ASSIGNMENT - 4

2451-18-733-001
M. Ramani Priya

1. Write a program to find the factorial of a number using R.

```
factorial ← function(n) {
    if (n==0) return(1)
    else
        return (n * factorial(n-1))
}

~~factorial~~
n = 5
answer ← factorial(n)
print(answer)
```

2. Describe the data structures in R programming language.

Data structures in R;

1. atomic vector.
2. list
3. matrix
4. data frame.
5. factors.

1. Vector: It is the most common & basic data structure in R

Vectors can be of 2 types:

(i) atomic vector.
(ii) list

A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.

→ vector() - creates an empty vector of default mode - logical.

→ Common approach to creating vectors is to use character(), numeric() etc.

→ vectors can be created directly by specifying their content. R will then guess appropriate mode of storage of vector.

x ← c(1, 2, 3) ⇒ mode - numeric

→ x1 ← c(1L, 2L, 3L) - Integer mode.

→ z ← c("Sarah", "Tracy", "John") mode - character.

Examining vectors.

the functions - typeof(), length(), class() & str() provide useful information about your vectors & R objects.

Adding elements.

the function c() (for combine) can also be used to add elements to vectors.

→ Vectors can also be created as a sequence of vectors

series ← 1:10

seq(10)

→ 1 2 3 4 5 6 7 8 9 10

→ R supports missing data in vectors. They are represented by NA - not available.

The function is.na() indicates the elements of vectors that represent missing data, & the function anyNA() returns TRUE if vector contains any missing values.

→ Inf is infinity.

2451-18-733-001

2. Lists

A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous data structures.
These are also one-dimensional data structures.
A list can be a list of vectors, a list of matrices, a list of characters or a list of functions & so on.

Eg: empId = c(1, 2, 3, 4)

empName = c("Obi", "C", "B", "A")

numberOfEmp = 4.

empList = list(empId, empName, numbersOfEmp)

=> empList => [[1]]
[1] 1 2 3 4

[[2]]
[1] "D" "C" "B" "A".

[[3]]
[1] 4.

3. Data frames.

Data frames are generic data objects of R which are used to store the tabular data. Dataframes are the foremost popular data objects in R programming because we are comfortable in seeing the data within the tabular form.
They are 2 dimensional, heterogenous data structures. These are lists of vectors of equal length.

Data frames have the following constraints placed on them:

1. A data frame must have column name & every row should have a unique name.

2. Each column must have the identical number of items.
3. Each item in a single column must be of same datatype.
4. Different columns may have different data types

To create a dataframe we use data.frame() function.

Eg: a = c("b", "c", "d")

x = c("p", "q", "r")

y = c(22, 25, 28)

df = data.frame(a, x, y)

df ⇒

	a	x	y
1	b	p	22
2	c	q	25
3	d	r	28

3. Matrix

A matrix is a rectangular arrangement of numbers in rows & columns. In a matrix, the rows are ones that run horizontally & columns run vertically. Matrices are 2 dimensional, homogenous data structures.

Matrices can be created using matrix() function.

The arguments of matrix() are set of elements in the vector.

By default, matrices are in column wise order.

Eg: A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE)

A ⇒

	[,1]	[,2]	[,3]
[1,]	1	2	3
[2,]	4	5	6
[3,]	7	8	9

5. Arrays

Arrays are the R objects which store the data in more than 2 dimensions. Arrays are n dimensional data structures.

for example, if we create an array of dimensions (2, 3, 3) then it creates 3 rectangular matrices each with 2 rows & 3 columns. They are homogeneous data structures.

Arrays can be created using array() function.

The arguments to array() are the set of elements in vectors & you have to pass a vector containing dimensions of array.

