**LIST OF EXPERIMENTS**

1. Write a R program to take input from the user (name and age) and display the values. Also print the version of R installation.
2. Write a R program to get the details of the objects in memory.
3. Write a R program to create a sequence of numbers from 20 to 50 and find the mean of numbers from 20 to 60 and sum of numbers from 51 to 91.
4. Write a R program to create a simple bar plot of five subjects marks.
5. Write a R program to get the unique elements of a given string and unique numbers of vector.
6. Write a R program to create three vectors a,b,c with 3 integers. Combine the three vectors to become a 3×3 matrix where each column represents a vector. Print the content of the matrix.
7. Write a R program to create a 5 x 4 matrix , 3 x 3 matrix with labels and fill the matrix by rows and 3 × 3 matrix with labels and fill the matrix by columns.
8. Write a R program to combine three arrays so that the first row of the first array is followed by the first row of the second array and then first row of the third array.
9. Write a R program to create a two-dimensional 5x3 array of sequence of even integers greater than 50.
10. Write a R program to create an array using four given columns, three given rows, and two given tables and display the content of the array.
11. Write a R program to create an empty data frame.
12. Write a R program to create a data frame from four given vectors.
13. Write a R program to create a data frame using two given vectors and display the

duplicated elements and unique rows of the said data frame.

1. Write a R program to save the information of a data frame in a file and display the information of the file.
2. Write a R program to create a matrix from a list of given vectors.
3. Write a R program to concatenate two given matrices of same column but different rows.
4. Write a R program to find row and column index of maximum and minimum value in a given matrix.
5. Write a R program to append value to a given empty vector.
6. Write a R program to multiply two vectors of integers type and length 3.
7. Write a R program to find Sum, Mean and Product of a Vector, ignore element like NA or NaN.
8. Write a R program to list containing a vector, a matrix and a list and give names to the elements in the list.
9. Write a R program to create a list containing a vector, a matrix and a list and give names to the elements in the list. Access the first and second element of the list.
10. Write a R program to create a list containing a vector, a matrix and a list and remove the second element.
11. Write a R program to select second element of a given nested list.
12. Write a R program to merge two given lists into one list.
13. Write a R program to create a list named s containing sequence of 15 capital letters, starting from ‘E’.
14. Write a R program to assign new names "a", "b" and "c" to the elements of a given list.
15. Write a R program to find the levels of factor of a given vector.
16. Write a R program to create an ordered factor from data consisting of the names of months.
17. Write a R program to concatenate two given factor in a single factor.

**EXPERIMENT 1:**

**Write a R Program to take input from the User (name and age)and display the values .Also print the version of R installation.**

## Description:

R program to take input from the user and display that values. we can use the readline() function to take input from the user (terminal).

name = readline(prompt="Input your name: ")

Here the variable **name** will hold the data read from the user through the readline() function and the argument prompt will provide the request message to the user that will show in the terminal.

In this R program, we accept the user's values into **name** and **age** by providing an appropriate message to the user using prompt and print the string after concatenating with appropriate text.

**print the version of r installation**

print(R.version.string)

Here the built-in argument **R.version.string** will tell us which version of RStudio is running on our computer. It is given inside the print() function to display the currently running version of R.

**Algorithm:**

**STEP 1:** Take user input using **readline()** into variables **name**, **age** by prompting appropriate messages to the user

**STEP 2: Print** the user input along with other text with the help of  **Paste()**

**STEP 3:Print** the current version of R using **R.version.string**

**Source Code:**

name = readline(prompt="Input your name: ")

age = readline(prompt="Input your age: ")

print(paste("My name is",name, "and I am",age ,"years old."))

print(R.version.string)

**Output:**

Input your name: John

Input your age: 23

[1]”My name is John and Iam 23 years old.”

[1]”R version 4.1.2 (2021-11-01)”

**EXPERIMENT 2:**

**Write a R Program to get the details of the objects in memory.**

**Description:**

Here we are explaining how to write an R program to get the details of the objects in memory. Here we are using built-in functions **ls()**means list objects for this calculation. The **ls()** helps to return a vector of character strings with the names of the objects in the specified environment. If we called this function without arguments it will give the data sets and functions that a user has defined. And the **ls.str** is used for a long listing based on the **str**.

Below are the steps used in the R program to get the details of the objects in memory. In this R program, we directly give the values to variables **name,num1,num2, nums**. And print the function result. Here the variable **name** is assigned with a string **num1** with an integer value,**num2** with floating value, and **nums** with vector values.

**Algorithm:**

**STEP 1**: Assign variable **name,num1,num2, nums** with corresponding values

**STEP 2**: Call the built-in functions **ls()**for a listing of objects

**STEP 3**: First print given objects

**STEP 4**: Call the built-in functions **ls.str()** for string based long listing

**Source Code:**

name = "Python";

num1 = 8;

num2 = 1.5

nums = c(10, 20, 30, 40, 50, 60)

print(ls())

print("Details of the objects in memory:")

print(ls.str())

**Output:**

"num1" "num2" "name" "nums"

"Details of the objects in memory:"

n1 : num 8

n2 : num 1.5

name : chr "Python"

nums : num [1:6] 10 20 30 40 50 60

**EXPERIMENT 3:**

**Write a R program to create a sequence of numbers from 20 to 50 and find the mean of numbers from 20 to 60 and sum of numbers from 51 to 91.**

**Description:**

**seq()** function in [R Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is used to create a sequence of elements in a Vector. It takes the length and difference between values as optional argument.

**Syntax:**

seq(from, to, by, length.out)

**Parameters:**

**from:** Starting element of the sequence  
 **to:** Ending element of the sequence  
 **by:** Difference between the elements  
 **length.out:** Maximum length of the vector

**mean()** function in [R Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is used to calculate the arithmetic mean of the elements of the numeric vector passed to it as argument.

**Syntax:**

mean(x, na.rm)

**Parameters:**

**x:** Numeric Vector

**na.rm:** Boolean value to ignore NA value

**sum()** function in [R Programming Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) return the addition of the values passed as arguments to the function.

**Syntax:**

 sum(…)

**Parameters:**

…: numeric or complex or logical vectors

**Algorithm:**

**STEP 1**: Print Sequence of numbers from 20 to 50.

**STEP 2**: Call the built-in functions **seq()**for printing sequence of numbers

**STEP 3**: Print Mean of numbers from 20 to 60.

**STEP 4**: Call the built-in functions **mean()** for calculating mean from start number to end number and print mean() of numbers.

**Step 5:** Print Sum of numbers from 51 to 91

**Step6:** Call the built-in functions **sum()** for calculating sum from start number to end number and print sum() of numbers.

**Source Code:**

print("Sequence of numbers from 20 to 50:")

print(seq(20,50))

print("Mean of numbers from 20 to 60:")

print(mean(20:60))

print("Sum of numbers from 51 to 91:")

print(sum(51:91))

**Output:**

"Sequence of numbers from 20 to 50:"

20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44

45 46 47 48 49 50

"Mean of numbers from 20 to 60:"

40

"Sum of numbers from 51 to 91:"

2911

**EXPERIMENT 4:**

**Write a R program to create a simple bar plot of five subjects marks.**

**Description:**

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

### Syntax:

barplot(H,xlab,ylab,main, names.arg,col)

**parameters :**

**H** is a vector or matrix containing numeric values used in bar chart.

**xlab** is the label for x axis.

**ylab** is the label for y axis.

**main** is the title of the bar chart.

**names.arg** is a vector of names appearing under each bar.

**col** is used to give colors to the bars in the graph.

**Algorithm:**

**STEP 1**: Assign variable **marks** with corresponding values

**STEP 2**: Call the built-in functions **barplot()**for a list of values

**Source Code:**

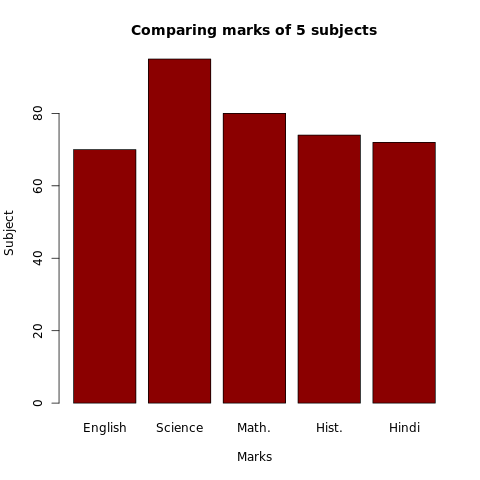
marks = c(70, 95, 80, 74,72)

barplot(marks,main = "Comparing marks of 5 subjects",xlab = "Marks",

ylab = "Subject",names.arg = c("English", "Science", "Math.", "Hist.","Hindi"),

col = "darkred",horiz = FALSE)

**Output:**



**EXPERIMENT 5:**

**Write a R program to get the unique elements of a given string and unique numbers of vector.**

**Description:**

**Unique()** function in R Programming Language it is used to return a vector, data frame, or array without any duplicate elements/rows.

