20IT53-R-PROGRAMMING LAB WEEK-3& WEEK-4

Week-3

a) Implement R Script to create a list.
b) Implement R Script to access elements in the list.
c) Implement R Script to merge two or more lists.
Implement R Script to perform matrix operation.

3(a) Implement R Script to create a list.

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 logical values
list_data <- list("Red","Green",c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

Output:

```
print(list_data)
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
[1] 119.1
```

3(b) Implement R Script to access elements in the list.

Giving a name to list elements

There are only three steps to print the list data corresponding to the name:

1. Creating a list.
2. Assign a name to the list elements with the help of names() function.
3. Print the list data.

Example: 1

```
# 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)
```

Output:

```
print(list_data)
$`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
```

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.

Example:2

```
# 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)
```

Output:

```
print(list_data[1])
$`1st Quarter`
[1] "Jan" "Feb" "Mar"
# Access the third element. As it is also a list, all its elements will be printed.

print(list_data[3])
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"
$`A Inner list`[[2]]
[1] 12.3
```

```
# Access the list element using the name of the element.
  print(list_data$A_Matrix)
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
```

3(c) Implement R Script to merge two or more lists.
Implement R Script to perform matrix operation.

Implement R Script to merge two or more lists.

Merging Lists

You can merge many lists into one list by placing all the lists inside one list() function.

```
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")

# Merge the two lists.
merged.list <- c(list1,list2)

# Print the merged list.
print(merged.list)
```

Output:

```
print(merged.list)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

[[6]]
[1] "Tue"
```

Implement R Script to perform matrix operation.

R Matrix

In R, a two-dimensional rectangular data set is known as a matrix. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation.

In the R matrix, elements are arranged in a fixed number of rows and columns. The matrix elements are the real numbers.

A Matrix is created using the **matrix()** function.

Syntax

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

Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Example

```
#Arranging elements sequentially by row.
P <- matrix(c(5:16), nrow = 4, byrow = TRUE)
print(P)

# Arranging elements sequentially by column.
Q <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(Q)

# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3")

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

Output:

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

print(Q)
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14

print(R)
     col1 col2 col3
row1    3    4    5
```

```
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

Accessing Elements of a Matrix
Elements of a matrix can be accessed by using the column and row index of the element.

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
```

# Create the matrix. Arranging elements sequentially by row.
```
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```
# Access the element at 3rd column and 1st row.
```
print(P[1,3])
```
# Access the element at 2nd column and 4th row.
```
print(P[4,2])
```

# Access only the 2nd row.
```
print(P[2,])
```
# Access only the 3rd column.
```
print(P[,3])
```

Output:
```
print(P)
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14

print(P[1,3])
[1] 5

print(P[4,2])
[1] 13

print(P[2,])
col1 col2 col3
   6    7    8
print(P[,3])
row1 row2 row3 row4
   5    8   11   14
```

Matrix operations
In R, we can perform the mathematical operations on a matrix such as addition, subtraction, multiplication, etc.
```
R <- matrix(c(5:16), nrow = 4,ncol=3)
S <- matrix(c(1:12), nrow = 4,ncol=3)
```

# Display two matrices R and S
print(R)
print(S)

#Addition
sum<-R+S
print(sum)

#Subtraction
sub<-R-S
print(sub)

#Multiplication
mul<-R*S
print(mul)

#Division
div<-R/S
print(div)

Output:
print(R)
```
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
[4,]    8   12   16
```
print(S)
```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

sum<-R+S
print(sum)
```
     [,1] [,2] [,3]
[1,]    6   14   22
[2,]    8   16   24
[3,]   10   18   26
[4,]   12   20   28
```

sub<-R-S
print(sub)
```
     [,1] [,2] [,3]
[1,]    4    4    4
```

```
[2,]    4    4    4
[3,]    4    4    4
[4,]    4    4    4
```

mul<-R*S

```
print(mul)
     [,1] [,2] [,3]
[1,]    5   45  117
[2,]   12   60  140
[3,]   21   77  165
[4,]   32   96  192
```

div<-R/S

```
print(div)
         [,1]     [,2]     [,3]
[1,] 5.000000 1.800000 1.444444
[2,] 3.000000 1.666667 1.400000
[3,] 2.333333 1.571429 1.363636
[4,] 2.000000 1.500000 1.333333
```

Week-4

Implement R script to perform following operations:

a) Various operations on vectors.
b) Finding the sum and average of given numbers using arrays.
c) To display elements of list in reverse order.
d) Finding the minimum and maximum elements in the array.

4(a) Implement R script to perform various operations on vectors.

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

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

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

How to create a vector in R?

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

There are various other ways to create a vector in R, which are as follows:

1) Using the colon(:) operator

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

1. z**<-x:y**

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

Example:

```
A <- 4: -10
A
```

Output

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

2) Using the seq() function

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

Example

```
numbers <- seq(from = 0, to = 100, by = 20)
numbers
```

Output:

```
[1]  0  20  40  60  80 100
```

Note: The seq() function has three parameters: from is where the sequence starts, to is where the sequence stops, and by is the interval of the sequence.