Eg: a = array (
c(1, 2, 3, 4, 5, 6, 7, 8),
dim = c(2, 2, 2)
)

```
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

6. Factors.

Factors are the data objects which are used to categorize the data & store it as levels. They are useful for storing categorical data. They can store both strings & integers. They are useful in data analysis for statistical modeling.

Factors can be created using a function - factors().

The argument to factors() is a vector.

Factors are useful to categorize unique values in columns like "TRUE" or "FALSE", "MALE" or "FEMALE". etc.

Eg: fac = factor (c("Male", "Female", "Male", "Male", "Female"))

fac ⇒ [1] Male Female Male Male Female.

- Levels: Female Male.

3. Define objects. List the methods for measuring distance between objects.

Objects: Objects are the instances of the class. Also, everything in R is an object & they can have their attributes like class, attributes, dimnames, names etc.

In R programming,

→ Stats::dist() is a base method used to calcal get distance between objects.

however this method can compute only following distances, euclidean, maximum, manhattan, canberra, binary & minkowski

→ distance() function is implemented using same logic as Stats::dist() & takes as input matrix or data.frame. The corresponding matrix & data.frame should store probability density functions (as rows) for which distance computations should be performed.

→ When defining matrix the probability vectors should be combined as rows using rbind().

→ distance() function allows to choose from 46 distance/similarity measures.

→ getDistMethods() can give information about which methods are implemented in distance().

→ distance() returns a symetric matrix whereas Stats::dist() returns only a part of matrix.

The arguments of distance() are matrix, method, use.rows.names (boolean), as.dist.obj (boolean).

Eg: P ← 1:10/sum(1:10)

Q ← 20:29/sum(20:29)

x ← rbind(P,Q) # combine P & Q as matrix object.

distance(x, method = "euclidean", use.row.names = TRUE,
as.dist.obj = TRUE)

stats :: dist(x, method = "euclidean").

Methods for measuring distances:

1. Euclidean Distance

The most common distance is Euclidean distance. The euclidean distance between two vectors x & y is defined as

distance(x, y) ← sqrt((x[1]-y[1])^2 + (x[2]-y[2])^2 + ...)

$$d(x,y) = \sqrt{\sum_{i=1}^{2}(x_i - y_i)^2}$$

2. Manhattan distance (city block)

Manhattan distance measure distance in the number of horizontal & vertical units it takes to get from one (real valued) point to another (no diagonal moves).

distance(x,y) ← sum(abs(x[1]-y[1]) + abs(x[2]-y[2]) + ...)

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

3. Cosine similarity.

is a common similarity metric in text analysis.

It measures the smallest angle between 2 vectors & is assumed to be between 0 & 90°.

Two perpendicular vectors ($\theta = 90°$) are the most dissimilar,

the cosine of 90° is 0.

Two parallel vectors are similar, cos(0) = 1

dot(x,y) ← sum(x[1] * y[1] + x[2] * y[2] + ...)

Cossim(x,y) ← dot(x,y) / (sqrt(dot(x,x) * dot(y,y)))

$$\cos\theta = \frac{A \cdot B}{|A||B|}$$

4. List out the various control structures supported by R programming language.

There are 8 control statements supported by R programming language.,

Control statements are expressions used to control the execution & flow of program based on the conditions provided in statements. The statements are used to make a decision after assessing the variable.

1. if condition

This control statement structure contains checks the expression provided in parenthesis is true or not. If true, the execution of statements in braces {} continues.

Syntax:
```
if (expression) {
        Statements
        ....
        ....
}
```

2. if - else condition

It is similar to if condition but when the test expression in if condition fails, then statements in else condition are executed.

Syntax: if (expression) {

Statements

....

....

}

else {

statements

....

....

}

3. for loop.

It is a type of loop or sequence of statements executed repeatedly until exit condition is ~~ex~~ reached.

Syntax: for (value in vector) {

Statements

....

....

}

4. Nested loops

Nested loops are similar to simple loops. Nested means loops inside loop. Moreover, nested loops are used to manipulate the matrix.

Eg:
```
m ← matrix (2:15, 2)
for (r in seq(nrow(m))) {
    for ( c in seq (ncol (m))) {
        print (m[r, c])
    }
}
```

5. While loop

While loop is another kind of loop iterated until a condition is satisfied. The testing expression is checked first before executing the body of loop.

Syntax:
```
while (expression){
    Statement
    ...
    ...
}
```

6. repeat loop & break statement

Repeat is a loop which can be iterated many number of times but there is no exit condition to come out from the loop. So, break statement is used to exit from loop. break statement can be used in any type of loop to exit from loop.

Syntax:
```
repeat {
        statement
        ....
        ....
        if (expression) {
            break
        }
}
```

7. return statement.

return statement is used to return the result of can executed function & returns control to the calling function.

Syntax: return (expression)

8. next statement.

next statement is used to skip the current iteration without executing further statements & continues the next iteration cycle without terminating the loop.

Eg:
```
x ← 1:10
for (i in x) {
    if (i%%2 != 0){
        next # jumps to next loop
    }
    print(i)
}
```

5) Same as 2nd.

6. Define List & Data frame in R & explain various operations on lists & dataframes with suitable examples.

Lists:

Lists are R objects which contains elements of different types like - numbers, strings, vectors & another list inside it A List can contain a matrix or a function as its elements.

→ List is created using list() function.

→ The list elements can be given names & can be accessed using these names using names() function.

→ The elements of list can be accessed by index of element in list. In case of named list it can be also be accessed using names.

→ We can add, delete, update list elements.
We can add & delete only at beginning or end of list
But we can update any element of list.

→ We can merge many lists into one list by placing all the lists in on list() function.

→ A list can be converted to vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vector vectors. To do this conversion, we use unlist() function.

Eg: list_data ← list (c("a", "b", "c"), matrix (c (3, 9, 5, -1, 2, 8), nrow = 2), list ("g", 2, 3))

names (list_data) ← c ("1st quarter", "Matrix", "Inner List")