**Syntax:** unique(x, incomparables, fromLast, nmax, …,MARGIN)

**Parameters:**

**x:** This parameter is a vector or a data frame or an array or NULL.

**incomparables:** This parameter is a vector of values that cannot be compared. If its value is FALSE, that means that all values can be compared, and maybe the only value accepted for methods other than the default. It will be coerced internally to the same type as x.

**fromLast:** This parameter indicates that if duplication should be considered from the last, i.e., the rightmost of identical elements will be kept. Its value is logical i.e., either true or false.

**nmax:** This parameter says the maximum number of unique items expected.

**…:** This is the arguments for particular methods.

**MARGIN:** This parameter says the array margin to be held fixed.

**Return value:** This function returns a vector, data frame, or array without any duplicate elements/rows.

**Algorithm:**

**STEP 1**: Assign variable **str1, nums** with corresponding values

**STEP 2**: Print the Original string.

**STEP 3:** Call the built-in functions **unique(tolower())**for a str1 and Print them.

**STEP 4**: Call the built-in functions **unique()**for a nums and Print them.

**Source Code:**

str1 = "The quick brown fox jumps over the lazy dog."

print("Original vector(string)")

print(str1)

print("Unique elements of the said vector:")

print(unique(tolower(str1)))

nums = c(1, 2, 2, 3, 4, 4, 5, 6)

print("Original vector(number)")

print(nums)

print("Unique elements of the said vector:")

print(unique(nums))

**Output:**

Original vector(string)"

"The quick brown fox jumps over the lazy dog."

"Unique elements of the said vector:"

"the quick brown fox jumps over the lazy dog."

"Original vector(number)"

1 2 2 3 4 4 5 6

"Unique elements of the said vector:"

1 2 3 4 5 6

**EXPERIMENT 6:**

**Write a R program to create three vectors a,b,c with 3 integers. Combine the three vectors to become a 3×3 matrix where each column represents a vector. Print the content of the matrix.**

**Description:**

In R programming language, vector is a basic object which consists of similar elements. The data type of vector can be integer, double, character, logical, complex or raw. A vector can be created by using **c() function**.

**Syntax:**

x <- c(val1, val2, .....)

Vectors in R are the same as the arrays in C language which are used to hold multiple data values of the same type. Vectors can also be used to create matrices.

Matrices can be created with the help of Vectors by using pre-defined functions in R Programming Language. These functions take vectors as arguments along with several other arguments for matrix dimensions, etc.

Functions used for Matrix creation:

1. **matrix()** function
2. **cbind()** function
3. **rbind()** function
4. **matrix() function**

The available data is present in a single/multiple vector, then **matrix()** function can be used to create the matrix by passing the following arguments in the function.

**Syntax:**

matrix(data, nrow, ncol, byrow, dimnames)

**where,**

**data** is the input vector which represents the elements in the matrix  
**nrow** specifies the number of rows to be created  
**ncol** specifies the number of columns to be created  
**byrow** specifies logical value. If TRUE, matrix will be filled by row. Default value is FALSE.  
**dimnames** specifies the names of rows and columns

1. **cbind() function**

**cbind()** function in R programming is used to combine vectors, data frames or matrices by columns and number of rows of data sets should be equal otherwise, output will be incomprehensible.  
**Syntax:**

cbind(v1, v2, v3, ....., deparse.level)

**where,**

**v1, v2, v3, ….** represent vectors, matrices or data frames  
**deparse.level** used for constructing labels for non-matrix like arguments. 0 for no labels. Default value is 1 or 2 for labelling from the arguments names.

1. **rbind() function**

**rbind()** function in R programming is used to combine vectors, data frames or matrices by rows and number of columns of data sets should be equal otherwise, output will be insignificant.  
**Syntax:**

rbind(v1, v2, v3, ....., deparse.level)

**where,**

**v1, v2, v3, ….** represent vectors, matrices or data frames  
**deparse.level** used for constructing labels for non-matrix like arguments. 0 for no labels. Default value is 1 or 2 for labelling from the arguments names.

**Algorithm:**

**STEP 1**: Assign variable **a,b,c** with corresponding values

**STEP 2**: Call the built-in functions **cbind()**for a combining of objects and Assign to m variable.

**STEP 3**: print m values.

**Source Code:**

a<-c(1,2,3)

b<-c(4,5,6)

c<-c(7,8,9)

m<-cbind(a,b,c)

print("Content of the said matrix:")

print(m)

**Output:**

"Content of the said matrix:"

a b c

1 4 7

2 5 8

3 6 9

**EXPERIMENT 7:**

**Write a R program to create a 5 x 4 matrix , 3 x 3 matrix with labels and fill the matrix by rows and 3 × 3 matrix with labels and fill the matrix by columns.**

**Description:**

**matrix() function**

The available data is present in a single/multiple vector, then matrix() function can be used to create the matrix by passing the following arguments in the function.

**Syntax:**

**matrix(data, nrow, ncol, byrow, dimnames)**

**where,**

**data**is the input vector which represents the elements in the matrix **nrow**specifies the number of rows to be created **ncol**specifies the number of columns to be created **byrow**specifies logical value. If TRUE, matrix will be filled by row. Default value is FALSE**.  
dimnames**specifies the names of rows and columns

**Algorithm:**

**STEP 1**: Assign variable **m1 with matrix values.**

**STEP 2**: Create **m1** matrix of 20 elements with 5 rows.

**STEP 3**: Print **m1** matrix

**STEP 4**: **Assign variables cells, rnames, cnames** with corresponding values

**STEP 5**: Create **m2** matrix of 9 elements with 3 rows filled by row ordering.

**STEP 6: Print m2 matrix.**

**STEP 7:** Create **m3** matrix of 9 elements with 3 rows filled by column ordering.

**STEP 5: Print m3 matrix.**

**Source Code:**

m1 = matrix(1:20, nrow=5, ncol=4)

print("5 × 4 matrix:")

print(m1)

cells = c(1,3,5,7,8,9,11,12,14)

rnames = c("Row1", "Row2", "Row3")

cnames = c("Col1", "Col2", "Col3")

m2 = matrix(cells, nrow=3, ncol=3, byrow=TRUE, dimnames=list(rnames, cnames))

print("3 × 3 matrix with labels, filled by rows: ")

print(m2)

print("3 × 3 matrix with labels, filled by columns: ")

m3 = matrix(cells, nrow=3, ncol=3, byrow=FALSE, dimnames=list(rnames, cnames))

print(m3)

**Output:**

"5 × 4 matrix:"

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

[1,] 1 6 11 16

[2,] 2 7 12 17

[3,] 3 8 13 18

[4,] 4 9 14 19

[5,] 5 10 15 20

[1] "3 × 3 matrix with labels, filled by rows: "

Col1 Col2 Col3

Row1 1 3 5

Row2 7 8 9

Row3 11 12 14

[1] "3 × 3 matrix with labels, filled by columns: "

Col1 Col2 Col3

Row1 1 7 11

Row2 3 8 12

Row3 5 9 14

**EXPERIMENT 8:**

**Write a R program to combine three arrays so that the first row of the first array is Followed by the first row of the second array and then first row of the third array.**

**Description:**

The arrays vectors can be created in R using the **rbind()** operation, by binding the rows together. The row binding operation in R creates a matrix that contains the number of rows specified. Similarly, n number of arrays can be created. The **cbind()** operation is then applied which takes as arguments the array vectors. It creates a combined matrix where the merging takes place using columns.

**cbind**(arr1, arr2, arr3..)

rep() is a very useful function for creating a vector by repeating a given number vector with the specified number of times.

**rep(v1,n1)**

Here, **v1 is repeated n1 times.**

The t() method is then applied over the result to create a transpose of the obtained output. The rows and columns are then reversed to produce a transpose matrix. The matrix operation is then applied over the output, where

**Syntax:**matrix ( data, ncol, byrow)

**Arguments :**

* data – The data to convert into a matrix
* ncol – The number of columns to produce in the result matrix
* byrow – logical. If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.

