is performed on columns
-MARGIN=c(1,2)` the manipulation is performed on rows and columns
-FUN: tells which function to apply. Built functions like mean, median, sum, min, max and even user-defined functions can be applied›

The simplest example is to sum a matrice over all the columns. The code apply(m1, 2, sum) will apply the sum function to the matrix 5x6 and return the sum of each column accessible in the dataset.

```
m1 <- matrix(C<-(1:10),nrow=5, ncol=6)
m1
a_m1 <- apply(m1, 2, sum)
a_m1
```

Output:

```
> m1
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    6    1    6    1    6
[2,]    2    7    2    7    2    7
[3,]    3    8    3    8    3    8
[4,]    4    9    4    9    4    9
[5,]    5   10    5   10    5   10
> a_m1 <- apply(m1, 2, sum)
> a_m1
[1] 15 40 15 40 15 40
>
```