R Programming Lab

II B.Tech - II Semester

2021-22

Prepared by

K S R Prasad
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

D.N.R.COLLEGE OF ENGINEERING & TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to JNTUK, Kakinada &

Accredited with NAAC B++ Grade)

Balusumudi, Bhimavaram - 534 202 W. G. Dist.

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:

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.

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:

```
> name = readline(prompt="Input your name: ")
Input your name:  surya
```

```
> age =  readline(prompt="Input your age: ")
Input your age:  33
> print(paste("My name is",name, "and I am",age ,"years old."))
[1] "My name is surya and I am 33 years old."
> print(R.version.string)
[1] "R version 3.4.4 (2018-03-15)"
```

---

2. Write a R program to get the details of the objects in memory

Description:

In R, after the creation of the object, It will allocate some space to the particular object. The object in memory is stored in bytes. Numeric type can allocate 56 bytes and character type can allocate 112 bytes.

In 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.

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 = readline(prompt="Input your name: ")  // name= “R Programming”;
n1= readline(prompt="n1 value is: ")  // n1=20;
n2= readline(prompt="n2 value is: ")  // n2=4;
```

```
nums = c(10, 20, 30, 40, 50, 60)
print(ls())
print("Details of the objects in memory:")
print(ls.str())
```

---

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:

The simplest way to create a sequence of numbers in R is by using the : operator. Type 1:20

```
1:20
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

The most basic use of seq() does exactly the same thing as the : operator.

```
seq(1, 20)
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Generating a sequence in R using the function seq() is vital and has many uses in data analysis. You can generate a particular general sequence from specifying beginning and end numbers as well. We can use the seq() function to generate the sequences.

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:

```
> print("Sequence of numbers from 20 to 50:")
[1] "Sequence of numbers from 20 to 50:"
> print(seq(20,50))
 [1] 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
[26] 45 46 47 48 49 50
> print("Mean of numbers from 20 to 60:")
[1] "Mean of numbers from 20 to 60:"
> print(mean(20:60))
[1] 40
> print("Sum of numbers from 51 to 91:")
[1] "Sum of numbers from 51 to 91:"
> print(sum(51:91))
[1] 2911
```

---

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:

The basic syntax to create a bar-chart in R is − barplot(H,xlab,ylab,main, names.arg,col)

Following is the description of the parameters used −

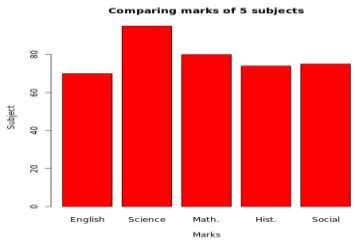
- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.

- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

Source Code:

```
marks = c(70, 95, 80, 74, 75)
barplot(marks,
main = "Comparing marks of 5 subjects",
xlab = "Marks",
ylab = "Subject",
names.arg = c("English", "Science", "Math.", "Hist.", "Social"),
col = "red",
horiz = FALSE)
```

Output:



---

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

Description:

To display distinct values in a vector in R Programming Language.

Method 1: Using unique()

For this, the vector from which distinct elements are to be extracted is passed to the unique() function. The result will give all distinct values in a vector.

Syntax:

unique(vector_name)

Where, vector_name is the input vector.

Method 2: Using duplicated()

By using this method we can get duplicate values. So, if we want to get unique values, we can implement this function along with **!** Operator. This will do exactly the reverse of the duplicated() function.

Syntax:

!duplicated(vector_data)

This will return boolean values.

In order to return actual values, we can use index operator — []

Syntax:

vector_name[!duplicated(vector_name)]

Source C ode:

```
str1 = c("CSE"," ECE","CSE")

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:

```
[1] "Original vector(string)"
[1] "CSE"  " ECE" "CSE"
```

```
[1] "Unique elements of the said vector:"
[1] "cse"  " ece"
[1] "Original vector(number)"
[1] 1 2 2 3 4 4 5 6
[1] "Unique elements of the said vector:"
[1] 1 2 3 4 5 6
```

---

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: Matrix is a two dimensional data structure in R programming. Matrix is similar to vector but additionally contains the dimension attribute. All attributes of an object can be checked with the attributes() function (dimension can be checked directly with the dim() function).'