**Algorithm:**

**STEP 1**: Create **num1, num2,num3 matrices.**

**STEP 2**: **rbind() and rep() used to assign values to num1, num2,num3.**

**STEP 3**: Print **num1, num2,num3** matrices

**STEP 4**: **Assign variables cells, rnames, cnames** with corresponding values

**STEP 5**: Create **a** matrix by calling like **matrix(t(cbind(num1,num2,num3)),ncol=3, byrow=T)**.

**STEP 6: Print** Combine three arrays, taking one row from each one by one

**STEP 7: Print a matrix.**

**Source Code:**

num1 = rbind(rep("A",3), rep("B",3), rep("C",3))

print("num1")

print(num1)

num2 = rbind(rep("P",3), rep("Q",3), rep("R",3))

print("num2")

print(num2)

num3 = rbind(rep("X",3), rep("Y",3), rep("Z",3))

print("num3")

print(num3)

a = matrix(t(cbind(num1,num2,num3)),ncol=3, byrow=T)

print("Combine three arrays, taking one row from each one by one:")

print(a)

**Output:**

[1] "num1"

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

[1,] "A" "A" "A"

[2,] "B" "B" "B"

[3,] "C" "C" "C"

[1] "num2"

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

[1,] "P" "P" "P"

[2,] "Q" "Q" "Q"

[3,] "R" "R" "R"

[1] "num3"

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

[1,] "X" "X" "X"

[2,] "Y" "Y" "Y"

[3,] "Z" "Z" "Z"

[1] "Combine three arrays, taking one row from each one by one:"

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

[1,] "A" "A" "A"

[2,] "P" "P" "P"

[3,] "X" "X" "X"

[4,] "B" "B" "B"

[5,] "Q" "Q" "Q"

[6,] "Y" "Y" "Y"

[7,] "C" "C" "C"

[8,] "R" "R" "R"

[9,] "Z" "Z" "Z"

**EXPERIMENT 9:**

**Write a R program to create a two-dimensional 5x3 array of sequence of even integers Greater than 50.**

**Description:**

Here we are using a built-in function **array()** for this. This function helps to create the array.

**array**(data = NA, dim = length(data), dimnames = NULL)

Where **data**is a vector giving data to fill the array. The **dim** is the dim attribute for the array to be created and **dimnames** are either NULL or the names for the dimensions.

R program to create a two-dimensional 5×3 array of a sequence of even integers. In this R program, we directly give the values to an array. Here we are using variable **Arr**for holding the list elements**.**Here we are displaying a 5x3 array of integers greater than 50 in length 15(means number of elements in the array is 15) and the difference between the elements is 2 . Finally, print the array.

**Source Code:**

a <- array(seq(from = 50, length.out = 15, by = 2), c(5, 3))

print("Content of the array:")

print("5×3 array of sequence of even integers greater than 50:")

print(a)

**Algorithm:**

**STEP 1**: Assign array elements into the variable **Arr**

**STEP 2**: The sequence of elements are created by using **seq()**

**STEP 3**:limit the count to 15 as **length.out** = 15 and the numbers should be even,

difference between the numbers is 2 as **by=2**

**STEP 4**: Print the 5x3 array

**Output:**

[1] "Content of the array:"

[1] "5×3 array of sequence of even integers greater than 50:"

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

[1,] 50 60 70

[2,] 52 62 72

[3,] 54 64 74

[4,] 56 66 76

[5,] 58 68 78

**EXPERIMENT 10:**

**Write a R program to create an array using four given columns, three given rows, and two given tables and display the content of the array.**

**Description:**

To create an array using given columns, rows, and tables in R Programming Language. The array is created using the **array()** function.

**Syntax:**Array(vector.., dim=c(row, columns, tables))

**Parameter:**

**x:** vector

**dim:** values used to create array

**Algorithm:**

**STEP 1**: Assign array elements into the variable **A**

**STEP 2**: The array is created by using **array(1:30, dim=c(3,5,2))**

**STEP 3**: Print the array

**Source Code:**

array1 = array(1:30, dim=c(3,4,2))

print(array1)

**Output:**

, , 1

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

[1,] 1 4 7 10

[2,] 2 5 8 11

[3,] 3 6 9 12

, , 2

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

[1,] 13 16 19 22

[2,] 14 17 20 23

[3,] 15 18 21 24

**EXPERIMENT 11:**

**Write a R program to create an empty data frame.**

**Description:**

**A data frame is used for storing data tables which has a list of vectors with equal length**. It is a matrix-like structure with different types of values or we can say that it is a table of data. It consists of vectors of the same length as columns in a table.

There are two basic ways to create an empty data frame in R

### **Method 1: Matrix with Column Names**

The first way to create an empty data frame is by using the following steps:

* Define a matrix with 0 rows and however many columns you’d like.
* Then use the **data.frame()** function to convert it to a data frame and the **colnames()**  function to give it column names.
* Then use the **str()**function to analyze the structure of the resulting data frame.

For example:

**#create data frame with 0 rows and 5 columns**

**df <- data.frame(matrix(ncol = 5, nrow = 0))**

**#provide column names**

**colnames(df) <- c('var1', 'var2', 'var3', 'var4', 'var5')**

**#view structure of the data frame**

**str(df)**

**'data.frame': 0 obs. of 5 variables:**

**$ var1: logi**

**$ var2: logi**

**$ var3: logi**

**$ var4: logi**

**$ var5: logi**

We can see that the resulting data frame has 0 observations (i.e. rows), 5 variables (i.e. columns), and each of the variables are of the class logical.

Although each variable is of the class logical, you can still add rows to the variables that are of different types.

### **Method 2: Initialize Empty Vectors**

The second way to create an empty data frame is by using the following steps:

* Define a data frame as a set of empty vectors with specific class types.
* Specify **stringsAsFactors=False** so that any character vectors are treated as strings, not factors.

For example:

**#create data frame with 5 empty vectors**

**df2 <- data.frame(Doubles=double(),**

**Integers=integer(),**

**Factors=factor(),**

**Logicals=logical(),**

**Characters=character(),**

**stringsAsFactors=FALSE)**

**#view structure of the data frame**

**str(df2)**

**'data.frame': 0 obs. of 5 variables:**

**$ Doubles : num**

**$ Integers : int**

**$ Factors : Factor w/ 0 levels:**

**$ Logicals : logi**

**$ Characters: chr**

We can see that the resulting data frame has 0 observations (i.e. rows),  5 variables (i.e. columns), and each of the variables are five different classes. Note that we were also able to provide column names for the data frame in just one step (e.g. the first column name is “Doubles”, the second column name is “Integers” and so on.

**Algorithm:**

**STEP 1**: Consider variable**df**as a data frame.

**STEP 2**: Call **data.frame()** with different data types

**STEP 3**: Assign **df** with function result

**STEP 4**: print the result of the function

**Source Code:**

df = data.frame(Ints=integer(),

Doubles=double(),

Characters=character(),

Logicals=logical(),

Factors=factor(),

stringsAsFactors=FALSE)

print("Structure of the empty dataframe:")

print(str(df))

**Output:**

[1] "Structure of the empty dataframe:"

'data.frame': 0 obs. of 5 variables:

$ Ints : int

$ Doubles : num

$ Characters: chr

$ Logicals : logi

$ Factors : Factor w/ 0 levels:

NULL

**EXPERIMENT 12:**

**Write a R program to create a data frame from four given vectors.**

**Description:**

**A data frame is used for storing data tables which has a list of vectors with equal length**. It is a matrix-like structure with different types of values or we can say that it is a table of data. It consists of vectors of the same length as columns in a table.

Here we are explaining how to write an R program to create a data frame from four given vectors. We can use a built-in function **data.frame()** for creating a data frame in R. The function **data.frame()** creates data frames that have tightly coupled collections of variables.

data.frame(...,row.names=NULL,check.rows=FALSE,check.names=TRUE,fix.empty.names=TRUE,stringsAsFactors=default.stringsAsFactors())

Where **dots(...)** indicates the arguments are of either the form value or tag = value and **row.names** is a NULL or a single integer or character string.

**Algorithm:**

**STEP 1**: Assign variables name, score, attempts, qualify with vector values

**STEP 2**: First **print**original vector values

**STEP 3:** Call the in-built function **data.frame**(name, score, attempts, qualify) and assign the result of the function to variable **df**

**STEP 4**: print the data frame **df**

**Source Code:**

name = c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas')

score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)

attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)

qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes')

print("Original data frame:")

print(name)

print(score)

print(attempts)

print(qualify)

df = data.frame(name, score, attempts, qualify)

print(df)

**Output:**

[1] "Original data frame:"

[1] "Anastasia" "Dima" "Katherine" "James" "Emily" "Michael"

[7] "Matthew" "Laura" "Kevin" "Jonas"

[1] 12.5 9.0 16.5 12.0 9.0 20.0 14.5 13.5 8.0 19.0

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

[1] "yes" "no" "yes" "no" "no" "yes" "yes" "no" "no" "yes"

name score attempts qualify

1 Anastasia 12.5 1 yes

2 Dima 9.0 3 no

3 Katherine 16.5 2 yes

4 James 12.0 3 no

5 Emily 9.0 2 no

6 Michael 20.0 3 yes

7 Matthew 14.5 1 yes

8 Laura 13.5 1 no

9 Kevin 8.0 2 no

10 Jonas 19.0 1 yes

**EXPERIMENT 13:**

**Write a R program to create a data frame using two given vectors and display the**

**duplicated elements and unique rows of the said data frame.**

**Description:**

R program to create a data frame using two given vectors and display the duplicated elements and unique rows. Here we are using a built-in function **data.frame()** for this. A data frame is used for storing data tables which has a list of vectors with equal length. The data frames are created by function data.frame(), which has tightly coupled collections of variables.