```
> print (list_data [1]
  1st Quarter
  "a"  "b"  "c"

> list_data[4] ← "New element"

> # Merging Lists
  list1 ← list(1, 2, 3)
  list2 ← list ("A", "B", "C")
  merged.list ← c(list1, list2)

> # Converting list to vector.
  list1 ← list(1:5)
  list2 ← list(10:14)
  V1 ← unlist(list1)
  V2 ← unlist(list2)
  result ← V1 + V2
```

Data frames

Data frames are generic data objects of R which are used to store the tabular data. Data frames are considered to be the most popular data objects in R objects because it is more comfortable to analyze the data in the tabular forms.

Operations that can be performed using Dataframes are,

1. Creating Dataframe

(i) Creating data frame from vectors:

We can use data.frame() function.

Eg:
```
name = c("a", "b", "c")
language = c("R", "P", "J")
age = c(22, 25, 28)
```

df = data.frame (name, language, age)

(ii) Creating dataframe using data from a file:

Dataframes can also be created by importing the data from a file..

newDF = read.table (path = "Path of file")

newDF = read.csv (path = "Path of file")

2. Accessing rows and columns.

df [val1, val2]

df = dataframe object

val1 = rows of a data frame

val2 = columns of dataframe

accessing rows.,

df [1:2 ,]

accessing columns

df [, 1:2]

3. Selecting subset of Dataframe.

A to subset can also be created based on certain conditions

newDF = subset (df , col conditions)

df = Original dataframe

conditions = certain conditions.

4. Editing Dataframes.

Dataframes can be edited in 2 ways,

(i) Editing dataframes by direct assignments: Much like the list in R you can edit the data frames by a direct assignment.

(ii) Editing data frames using edit() command.

Step1 : Create an empty instance of data frame.

Step2 : use edit function to launch viewer.

myTable = data.frame()

myTable = edit(myTable)

Step 3: enter the values in table & close the editor

Step 4: Check resulting table.

> myTable

5. Adding rows & columns to data frame.

(i) Adding rows:

function: rbind (df, the entries of row)

newDF = rbind (df, data.frame(name = "d",
language = "C",
age = 40))

(ii) Adding extra columns.

function: cbind (df, the entries of column)

newDF = cbind (df, Rank = c(3, 5, 4, 1))

6. Adding new variables to data frames.

We can add variables to data frame based on existing ones. To do that we have to first called the dplyr library using command library(). Then call mutate() function will add extra variable columns based on existing ones

library (dplyr)

newDF = mutate (df, new_var = [existing_var])

df = original dataframe.

new_var = Name of new variable.

existing_var = The modify action you are taking (e.g log value, multiply by 10).

7. Deleting rows & columns from a data frame. 2451-18-733-001

To delete a row or column you need to access that row or column & then insert a negative sign before that row or column. It indicates that you need to delete that row or column.

newDF = df [-rowNo, -colNo]