Atomic vectors in R

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

Example

```
# Vector of strings
fruits <- c("banana", "apple", "orange")
# Print fruits
fruits
```

Output:

```
[1] "banana" "apple" "orange"
```

In this example, we create a vector that combines numerical values:

Example

```
# Vector of numerical values
numbers <- c(1, 2, 3)
# Print numbers
numbers
```

Output:

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

To create a vector with numerical values in a sequence, use the : operator:

Example

```
# Vector with numerical values in a sequence
numbers <- 1:10
numbers
```

Output:

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

Atomic vectors in R

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

1. Numeric vector

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

Example:

```
d<-45.5
num_vec<-c(10.1, 10.2, 33.2)
d
num_vec
class(d)
class(num_vec)
```
Output:
```
[1] 10.1 10.2 33.2
[1] "numeric"
[1] "numeric"
```

2. Integer vector

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

Example:
```
d<-as.integer(5)
e<-5L
int_vec<-c(1,2,3,4,5)
int_vec<-as.integer(int_vec)
int_vec1<-c(1L,2L,3L,4L,5L)
class(d)
class(e)
class(int_vec)
class(int_vec1)
```
Output:
```
[1] "integer"
[1] "integer"
[1] "integer"
[1] "integer"
```

3. Character vector

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

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

Example:
```
d<-'shubham'
e<-"Arpita"
f<-65
f<-as.character(f)
d
e
f
char_vec<-c(1,2,3,4,5)
char_vec<-as.character(char_vec)
char_vec1<-c("shubham","arpita","nishka","vaishali")
char_vec
class(d)
class(e)
class(f)
class(char_vec)
```

```
    class(char_vec1)
```
Output:

```
> d
[1] "shubham"
> e
[1] "Arpita"
> f
[1] "65"
> char_vec
[1] "1" "2" "3" "4" "5"
> class(d)
[1] "character"
> class(e)
[1] "character"
> class(f)
[1] "character"
> class(char_vec)
[1] "character"
> class(char_vec1)
[1] "character"
```

Accessing elements of vectors

We can access the elements of a vector with the help of vector indexing. Indexing denotes the position where the value in a vector is stored. Indexing will be performed with the help of integer, character, or logic.

1) Indexing with integer vector

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

Example:

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

Output

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

2) Indexing with a character vector

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

Example:

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

Output

```
shubham   arpita vaishali
```

```
  22     23    25
arpita
    23
```

3) Indexing with a logical vector

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

Example:

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

Output

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

Vector Operation

In R, there are various operation which is performed on the vector. We can add, subtract, multiply or divide two or more vectors from each other.

1) Combining vectors

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

Example:

```
p <- c(1,2,3,5,7,8)
q <- c("subbu","raju","raju","sankar","rajesh","ramesh")
r <- c(p,q)
r
```

Output:

```
[1] "1"    "2"    "3"    "5"    "7"    "8"    "subbu"  "raju"   "raju"   "sankar" "rajesh"
"ramesh"
```

2) Arithmetic operations

We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors. We can add, subtract, multiply, or divide two vectors. Let see an example to understand how arithmetic operations are performed on vectors.

Example:

```
a<-c(1,3,5,7)
b<-c(2,4,6,8)
print("Addition of a+b")
a+b
print("Subtraction of a-b")
a-b
print("Division of a/b")
a/b
print("Modolus of a%%b")
a%%b
```

Output:

```
[1] "Addition of a+b"
> a+b
```

```
[1]  3  7 11 15
[1] "Subtraction of a-b"
> a-b
[1] -1 -1 -1 -1
[1] "Division of a/b"
> a/b
[1] 0.5000000 0.7500000 0.8333333 0.8750000
[1] "Modolus of a%%b"
> a%%b
[1] 1 3 5 7
```

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)
add.result <- v1+v2
print(add.result)
sub.result <- v1-v2
print(sub.result)
```

Output:

```
[1]  7 19  8 16  4 22
[1] -1 -3  0 -6 -4  0
```

Vector Element Sorting

Elements in a vector can be sorted using the **sort()** function.

```
v <- c(3,8,4,5,0,11, -9, 304)

# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)

# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)

# Sorting character vectors.
v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
print(sort.result)

# Sorting character vectors in reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

Output:

```
print(sort.result)
[1]  -9   0   3   4   5   8  11 304

print(revsort.result)
[1] 304  11   8   5   4   3   0  -9

print(sort.result)
[1] "Blue"   "Red"    "violet" "yellow"

print(revsort.result)
[1] "yellow" "violet" "Red"    "Blue"
```

4(b) Implement R script for finding the sum and average of given numbers using arrays.

```
thisarray <- c(1:24)
multiarray <- array(thisarray,dim = c(4,3,2))
print(multiarray)

print("sum of array elements:")
print(sum(multiarray))

print("Length of the array:")
len <-length(multiarray)
len

print("Average of array elements:")
print(sum(multiarray)/len)
```

Output:

```
[1] "sum of array elements:"
[1] 300
[1] "Length of the array:"
[1] 24
[1] "Average of array elements:"
[1] 12.5
```

4(c) Implement R script to display elements of list in reverse order.

```
list1 <-c(1:24)
print(list1)
print(Elements in Reverse Order")
rev.default(list1)
```

Output:

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
[1] "Elements in Reverse Order"
[1] 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

4(d) Implement R script to find the minimum and maximum elements in the array.

```
nums = c(10, 20, 30, 40, 50, 60)
array1 <-array(nums)
print("The elements in the Array:")
array1
print(paste("Maximum value :",max(nums)))
print(paste("Minimum value :",min(nums)))
```

Output:

```
[1] "The elements in the Array:"
[1] 10 20 30 40 50 60
[1] "Maximum value : 60"
[1] "Minimum value : 10"
```