**Syntax**:

data.frame(…, row.names = NULL, check.rows = FALSE,check.names = TRUE, fix.empty.names = TRUE,stringsAsFactors = default.stringsAsFactors())

Where **dots(...)** indicates the arguments are of either the form value or tag = value and **row. name** is a NULL or a single integer or character string.

R program to create a data frame using two given vectors and display the duplicated elements and unique rows. In this R program, we directly give the data frame to a built-in function. Here we are using variables**v1,v2**for holding different types of vectors, and **V** for holding created data frame**.**Call the function data.frame() for creating**dataframe**. For getting duplicate elements call the method like **duplicated(v1v2)**and for getting unique elements call it like **unique(v1v2).**

**Algorithm:**

**STEP 1**: Assign variables **v1,v2**with vector values and **V** for data frame

**STEP 2**: First print original vector values

**STEP 3**: Create a data frame from given vectors as **data.frame(v1,v2)**

**STEP 4**: Print the duplicate elements by calling **duplicated(v1v2)**

**STEP 5**: Print the unique elements by calling **unique(v1v2)**

**Source Code:**

v1 = c(10,20,10,10,40,50,20,30)

v2 = c(10,30,10,20,0,50,30,30)

print("Original data frame:")

v1v2= data.frame(v1,v2)

print(v1v2)

print("Duplicate elements of the said data frame:")

print(duplicated(v1v2))

print("Unique rows of the said data frame:")

print(unique(v1v2))

**Output:**

[1] "Original data frame:"

v1 v2

1 10 10

2 20 30

3 10 10

4 10 20

5 40 0

6 50 50

7 20 30

8 30 30

[1] "Duplicate elements of the data frame:"

[1] FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE

[1] "Unique rows of the data frame:"

v1 v2

1 10 10

2 20 30

4 10 20

5 40 0

6 50 50

8 30 30

**EXPERIMENT 14:**

**Write a R program to save the information of a data frame in a file and display the**

**Information of the file.**

**Description:**

To save the information of a data frame in a RDATA file and display the information of the file using R Programming language. To save the information of a data frame in a file and display the information of the file in R language is as follows:

1. Using the save function to save the file in.RData format.
2. Using the load() function to load that saved.RData file
3. Using file.info() function to get the information of a particular file.

**Step 1: Using the save() function to save the file in.RData format**

In this step user need to call the save(), function with the name of the new file and with its format passed as its parameter, This function will simply save the file as per the user-specified parameter in the working directory.

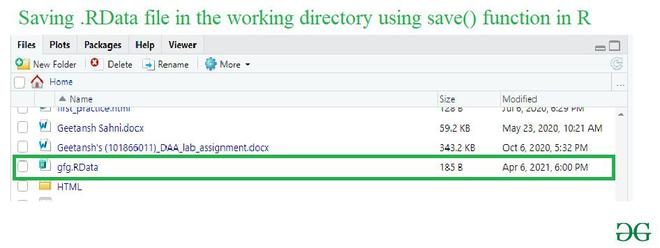
**save() function:**This function writes an external representation of R objects to the specified file.

**Syntax:**save(…, list = character(), file = stop(“‘file’ must be specified”),ascii = FALSE, version = NULL, envir = parent.frame(), compress = isTRUE(!ascii), compression\_level, eval.promises = TRUE, precheck = TRUE)

**Example:**

In this example, we will be simply saving a data frame in new.RData file in the working directory.

|  |
| --- |
| **gfg\_data = data.frame(A = c(7,6,2,8,1),**  **B = c(4,2,9,7,3),**  **C = c(1,7,2,6,8))**    **print("Dataframe:->")**  **print(gfg\_data)**  **save(gfg\_data, file = "gfg.RData")** |

**Output:**

**Step 2: Using the load() function to load that saved.RData file**

In this step user just need to call the load() function with the name of the file saved in step-1 as its parameter, further this will help to load the save file in the console so that the user can manage the operation on this file.

**Load() function:**This function is used to reload datasets written with the function save.

**Syntax:**load(file, envir = parent.frame(), verbose = FALSE)

**Parameters:**

**File:-**a character string giving the name of the file to load.

**envir:-**the environment where the data should be loaded.

**Verbose:-**should item name be printed during loading?

**Example:**

Under this example, we will be loading the .RData file using load() function which was saved in the previous example.

|  |
| --- |
| **gfg\_data= data.frame(A=c(7,6,2,8,1),**  **B=c(4,2,9,7,3),**  **C=c(1,7,2,6,8))**    **print("Dataframe:->")**  **print(gfg\_data)**    **save(gfg\_data,file="gfg.RData")**  **load("gfg.RData")** |

**Step3: Using file.info() function to get the information of a particular file.**

This is the final step to save the information of a data frame in a file and displays its information, Here the step1 and step2 is used to save the information of the given data-frame in a particular file and this step3 will be used to displaying the information of the file where the given data-frame is been saved and for this, we will be using the **file.info()** function with the name of the previously loaded file in the console to display the information of the file saved.

**file.info() function:**This is a utility function used to extract information about files on the user’s file systems.

**Syntax:**file.info(…, extra\_cols = TRUE)

**Parameter:**

**…:-**character vectors containing file paths

**extra\_cols:-**Logical: return all cols rather than just the first six.

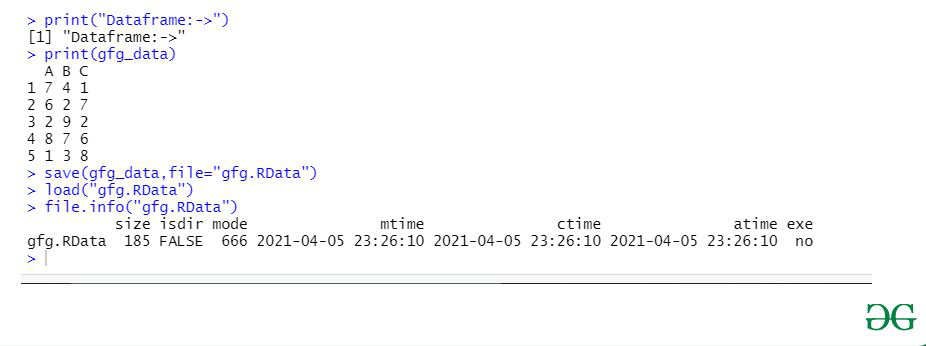
***Returns:****save*

The complete information of the file including – size, mode, ctime, exe, time, atime, and isdir.

**Example:**

Using the **file.info()** function in R language we will be displaying the completed information of the file save and loaded in the previous examples.

**Output:**



**Algorithm:**

**STEP 1**: Assign variables name, score, attempts, qualify with vector values

**STEP 2**: First **print**original vector values

**STEP 3:** Call the in-built function **data.frame**(name, score, attempts, qualify) and assign the result of the function to variable **exam\_data**

**STEP 4**: print the data frame **exam\_data**

**STEP 5: Save the exam\_data dataframe into data.rda**

**STEP 6:Load the saved data.rda file**

**STEP 7:Get the information of data.rda file.**

**Source Code:**

exam\_data = data.frame(

name = c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'),

score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19),

attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1),

qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes')

)

print("Original dataframe:")

print(exam\_data)

save(exam\_data,file="data.rda")

load("data.rda")

file.info("data.rda")

**Output:**

[1] "Original dataframe:"

name score attempts qualify

1 Anastasia 12.5 1 yes

2 Dima 9.0 3 no

3 Katherine 16.5 2 yes

4 James 12.0 3 no

5 Emily 9.0 2 no

6 Michael 20.0 3 yes

7 Matthew 14.5 1 yes

8 Laura 13.5 1 no

9 Kevin 8.0 2 no

10 Jonas 19.0 1 yes

size isdir mode mtime ctime

data.rda 344 FALSE 644 2018-10-25 12:06:09 2018-10-25 12:06:09

atime uid gid uname grname

data.rda 2018-10-25 12:06:09 1000 1000 trinket trinket

**EXPERIMENT 15:**

**Write a R program to create a matrix from a list of given vectors.**

**Description:**

**For** Loop in R.It is a type of control statement that enables one to easily construct a loop that has to run statements or a set of statements multiple times. For loop is commonly used to iterate over items of a sequence. It is an entry controlled loop, in this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