How to create a matrix in R programming?

Matrix can be created using the `matrix()` function. Dimension of the matrix can be defined by passing appropriate value for arguments `nrow` and `ncol`. Providing value for both dimension is not necessary. If one of the dimension is provided, the other is inferred from length of the data.

```
> cbind(c(1,2,3),c(4,5,6))
[,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> rbind(c(1,2,3),c(4,5,6))
[,1] [,2] [,3]
```

```
[1,]    1    2    3
[2,]    4    5    6
```

Source C ode:

```
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:

```
[1] "Content of the said matrix:"
     a b c
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

---

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 2 × 2 matrix with labels and fill the matrix by columns.

Source C ode:

```
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:

```
[1] "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
```

---

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.

Source C ode:

```
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"
```

---

9) Write a R program to create a two-dimensional 5x3 array of sequence of even integers greater than 50.

Method 1 : Using seq() method

The array() method can be used to create an array with >=1 dimensions. This method returns an array with the extents specified in the dim attribute. For a two-dimensional array, the dim attribute contains a vector of two elements, first indicating the number of rows and then the number of columns respectively.

Syntax: *array(data , dim)*

Parameter :

- *data – The data to fill into the array*
- *dim – The dimensions of the matrix in the form of vector*

Method 2: Manually using for loop

The starting position to begin the sequence of data is specified along with the desired dimensions of the 2-D array. Since the total number of required elements is equal to the product of dimensions, but here we need even numbers that occur at alternate positions. So, the total number of elements will be equivalent to double the product of dimensions. Now, the ending position is given by :

ending_position = starting_position + number of elements

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)
```

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
```

---

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.

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
```

11. Aim: Write a R program to create an empty data frame.

Description:

Method: 1

An empty data frame can also be created with or without specifying the column names and column types to the data values contained within it. data.frame() method can be used to create a data frame, and we can assign the column with the empty vectors. Column type checking with zero rows is supported by the data frames, where in we define the data type of each column prior to its creation.

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
```

Method: 2

Description:

We first create a matrix with both rows and columns and then convert it to a data frame. A data frame and matrix are easily inter-convertible to each other, we first create a matrix with both rows and columns equivalent to 0 and then convert it to a data frame. The dimensions of the equivalent data frame are 0. Time complexity incurred in the creation of an empty data frame is O(1), since a constant time is required.

Source Code:

```
# creating a matrix with 0 rows
# and columns
mat = matrix(ncol = 0, nrow = 0)
# converting the matrix to data
# frame
df=data.frame(mat)
print ("Data Frame")
print (df)
# check for the dimensions of data
# frame
print("Dimensions of data frame")
dim(df)
```

Output:

```
[1] "Data Frame"
data frame with 0 columns and 0 rows
[1] "Dimensions of data frame"
[1] 0 0
```

---

12. Aim: Write a R program to create a data frame from four given vectors.

Description:

In this aprogram we will see how to create a Dataframe from four given vectors in R. To create a data frame in R using the vector, we must first have a series of vectors containing data. The data.frame() function is used to create a data frame from vector in R.

Syntax: data.frame(vectors)

Source Code:

```
name = c('Divya', 'Rani', 'Pavan', 'Surya', 'Ambika')
score = c(82.5, 90, 86.5, 95, 92)
attempts = c(1, 3, 2, 3, 2)
qualify = c('yes', 'no', 'yes', 'no', 'Yes')
print("Original data frame:")
df = data.frame(name, score, attempts, qualify)
print(df)
```

Output:

```
> name = c('Divya', 'Rani', 'Pavan', 'Surya', 'Ambika')
> score = c(82.5, 90, 86.5, 95, 92)
> attempts = c(1, 3, 2, 3, 2)
> qualify = c('yes', 'no', 'yes', 'no', 'Yes')
> print("Original data frame:")
[1] "Original data frame:"
>
> df = data.frame(name, score, attempts, qualify)
> print(df)
   name score attempts qualify
1  Divya  82.5        1     yes
2   Rani  90.0        3      no
3  Pavan  86.5        2     yes
4  Surya  95.0        3      no
5 Ambika  92.0        2     Yes
```

OR

Source Code:

```
# creating a vectors with some value
id = c(1, 2, 3)
name = c("karthik" , "nikhil" , "sravan")
branch = c("IT" , "CSE" , "IT")
favourite_subject = c("SE" ,"DAA" , "OS")
 # passing the vectors into data.frame() function
# as parameters
df1=data.frame(id, name, branch, favourite_subject)
print(df1)
```

Output:

```
> # creating a vectors with some value
> id = c(1, 2, 3)
> name = c("karthik" , "nikhil" , "sravan")
> branch = c("IT" , "CSE" , "IT")
> favourite_subject = c("SE" ,"DAA" , "OS")
> # passing the vectors into data.frame() function
> # as parameters
> df1=data.frame(id, name, branch, favourite_subject)
> print(df1)
  id    name branch favourite_subject
1  1 karthik     IT              SE
2  2  nikhil    CSE             DAA
3  3  sravan     IT              OS
>
```

---

13. Aim: 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:

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. The syntax of this function is,

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.

Below are the steps used in the 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.

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)**

Sorce Code:

```
a = c(10,20,10,10,40,50,20,30)
b = c(10,30,10,20,0,50,30,30)
print("Original data frame:")
ab = data.frame(a,b)
print(ab)
print("Duplicate elements of the said data frame:")
print(duplicated(ab))
print("Unique rows of the said data frame:")
print(unique(ab))
```

Output:

```
> a = c(10,20,10,10,40,50,20,30)
> b = c(10,30,10,20,0,50,30,30)
> print("Original data frame:")
[1] "Original data frame:"
> ab = data.frame(a,b)
> print(ab)
  a  b
1 10 10
2 20 30
3 10 10
4 10 20
5 40  0
6 50 50
7 20 30
8 30 30
> print("Duplicate elements of the said data frame:")
[1] "Duplicate elements of the said data frame:"
> print(duplicated(ab))
[1] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
> print("Unique rows of the said data frame:")
[1] "Unique rows of the said data frame:"
> print(unique(ab))
  a  b
1 10 10
2 20 30
4 10 20
5 40  0
6 50 50
8 30
```

---

14) Aim: Write a R program to save the information of a data frame in a file and display the information of the file.

Description:

Source Code:

```
exam_data = data.frame(

name = c('RAM', 'KUMAR', 'BABU', 'MOHAN', 'RAJU'),

score = c(12.5, 9, 16.5, 12, 9),

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

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

)

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          RAM          12.5       1     yes

2         KUMAR        9.0       3     no

3         BABU          16.5       2     yes

4        MOHAN         12.0       3      no
```

```
5       RAJU            9.0       2      no

      size isdir mode               mtime               ctime

data.rda  281 FALSE  644 2022-04-20 05:47:43 2022-04-20 05:47:43

                  atime  uid  gid   uname  grname

data.rda 2022-04-20 05:47:43 1000 1000 trinket trinket
```

---

15) Aim: Write a R program to create a matrix from a list of given vectors.

Description:

In R programming language, **vector** is a basic object which consists of homogeneous 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:

- **matrix()** function
- **cbind()** function
- **rbind()** function

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:

```
> l = list()
> for (i in 1:5) l[[i]] <- c(i, 1:4)
> print("List of vectors:")
[1] "List of vectors:"
> print(l)
[[1]]
[1] 1 1 2 3 4
[[2]]
[1] 2 1 2 3 4
[[3]]
[1] 3 1 2 3 4
[[4]]
[1] 4 1 2 3 4
[[5]]
[1] 5 1 2 3 4
> result = do.call(rbind, l)
> print("New Matrix:")
[1] "New Matrix:"
> print(result)
     [,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
```

---

16. Aim: Write a R program to concatenate two given matrices of same column but different rows.

Description:

Given two matrices **A** and **B** of size **M x N**, the task is to concatenate them into one matrix. **Concatenation of matrix:** The process of appending elements of every row of the matrix one after the other is known as the Concatenation of matrix.

Examples:

Input:

A[] [] = {{1, 2},
{3, 4}},
B[] [] = {{5, 6},
{7, 8}}

Output:

1 2 5 6
3 4 7 8

Explanation:

Elements of every row of the matrix B
is appended into the row of matrix A.

Source Code:

```
x = matrix(1:12, ncol=3)
y = matrix(13:24, ncol=3)
print("Matrix-1")
print(x)
print("Matrix-2")
print(y)
result = dim(rbind(x,y))
print("After concatenating two given matrices:")
print(result)
```

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 two given matrices:"
[1] 8 3
```

---

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

Description:

Finding Maximum value:

- In the code below, we have created a sample matrix, in which we have passed "**nrow=3**"(matrix will have only 3 rows) in example 1 and "**ncol=2**"(matrix will have only 2 columns) in example 2.
- Then we have printed the sample matrix in the next line with the message "Sample Matrix".
- Then we have used the syntax below to find the row and column number of the maximum element and stored it in the variable "max". **We have made use of the 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.
- **The** "**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)*

Finding Minimum value:

- In the code below, we have created a sample matrix, in which we have passed "**nrow=3**"(matrix will have only 3 rows) in example 1 and "**ncol=8**"(matrix will have only 8 columns) in example 2 as a parameter while defining the matrix.
- Then we have printed the sample matrix in the next line with the message "Sample Matrix".
- Then we have used the syntax below to find the row and column number of the minimum element and stored it in the variable "min". **We have made use of the 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.
- **The "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)*

Source Code:

```
# defining a sample matrix
m = matrix(c(1:16), ncol = 2)
print("Sample Matrix:")
print(m)
# stores indexes of max value
max = which(m == max(m), arr.ind=TRUE)
print(paste("Maximum value: ",m[max]))
print(max)
```

Output:

```
[1] "Sample Matrix:"
     [,1] [,2]
[1,]    1    9
[2,]    2   10
[3,]    3   11
[4,]    4   12
[5,]    5   13
```

```
[6,]    6   14
[7,]    7   15
[8,]    8   16
[1] "Maximum value:  16"
     row col
[1,]   8   2
```

Source Code:

```
# defining a sample matrix
m = matrix(c(11, 20, 13, -9, 1, 99, 36, 81, 77), nrow = 3)
print("Sample Matrix:")
print(m)
# stores indexes of min value
min = which(m == min(m), arr.ind = TRUE)
print(paste("Minimum value: ", m[min]))
print(min)
```

Output:

```
[1] "Sample Matrix:"
     [,1] [,2] [,3]
[1,]   11   -9   36
[2,]   20    1   81
[3,]   13   99   77
[1] "Minimum value:  -9"
     row col
[1,]   1   2
```

---

18) Aim: Write a R program to append value to a given empty vector.

Description:

To create an empty vector and add elements into a vector in R Programming Language. An empty vector can be created by simply not passing any value while creating a regular vector using the **c()** function.

Syntax: c()

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:

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

---

19) Aim: Write a R program to multiply two vectors of integers type and length 3.

Description:

To write an R program to multiply two vectors. We can give two vector values to calculate the multiplication. For the calculation of vector product here we use the product(*) operator. Given below are the steps which are used in the R program to multiply two vectors. In this R program, we accept the vector values into variables **A** and **B**. The result value of vectors is assigned variable **C**.Finally the variable **C** is printed as output vector.