**R – For loop Syntax:**

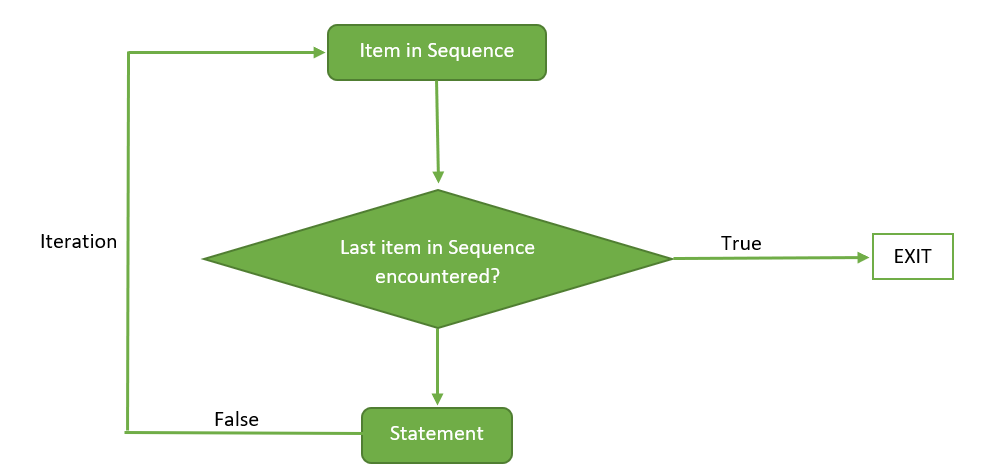
for (value in sequence)

{

statement

}

**For Loop Flow Diagram:**



Below are some programs to illustrate the use of **for** loop in R programming.

|  |
| --- |
| **Example :**Program to display numbers from 1 to 5 using for loop in R.  # R program to demonstrate the use of for loop    # using for loop  for (val in 1: 5)  {      # statement      print(val)  } |

**Output:**

[1] 1

[1] 2

[1] 3

[1] 4

[1] 5

Here, for loop is iterated over a sequence having numbers from 1 to 5. In each iteration, each item of the sequence is displayed.

**do.call** constructs and executes a function call from a name or a function and a list of arguments to be passed to it.

**Syntax:**do.call(what, args, quote = FALSE, envir = parent.frame())

**Parameters:**

**what:**Either a function or a non-empty character string naming the function to be called.

**args:** A list of arguments to the function call. The names attribute of args gives the argument names.

**quote:** A logical value indicating whether to quote the arguments.

**envir:** An environment within which to evaluate the call. This will be most useful if what is a character string and the arguments are symbols or quoted expressions.

If quote is FALSE, the default, then the arguments are evaluated (in the calling environment, not in envir). If quote is TRUE then each argument is quoted (see quote) so that the effect of argument evaluation is to remove the quotes -- leaving the original arguments unevaluated when the call is constructed.

The behavior of some functions, such as substitute, will not be the same for functions evaluated using do.call as if they were evaluated from the interpreter. The precise semantics are currently undefined and subject to change.

**Algorithm:**

**STEP1:**Create a 5 lists by using for loop

**STEP2:**Convert these lists into vectors

**STEP3:**Print list of vectors

**STEP4:**Combine those vectors to convert into matrix by calling **do.call(rbind, l)**

**STEP5:** Print **result** matrix

**Source Code:**

l = list()

for (i in 1:5) l[[i]] <- c(i, 1:4)

print("List of vectors:")

print(l)

result = do.call(rbind, l)

print("New Matrix:")

print(result)

**Output:**

"List of vectors:"

1 1 2 3 4

2 1 2 3 4

3 1 2 3 4

4 1 2 3 4

5 1 2 3 4

"New Matrix:"

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

[1,] 1 1 2 3 4

[2,] 2 1 2 3 4

[3,] 3 1 2 3 4

[4,] 4 1 2 3 4

[5,] 5 1 2 3 4

**EXPERIMENT 16:**

**Write a R program to concatenate two given matrixes of same column but different rows.**

**Description:**

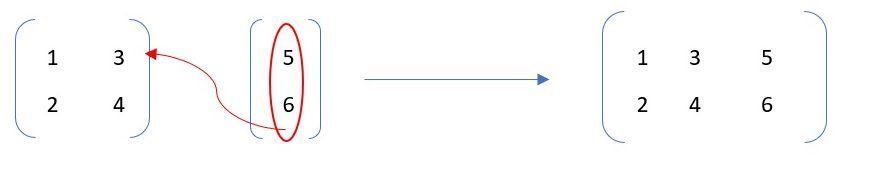
Combining matrices involves the concatenation of two or more smaller matrices, either row or column wise to form a larger matrix. It is basically a data manipulation operation where the involved matrices must be of compatible sizes to execute the operation. Matrices can be combined either horizontally or vertically.

There are two ways of combining matrices in R:

* Column-wise combination
* Row-wise Combination

#### Column-Wise Combination

Column bind, **cbind()** function in R, is used to merge two data frames or matrices A m\*n and Bm\*p (n may or may not be equal to p) together by their columns. The matrices involved should have the same number of rows.

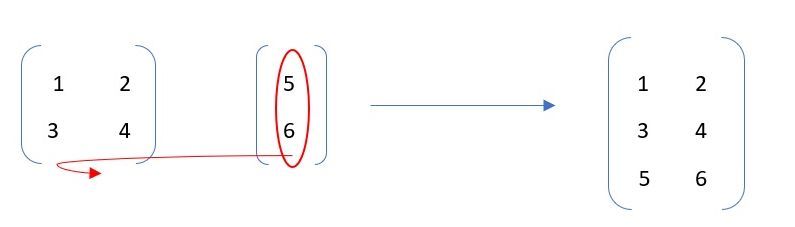


**Properties:**

* The number of columns of the resultant matrix is the sum of the columns of the two matrices.
* **Non-Commutative:** The columns are merged in the order in which the parameters are specified in the function. Therefore, cbind(a, b) != cbind(b, a)
* **Associative:** cbind(cbind(a, b), c) = cbind(a, cbind(b, c))

#### Row-Wise Combination

Row bind, **rbind()** function in R, is used to merge two data frames or matrices A m\*p and B n\*p (m may or may not be equal to n), together by their rows. The matrices involved should have the same number of columns.



**Properties:**

* The number of rows of the resultant matrix is the sum of the rows of the two matrices.
* **Non-Commutative:** The rows are merged in the order in which the parameters are specified in the function. Therefore, rbind(a, b) != rbind(b, a)
* **Associative:** rbind(rbind(a, b), c) = rbind(a, rbind(b, c))

**Algorithm:**

**STEP 1:** Create a x,y matrices by calling matrix().

**STEP 2:** Print x,y matrices

**STEP3 :** Combining dimensions of matrix with rbind().

**STEP 4:** Print result of dimensions of matrix

**STEP5:** Combine x,y matrices and print resultant matrix.

**Source Code:**

x = matrix(1:12, ncol=3)

y = matrix(13:24, ncol=3)

print("Matrix-1")

print(x)

print("Matrix-2")

print(y)

print("After concatenating dimensions of matrix:")

result = dim(rbind(x,y))

print(result)

print("After concatenating two given matrices:")

print(rbind(x,y))

**Output:**

[1] "Matrix-1"

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

[1,] 1 5 9

[2,] 2 6 10

[3,] 3 7 11

[4,] 4 8 12

[1] "Matrix-2"

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

[1,] 13 17 21

[2,] 14 18 22

[3,] 15 19 23

[4,] 16 20 24

[1] "After concatenating dimensions of matrix:"

[1] 8 3

[1] "After concatenating two given matrices:"

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

[1,] 1 5 9

[2,] 2 6 10

[3,] 3 7 11

[4,] 4 8 12

[5,] 13 17 21

[6,] 14 18 22

[7,] 15 19 23

[8,] 16 20 24

**EXPERIMENT 17:**

**Write a R program to find row and column index of maximum and minimum value in a given matrix.**

**Description:**

**max() function** which is used to find the maximum element present in an object. This object can be a Vector, a list, a matrix, a data frame, etc.

 “**which()”** function is used to get the index or position of the value which satisfies the given condition. Then we have printed the maximum value along with its row and column index.

**Syntax**: which(m == max(m), arr.ind=TRUE)

**min() function** which is used to find the minimum element present in an object. This object can be a Vector, a list, a matrix, a data frame, etc.

**“which()”** function is used to get the index or position of the value which satisfies the given condition. Then we have printed the minimum value along with its row and column index.

**Syntax**: which(m == min(m), arr.ind=TRUE)

**Algorithm**:

**STEP 1:** Assign variable**m**with matrix values

**STEP 2:** First print original matrix values

**STEP 3:** Assign variable **result** with dimensions of matrix max value by calling **which(m == max(m), arr.ind=TRUE)**

**STEP 4:** Print the maximum value from the matrix

**STEP 5:** Assign variable **result** with dimensions of matrix min value by calling **which(m == min(m), arr.ind=TRUE)**

**STEP 6:** Print the minimum value from the matrix

**Source Code:**

m = matrix(c(1:16), nrow = 4, byrow = TRUE)

print("Original Matrix:")

print(m)

result = which(m == max(m), arr.ind=TRUE)

print("Row and column of maximum value of the said matrix:")

print(result)

print(paste("maximum value is",m[result]))

result = which(m == min(m), arr.ind=TRUE)

print("Row and column of minimum value of the said matrix:")

print(result)

print(paste("minimum value is",m[result]))

**Output:**

[1] "Original Matrix:"

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

[1,] 1 2 3 4

[2,] 5 6 7 8

[3,] 9 10 11 12

[4,] 13 14 15 16

[1] "Row and column of maximum value of the said matrix:"

row col

[1,] 4 4

[1] "maximum value is 16"

[1] "Row and column of minimum value of the said matrix:"

row col

[1,] 1 1

[1] "minimum value is 1"

**EXPERIMENT 18:**

**Write a R program to append value to a given empty vector.**

**Description:**

An R program to append values to a given empty vector. Using for loop each of the given values are appended to the vector. The assigning of values can be done as like **vector[i] <- values[i]**.

  The iteration starts from 1 to the length of given vector values. And the value are assigned like **V[i] <- Val[i]**. Here we are using **V** as empty vector and variable **Val** as vector values.

**Algorithm:**

**STEP 1**: Consider empty vector as variable **V**

**STEP 2**: Consider vector values as variable **Val**

**STEP 3**: Iterate through for loop from **1**to**length(Val)**

**STEP 4**: Assign values to empty vector as **V[i] <- Val[i]**

**STEP 4**: Print the vector **V** as result

**Source Code:**

vector = c()

values = c(0,1,2,3,4,5,6,7,8,9)

for (i in 1:length(values))

vector[i] <- values[i]

print(vector)

**Output:**

0 1 2 3 4 5 6 7 8 9

**EXPERIMENT 19:**

**Write a R program to multiply two vectors of integers type and length 3.**

**Description:**

We can give two vector values to calculate the multiplication. For the calculation of vector product here we use the product(\*) operator.

In this R program, we accept the vector values into variables **x** and **y**. The result value of vectors is assigned variable **z**. Finally the variable **z** is printed as output vector.

**Algorithm:**

**STEP 1**: take the two vector values into the variables **x,y**

**STEP 2**: Consider **z** as the result vector

**STEP 3**: Calculate the vector product using the **\***operator

**STEP 4**: First print the original vectors

**STEP 5**: Assign the result of product value to vector **z** as **z=x\*y**

**STEP 6**: print the vector **z** as result vector

**Source Code:**

x = c(10, 20, 30)

y = c(20, 10, 40)

print("Original Vectors:")

print(x)

print(y)

print("Product of two Vectors:")

z = x \* y

print(z)

**Output:**

"Original Vectors:"

10 20 30

20 10 40

Product of two Vectors:"

200 200 1200

**EXPERIMENT 20:**

**Write a R program to find Sum, Mean and Product of a Vector, ignore element like NA or NaN**

**Description:**

An R program to find the Sum, Mean, and Product of a Vector, ignoring elements like NA or NaN. Here we are using built-in functions **sum, mean, prod** for this calculation. The numbers are passed to these functions directly here. The function **sum()** returns the sum of all the values present in its arguments. The sum of the values divided with the number of values in a data series is calculated using the **mean()**function. Finally, the **prod()** is for finding the product of given arguments.

**sum(…, na.rm = FALSE)**

**mean(x, …)**

**prod(…, na.rm = FALSE)**

In the above function argument structure by making **na.rm = TRUE** we can avoid the elements like **NA, NAN**.

Consider variable **x**for assigning vector value. And call each function by giving x as an argument. Make sure na.rm should be true as like **na.rm = TRUE**. Finally, print the function result.

**Algorithm:**

**STEP 1**: Use the built-in functions

**STEP 2**: call sum() with vector and na.rm = TRUE as argument

**STEP 3**: call mean() with vector and na.rm = TRUE as argument

**STEP 4**: call prod() with vector and na.rm = TRUE as argument

**STEP 5**: print the result of each function

**Source Code:**

x = c(10, NULL, 20, 30, NA)

print("Sum:")

#ignore NA and NaN values

print(sum(x, na.rm=TRUE))

print("Mean:")

print(mean(x, na.rm=TRUE))

print("Product:")

print(prod(x, na.rm=TRUE))

**Output:**

"Sum:"

60

"Mean:"

20

"Product:"

6000

**EXPERIMENT 21:**

**Write a R program to list containing a vector, a matrix and a list and give names to the elements in the list**.

**Description:**

An R program to create a list of elements using vectors, matrices, and functions. Here we are using a built-in function **list()** for this. This function helps to create a list.

**Syntax :** list(…)

Where **....(dots)**are the objects, possibly named.

In this R program, we directly give the values to a built-in function **list()**. Here we are using variables **L**for holding the list elements**.**Call the function **list()**with different types of elements. Here **month.abb** isthe three-letter abbreviations for the English month names. And **asin** helps to find the sine inverse of the given value. The returned angle is in the range -pi/2 through pi/2.Finally, print the list content.

**Algorithm:**

**STEP 1**: Assign variables **L** with a lists

**STEP 2**: include list of vectors

**STEP 3** include list of months as **month.abb**

**STEP 4**: include list of matrix as **matrix(c(3, -8, 1, -3), nrow = 2)**

**STEP 4**: include sine inverse of value as **asin**

**STEP 4**: Finally print the list contents

**Source Code:**

L = list(

c(1, 2, 3, 4, 5, 6),

month.abb,

matrix(c(3, -8, 1, -3), nrow = 2),

asin

)

print("Content of the list:")

print(L)

**Output:**

[1] "Content of the list:"

[[1]]

[1] 1 2 3 4 5 6

[[2]]

[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

[[3]]

[,1] [,2]

[1,] 3 1

[2,] -8 -3

[[4]]

function (x) .Primitive("asin")

**EXPERIMENT 22:**

**Write a R program to create a list containing a vector, a matrix and a list and give names to the elements in the list. Access the first and second element of the list**.

**Description:**

After we built our list, we can access it quite easily. We need to use the [index] to select an element in a list. The value inside the double square bracket represents the position of the item in a list we want to extract. For instance, we pass 2 inside the parenthesis, R returns the second element listed.

**Algorithm:**

**STEP 1:** Assign variables **list\_data**  with a lists

**STEP 2:** include list of vector as **c**

**STEP 3:** include list of matrix

**STEP 4:** include list of vector as list

**STEP 5:**Print **list\_data**

**STEP 6:**Assign and print column names to the list

**STEP 7:**Print 1&2 lists of **list\_data**

**Source Code:**

list\_data <- list(c("Red","Green","Black"), matrix(c(1,3,5,7,9,11), nrow = 2),

list("Python", "PHP", "Java"))

print("List:")

print(list\_data)

names(list\_data) = c("Color", "Odd numbers", "Language(s)")

print("List with column names:")

print(list\_data)

print('1st element:')

print(list\_data[1])

print('2nd element:')

print(list\_data[2])

**Output:**

list\_data <- list(c("Red","Green","Black"), matrix(c(1,3,5,7,9,11), nrow = 2),

+ list("Python", "PHP", "Java"))

> print("List:")

[1] "List:"

> print(list\_data)

[[1]]

[1] "Red" "Green" "Black"

[[2]]

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

[1,] 1 5 9

[2,] 3 7 11

[[3]]

[[3]][[1]]

[1] "Python"

[[3]][[2]]

[1] "PHP"

[[3]][[3]]

[1] "Java"

> names(list\_data) = c("Color", "Odd numbers", "Language(s)")

> print("List with column names:")

[1] "List with column names:"

> print(list\_data)

$Color

[1] "Red" "Green" "Black"

$`Odd numbers`

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

[1,] 1 5 9

[2,] 3 7 11

$`Language(s)`

$`Language(s)`[[1]]

[1] "Python"

$`Language(s)`[[2]]

[1] "PHP"

$`Language(s)`[[3]]

[1] "Java"

> print('1st element:')

[1] "1st element:"

> print(list\_data[1])

$Color

[1] "Red" "Green" "Black"

> print('2nd element:')

[1] "2nd element:"

> print(list\_data[2])

$`Odd numbers`

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

[1,] 1 5 9

[2,] 3 7 11

**EXPERIMENT 23:**

**Write a R program to create a list containing a vector, a matrix and a list and remove the second element.**

**Description:**

The vector to be deleted can be assigned to a **NULL** value using its corresponding position in the list. The changes are made to the original list. The vectors can be deleted at any position in the list, thereby, the size reduces by one and the elements are pushed back accordingly. The entire list can be deleted by successively forming a loop and deleting elements one by one.