ALGORITHM

STEP 1: take the two vector values into the variables **A,B**

STEP 2: Consider **C** 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 **C** as **C=A*B**

STEP 6: print the vector **C** 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
u = x * y
print(z)
print(u)
```

Output:

```
[1] "Original Vectors:"
[1] 10 20 30
[1] 20 10 40
[1] "Product of two Vectors:"
[1] 0.50 2.00 0.75
[1]  200  200 1200
```

---

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

Description:

sum(), mean(), and **prod()** methods are available in R which are used to compute the specified operation over the arguments specified in the method. In case, a single vector is specified, then the operation is performed over individual elements, which is equivalent to the application of for loop.

Function Used:

- mean() function is used to calculate mean

Syntax: *mean(x, na.rm)*

Parameters:

x: *Numeric Vector*

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

- *sum() is used to calculate sum*

Syntax: *sum(x)*

Parameters:

x: *Numeric Vector*

- *prod() is used to calculate product*

Syntax: *prod(x)*

Parameters:

- ***x:*** *Numeric Vector*

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:

```
[1] "Sum:"
[1] 60
[1] "Mean:"
[1] 20
[1] "Product:"
[1] 6000
```

---

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:

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

```
# Create a list containing strings, numbers, vectors and a logical
# values.
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
```

[1] 119.1

Naming List Elements

The list elements can be given names and they can be accessed using these names.

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

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

When we execute the above code, it produces the following result −

$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3

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)
```

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] "List with column names:"
$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"
```

---

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:

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example −

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
  list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Access the first element of the list.
```

```
print(list_data[1])

# Access the thrid element. As it is also a list, all its elements will be printed.
print(list_data[3])

# Access the list element using the name of the element.
print(list_data$A_Matrix)
```

When we execute the above code, it produces the following result −

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3

[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8
```

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:

```
[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] "List with column names:"
$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"



[1] "1st element:"
$Color
[1] "Red"   "Green" "Black"

[1] "2nd element:"
$`Odd numbers`
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    3    7   11
```

---

23) Aim: Write a R program to create a list containing a vector, a matrix and a list and remove the second element.

Description:

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Add element at the end of the list.
list_data[4] <- "New element"
print(list_data[4])

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

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

# Update the 3rd Element.
list_data[3] <- "updated element"
print(list_data[3])
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] "New element"

$<NA>
NULL

$`A Inner list`
[1] "updated element"
```

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"
```

---

24) Aim: Write a R program to select second element of a given nested list.

Description:

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` . The syntax of this function is

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

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_lst, '[[', 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:

```
x = list(list(0,2), list(3,4), list(5,6))
print("Original nested list:")
print(x)
e = lapply(x, '[[', 2)
print("Second element of the nested list:")
print(e)
```

Output:

```
[1] "Original nested list:"
[[1]]
[[1]][[1]]
[1] 0

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

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

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

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

[[3]][[2]]
[1] 6
```

```
[1] "Second element of the nested list:"
[[1]]
[1] 2

[[2]]
[1] 4
[[3]]
[1] 6
```

---

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

Description:

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, The syntax of this function is

Syntax: c(...) # Where ... indicates object to be concatenate

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:

```
n1 = list(1,2,3)
c1 = list("Red", "Green", "Black")
print("Original lists:")
print(n1)
print(c1)
```

```
print("Merge the said lists:")
mlist =  c(n1, c1)
print("New merged list:")
print(mlist)
```

Output:

```
[1] "Original lists:"
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] "Black"

[1] "Merge the said lists:"
[1] "New merged list:"
[[1]]
[1] 1

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

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

[[4]]
[1] "Red"

[[5]]
[1] "Green"

[[6]]
[1] "Black"
```

---

26) Aim: Write a R program to create a list named s containing sequence of 15 capital letters, starting from 'E'.

Description:

Return all upper case letters using LETTERS function

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

Syntax:

LETTERS

Subsequence letters using range operation

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]

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:

```
[1] "Content of the list:"
[1] "Sequence of 15 capital letters, starting from ‘E’-"
 [1] "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"
```

---

27) Aim: Write a R program to assign new names "a", "b" and "c" to the elements of a given list.

Description:

A list is an object in R Language which consists of heterogeneous elements. A list can even contain matrices, data frames, or functions as its elements. The list can be created using **list()** function in R. Named list is also created with the same function by specifying the names of the elements to access them. Named list can also be created using **names()** function to specify the names of elements after defining the list. In this article, we’ll learn to create named list in R using two different methods and different operations that can be performed on named lists.

Syntax: *names(x) <- value*

Parameters:

x: *represents an R object*

value: *represents names that has to be given to elements of* ***x*** *object*

Source Code:

```
list1 = list(g1 = 1:10, g2 = "R Programming", g3 = "HTML")
print("Original list:")
print(list1)
names(list1) = c("one", "two", "three")
print("Assign new names 'one', 'two' and 'three' to the elements of the said list")
print(list1)
```

Output:

```
[1] "Original list:"
$g1
```

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

$g2
[1] "R Programming"

$g3
[1] "HTML"

[1] "Assign new names 'one', 'two' and 'three' to the elements of the said list"
$one
 [1]  1  2  3  4  5  6  7  8  9 10

$two
[1] "R Programming"

$three
[1] "HTML"
```

---

28) Aim: Write a R program to find the levels of factor of a given vector.

Description:

Factors are the objects which are used to categorize the data and display the data as levels. The objects can be integer, character. They can be used to find the unique values in the given vector. The resulting data is known as levels. Factors are useful in statistical analysis in analyzing the categorical data. It is used to find the levels of the given vector. In this article, we are going to find the levels of factor in the given vector in R.

Example:

Input:

data=[female,female,male,male,other]

Output:

factor=female,male,other.

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:

```
[1] "Original vector:"
 [1]  1  2  3  3  4 NA  3  2  4  5 NA  5
[1] "Levels of factor of the said vector:"
[1] "1" "2" "3" "4" "5"
```

---

29) Aim: Write a R program to create an ordered factor from data consisting of the names of months.

Description:

R – Level Ordering of Factors

Factors are data objects used to categorize data and store it as levels. They can store a string as well as an integer. They represent columns as they have a limited number of unique values. Factors in R can be created using **factor()** function. It takes a vector as input. **c() function** is used to create a vector with explicitly provided values.

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:

```
[1] "Original vector:"
 [1] "March"     "April"     "January"   "November"  "January"   "September"
 [7] "October"   "September" "November"  "August"    "February"  "January"
[13] "November"  "November"  "February"  "May"       "August"    "February"
[19] "July"      "December"  "August"    "August"    "September" "November"
[25] "September" "February"  "April"
[1] "Ordered factors of the said vector:"
 [1] March     April     January   November  January   September October
 [8] September November  August    February  January   November  November
[15] February  May       August    February  July      December  August
[22] 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
```

---

30) Aim: Write a R program to concatenate two given factor in a single factor.

Description:

Below are the steps used in the R program to concatenate two given factors into a single factor. 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:

```
f1 <- factor(sample(LETTERS, size=6, replace=TRUE))
f2 <- factor(sample(LETTERS, size=6, replace=TRUE))
print("Original factors:")
print(f1)
print(f2)
f = factor(c(levels(f1)[f1], levels(f2)[f2]))
print("After concatenate factor becomes:")
print(f)
```

Output:

```
[1] "Original factors:"
[1] Q V O P L B
Levels: B L O P Q V
[1] Z H V Y S K
Levels: H K S V Y Z
[1] "After concatenate factor becomes:"
 [1] Q V O P L B Z H V Y S K
Levels: B H K L O P Q S V Y Z
```