**Algorithm:**

**STEP 1:** Assign variables **list\_data**  with a lists

**STEP 2:** include list of vector as **c**

**STEP 3:** include list of matrix

**STEP 4:** include list of vector as list

**STEP 5:**Print **list\_data**

**STEP 6:** Remove the second element of the list by using **list\_data[2] = NULL**

**STEP 7:**Print updated **list\_data**

**Source Code:**

list\_data <- list(c("Red","Green","Black"), matrix(c(1,3,5,7,9,11), nrow = 2),

list("Python", "PHP", "Java"))

print("List:")

print(list\_data)

print("Remove the second element of the list:")

list\_data[2] = NULL

print("New list:")

print(list\_data)

**Output:**

[1] "List:"

[[1]]

[1] "Red" "Green" "Black"

[[2]]

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

[1,] 1 5 9

[2,] 3 7 11

[[3]]

[[3]][[1]]

[1] "Python"

[[3]][[2]]

[1] "PHP"

[[3]][[3]]

[1] "Java"

[1] "Remove the second element of the list:"

[1] "New list:"

[[1]]

[1] "Red" "Green" "Black"

[[2]]

[[2]][[1]]

[1] "Python"

[[2]][[2]]

[1] "PHP"

[[2]][[3]]

[1] "Java"

**EXPERIMENT 24:**

**Write a R program to select second element of a given nested list**

**Description:**

An R program to select the second element of a given nested list. Here we are using a built-in function lapply() for this. Usually, in programming languages, we use loops for this type of program. There are different types of apply() functions are available to work on lists, vectors, and data frames, also they can be viewed as substitutes to the loop constructs. lapply() is one of them and ' l ' stands for list. This function helps to perform some operations on list objects and return a list of the same length as the list object given as an argument. The return list will be the result of applying FUN to the corresponding elements of X .

**Syntax:** lapply(X, FUN, …)

Where X is a vector (atomic or list) or an expression object and FUN is the function to be applied to each element of X

In this R program, we directly give the values to a built-in function lapply(). Here we are using the variable **values\_lst**for holding the nested list**.**And variable **rslt\_lst**is the result list after applying the function to each element of **values\_lst.**Call the function lapply() for applying the function FUN to list elements, FUN we used is a double square ' [[ ' which will extract one element from a list, here we are selecting the second element of each nested list.

**Algorithm:**

**STEP 1**: Assign variable **values\_lst**with a nested list

**STEP 2**: Print the original nested list

**STEP 3**: Call the apply function as lapply(values\_1st,’[[‘,2)for finding the second element of each list

**STEP 4**: Assign result list into the variable **rslt\_lst**

**STEP 4**: Print the result

**Source Code:**

values\_lst = list(list(1,3), list(2,5), list(6,7))

print("Original nested list is:")

print(values\_lst)

rslt\_lst = lapply(values\_lst, '[[', 2)

print("Second element of the nested list is:")

print(rslt\_lst)

**Output:**

[1] "Original nested list is:"

[[1]]

[[1]][[1]]

[1] 1

[[1]][[2]]

[1] 3

[[2]]

[[2]][[1]]

[1] 2

[[2]][[2]]

[1] 5

[[3]]

[[3]][[1]]

[1] 6

[[3]][[2]]

[1] 7

[1] "Second element of the nested list is:"

[[1]]

[1] 3

[[2]]

[1] 5

[[3]]

[1] 7

**EXPERIMENT 25:**

**Write a R program to merge two given lists into one list.**

**Description:**

An R program to merge two given lists into one list. Here we are using a built-in combine function **c()** for this. This function helps to combine its arguments to form a list in this, all the arguments are coerced to a common type and it is the type of the return value.

**Syntax:** c(…)

Where ... indicates object to be concatenate

In this R program, we directly give the values to a built-in function c(). Here we are using variables **lst1,lst2**for holding the list elements of two different types**.**Call the function c() for merging the two lists and assign them to a list variable **merge\_lst**.

**Algorithm:**

**STEP 1**: Assign variables **lst1,lst2** with a list of two different types

**STEP 2**: Print the original lists

**STEP 3**: Merge the two lists by calling the combine function as c(lst1,lst2)

**STEP 4**: Assign merged lists into the variable **merge\_lst**

**STEP 5**: Print the merged list **merge\_lst**

**Source Code:**

lst1 = list(5,2,1)

lst2 = list("Red", "Green", "Black")

print("Original lists:")

print(lst1)

print(lst2)

print("Merge the lists:")

merge\_lst = c(lst1, lst2)

print("New merged list:")

print(merge\_lst)

**Output:**

[1] "Original lists:"

[[1]]

[1] 5

[[2]]

[1] 2

[[3]]

[1] 1

[[1]]

[1] "Red"

[[2]]

[1] "Green"

[[3]]

[1] "Black"

[1] "Merge the lists:"

[1] "New merged list:"

[[1]]

[1] 5

[[2]]

[1] 2

[[3]]

[1] 1

[[4]]

[1] "Red"

[[5]]

[1] "Green"

[[6]]

[1] "Black"

**EXPERIMENT 26:**

**Write a R program to create a list named s containing sequence of 15 capital letters,**

**starting from ‘E’.**

**Description:**

We will get all letters in lowercase sequence by using the letters function

**Syntax:**

letters

**Example:** Return all lowercase letters using letters function

**# return all lower case letters in sequence**

**print(letters)**

**Output:**

[1] “a” “b” “c” “d” “e” “f” “g” “h” “i” “j” “k” “l” “m” “n” “o” “p” “q” “r” “s”

[20] “t” “u” “v” “w” “x” “y” “z”

We will get all letters in uppercase sequence by using LETTERS function.

**Syntax:**

LETTERS

**Example:** Return all uppercase letters using LETTERS function

# return all upper case letters in sequence

**print(LETTERS)**

**Output:**

[1] “A” “B” “C” “D” “E” “F” “G” “H” “I” “J” “K” “L” “M” “N” “O” “P” “Q” “R” “S”

[20] “T” “U” “V” “W” “X” “Y” “Z”

We can get the subsequence of letters using the index. Index starts with 1 and ends with 26 (since there are 26 letters from a to z). We are getting from letters/LETTERS function.

**Syntax:**

letters[start:end]

LETTERS[start:end]

Where, start is the starting letter index and end is the ending letter index.

**Example:** R program to get subsequence letters using index

# return all upper case letters from

# index 1 to index 12

**print(LETTERS[1:12])**

# return all lower case letters from

# index 1 to index 12

**print(letters[1:12])**

# return all upper case letters from

# index 5 to index 26

**print(LETTERS[5:26])**

# return all lower case letters from

# index 5 to index 26

**print(letters[5:26])**

**Output:**

[1] “A” “B” “C” “D” “E” “F” “G” “H” “I” “J” “K” “L”

[1] “a” “b” “c” “d” “e” “f” “g” “h” “i” “j” “k” “l”

[1] “E” “F” “G” “H” “I” “J” “K” “L” “M” “N” “O” “P” “Q” “R” “S” “T” “U” “V” “W” “X” “Y” “Z”

[1] “e” “f” “g” “h” “i” “j” “k” “l” “m” “n” “o” “p” “q” “r” “s” “t” “u” “v” “w” “x” “y” “z”

Random sequence of letters using sample() function

Here in this scenario, we will get the random letters randomly using sample() function. sample() function is used to generate random letters

**Syntax:**

sample(letters/LETTERS,size)

Where,

**letters/LETTERS** is a function that is a first parameter to display letters in lower/upper case

**size** is used to get the number of letters randomly

**Example:** R program to display random letters

sample(letters, 20)

**Output:**

[1] “k” “p” “g” “c” “j” “r” “s” “u” “h” “i” “d” “o” “a” “m” “y” “f” “t” “l” “q” “b”

**match()** function in [R Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is used to return the positions of the first match of the elements of the first vector in the second vector. If the element is not found, it returns NA.

**Syntax:** match(x1, x2, nomatch)

**Parameters:**  
**x1:** Vector 1  
**x2:** Vector 2  
**nomatch:** value to be returned in case of no match

**Algorithm:**

**STEP1:**Assign **l** as sequence of letters. **Step 2:**Sequence of Letters Starts from E and end with E+15**(T)** by calling **LETTERS [match("E", LETTERS):(match("E", LETTERS)+15)]**

**Step 3: Print Sequence of letters.**

**Source Code:**

l = LETTERS[match("E", LETTERS):(match("E", LETTERS)+15)]

print("Content of the list:")

print("Sequence of 15 capital letters, starting from ‘E’-")

print(l)

**Output:**

"Content of the list:"

"Sequence of 15 capital letters, starting from ‘E’-"

"E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"

**EXPERIMENT 27:**

**Write a R program to assign new names "a", "b" and "c" to the elements of a given list.**

**Description:**

**names()** function in [R Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is used to get or set the name of an Object. This function takes object i.e. vector, matrix or data frame as argument along with the value that is to be assigned as name to the object. The length of the value vector passed must be exactly equal to the length of the object to be named.

**Syntax:** names(x) <- value

**Parameters:**  
**x:** Object i.e. vector, matrix, data frame, etc.  
**value:** Names to be assigned to x

**Algorithm**

**STEP 1**: Assign variables **L1**with a list

**STEP 2**: Create **L1** with 3 sets of elements **g1,g2,g3**

**STEP 3**: Print the original lists

**STEP 4**: Add a new names **a,b,c** to the element of the list as  **names(L1) = c("a", "b", "c")**

**STEP 5**: Print the final list **L1**

**Source Code:**

list1 = list(g1 = 1:10, g2 = "R Programming", g3 = "HTML")

print("Original list:")

print(list1)

names(list1) = c("a", "b", "c")

print("Assign new names 'a', 'b' and 'c' to the elements of the said list")

print(list1)

**Output:**

"Original list:"

$g1

1 2 3 4 5 6 7 8 9 10

$g2

"R Programming"

$g3

"HTML"

"Assign new names 'a', 'b' and 'c' to the elements of the said list"

$a

1 2 3 4 5 6 7 8 9 10

$b

"R Programming"

$c

"HTML"

**EXPERIMENT 28:**

**Write a R program to find the levels of factor of a given vector.**

**Description:**

An R program to find the levels of a factor of a given vector. Here we are using built-in functions **levels(factor())** for this calculation. The vector values are passed to these functions directly here. The **levels(factor()** function in R computes the levels of factors of the vector in a single function.

Using the function **factor()**we can create a factor of the vector and by using the **level()** function we can find levels of a factor. Factors are stored as integer vectors and which is closely related to vectors.

In the R program to find the levels of a factor of a given vector. In this R program, we directly give the values to built-in functions. And print the function result. Here we used variable **v** for assigning vector values.

**Algorithm:**

**STEP 1**: Assign variable **v** with vector values

**STEP 2**: Use the built-in functions

**STEP 3**: First print original vector values

**STEP 4**: levels(factor(v) with an argument as **v** to find levels of factors

**STEP 5**: print the result of the function

**Source Code:**

v = c(1, 2, 3, 3, 4, NA, 3, 2, 4, 5, NA, 5)

print("Original vector:")

print(v)

print("Levels of factor of the said vector:")

print(levels(factor(v)))

**Output:**

"Original vector:"

1 2 3 3 4 NA 3 2 4 5 NA 5

"Levels of factor of the said vector:"

"1" "2" "3" "4" "5"

**EXPERIMENT 29:**

**Write a R program to create an ordered factor from data consisting of the names of**

**months.**

**Description:**

The **table()** function in R can be used to quickly create frequency tables.

This tutorial provides examples of how to use this function with the following data frame in R:

#create data frame

df <- data.frame(player = c('AJ', 'Bob', 'Chad', 'Dan', 'Eric', 'Frank'),

position = c('A', 'B', 'B', 'B', 'B', 'A'),

points = c(1, 2, 2, 1, 0, 0))

#view data frame

df

player position points

1 AJ A 1

2 Bob B 2

3 Chad B 2

4 Dan B 1

5 Eric B 0

6 Frank A 0

### **Example 1: Frequency Table for One Variable**

The following code shows how to create a frequency table for the **position** variable in our data frame:

#calculate frequency table for position variable

table(df$position)

A B

2 4

From the output we can observe:

* 2 players in the data frame have a position of ‘**A**‘
* 4 players in the data frame have a position of ‘**B**‘

### **Example 2: Frequency Table of Proportions for One Variable**

The following code shows how to use **prop.table()** to create a frequency table of proportions or the **position** variable in our data frame:

#calculate frequency table of proportions for position variable

prop.table(table(df$position))

A B

0.3333333 0.6666667

From the output we can observe:

* 33.33% of players in the data frame have a position of ‘**A**‘
* 66.67% of players in the data frame have a position of ‘**B**‘

Note that in a proportion table the sum of the proportions will always be equal to 1.

### **Example 3: Frequency Table for Two Vaiables**

### The following code shows how to create a frequency table for the  **position**  and  **points**variable in our data frame:

#calculate frequency table for position and points variable

table(df$position, df$points)

0 1 2

A 1 1 0

B 1 1 2

From the output we can observe:

* 1 player in the data frame has a position of ‘**A**‘ and **0** points
* 1 player in the data frame has a position of ‘**A**‘ and **1** point
* 0 players in the data frame have a position of ‘**A**‘ and **2** points
* 1 player in the data frame has a position of ‘**B**‘ and **0** points
* 1 player in the data frame has a position of ‘**B**‘ and **1** point
* 2 players in the data frame have a position of ‘**B**‘ and **2** points

### **Example 4: Frequency Table of Proportions for Two Variables**

### The following code shows how to create a frequency table of proportions for the **position**  and **points**variable in our data frame:

#calculate frequency table of proportions for position and points variable

prop.table(table(df$position, df$points))

0 1 2

A 0.1666667 0.1666667 0.0000000

B 0.1666667 0.1666667 0.3333333

From the output we can observe:

* 16.67% of players in the data frame have a position of ‘**A**‘ and **0** points
* 16.67% of players in the data frame have a position of ‘**A**‘ and **1** point
* 0% of players in the data frame have a position of ‘**A**‘ and **2** points
* 16.67% of players in the data frame have a position of ‘**B**‘ and **0** points
* 16.67% of players in the data frame have a position of ‘**B**‘ and **1** point
* 33.3% of players in the data frame have a position of ‘**B**‘ and **2** points

Note that we can also use the **options()** function to specify how many decimals to show in the proportion table:

#only display two decimal places

options(digits=2)

#calculate frequency table of proportions for position and points variable

prop.table(table(df$position, df$points))

0 1 2

A 0.17 0.17 0.00

B 0.17 0.17 0.33

**Algorithm:**

**STEP 1**: Assign variable **v** with vector values

**STEP 2**: First print original vector values

**STEP 3**: Use the built-in functions

**STEP 4**: factor(mons\_v) value assign to f to print levels factors of the vector

**STEP 5**: print levels with table format by using **table(f)**

**Source Code:**

mons\_v = c("March","April","January","November","January",

"September","October","September","November","August","February",

"January","November","November","February","May","August","February",

"July","December","August","August","September","November","September",

"February","April")

print("Original vector:")

print(mons\_v)

f = factor(mons\_v)

print("Ordered factors of the said vector:")

print(f)

print(table(f))

**Output:**

"Original vector:"

"March" "April" "January" "November" "January" "September"

"October" "September" "November" "August" "February" "January"

"November" "November" "February" "May" "August" "February"

"July" "December" "August" "August" "September" "November"

"September" "February" "April"

"Ordered factors of the said vector:"

March April January November January September October

September November August February January November November

February May August February July December August

August September November September February April

11 Levels: April August December February January July March May ...September

f

April August December February January July March May

2 4 1 4 3 1 1 1

November October September

5 1 4

**EXPERIMENT 30:**

**Write a R program to concatenate two given factor in a single factor.**

**Description:**

An R program to concatenate two given factors into a single factor. We can accomplish the concatenation using built-in functions such as levels() and factor(). In this case, the vector values are passed directly to these functions.

Using the function factor() we can create a factor of the vector and by using the level() function we find the levels of a factor. Factors are stored as integer vectors and are closely related to vectors.

**Syntax:**

levels(x) #where x is an object, for example, a factor

factor(x = character(), levels, labels = levels,exclude = NA, ordered = is.ordered(x), nmax = NA)

#Where x is a vector of data, usually taking a small number of distinct values

In this R program, we directly give the values to built-in functions. And print the function result. Here we used two variables namely fact1 and fact2 for assigning factor values. The third variable fact contains the concatenated factor and finally prints the resulting factor.

**Algorithm:**

**STEP 1**: Assign variable **fact1,fact2**with factor values

**STEP 2**: First print original factors values

**STEP 3**:Call the built-in function factor with level as **factor(c(levels(fact1)[fact1], levels(fact2)[fact2]))**

**STEP 4**: Assign variable **fact**with the function result

**STEP 5**: Print the concatenated factor

**Source Code:**

fact1 <- factor(sample(LETTERS, size=6, replace=TRUE))

fact2 <- factor(sample(LETTERS, size=6, replace=TRUE))

print("Original factors are:")

print(fact1)

print(fact2)

fact= factor(c(levels(fact1)[fact1], levels(fact2)[fact2]))

print("After concatenate factor becomes:")

print(fact)

**Output:**

[1] "Original factors are:"

[1] Q Y M J J H

Levels: H J M Q Y

[1] B J L S F Z

Levels: B F J L S Z

[1] "After concatenate factor becomes:"

[1] Q Y M J J H B J L S F Z

Levels: B F H J L M Q S Y Z