**Functions, Array, Vector, List and Matrix in R**

**Day 2**

**R functions**

func\_name<- function(arg\_1, arg\_2, ...)

{

function body

}

Similar to the other languages, R also has two types of function, i.e. Built-in Function and User-defined Function. In R, there are lots of built-in functions which we can directly call in the program without defining them. R also allows us to create our own functions.

**# A simple R function to calculate area of a circle**

areaOfCircle = function(radius)

{

area = pi\*radius^2

return(area)

}

print(areaOfCircle(2))

**Examples of built-in functions**

**# Find sum of numbers 4 to 6.**

print(sum(4:6))

**# Find max of numbers 4 and 6.**

print(max(4:6))

**# Find min of numbers 4 and 6.**

print(min(4:6))

**Dots arguments**

Dots argument (…) is also known as ellipsis which allows the function to take an undefined number of arguments. It allows the function to take an arbitrary number of arguments. Below is an example of a function with an arbitrary number of arguments.

**Example:**

**# Function definition of dots operator**

fun<- function(n, ...)

{

l <- list(n, ...)

paste(l, collapse = " ")

}

**# Function call**

fun(5, 1L, 6i, TRUE, "GeeksForGeeks", "Dots operator")

**Vector**

A vector is a basic data structure which plays an important role in R programming. 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.

1. Vectors may be created using the c command, separating your elements with commas.

> a <- c(1, 7, 32, 16)

>a

[1] 1 7 32 16

**Sequences of integers may be created using a colon (:).**

> b <- 1:10

>b

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

> c <- 20:15

>c

[1] 20 19 18 17 16 15

Other regular vectors may be created using the seq (sequence) and rep (repeat) commands.

> d <- seq(1, 5, by=0.5)

>d

[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

> e <- seq(0, 10, length=5)

>e

[1] 0.0 2.5 5.0 7.5 10.0

> f <- rep(0, 5)

>f

[1] 0 0 0 0 0

> g <- rep(1:3, 4)

>g

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

> h <- rep(4:6, 1:3)

>h

[1] 4 5 5 6 6 6

**Sorting elements of a Vector**

sort() function is used with the help of which we can sort the values in ascending or descending order.

# R program to sort elements of a Vector

# Creation of Vector

X <- c(8, 2, 7, 1, 11, 2)

# Sort in ascending order

A <- sort(X)

cat('ascending order', A, '\n')

B <- sort(X, decreasing = TRUE)

cat('descending order', B)

**Creating a List**

To create a List in R you need to use the function called “list()”.

In other words, a list is a generic vector containing other objects. To illustrate how a list looks, we take an example here. We want to build a list of employees with the details. So for this, we want attributes such as ID, employee name, and the number of employees.

**Example**

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

empName = c("Debi", "Sandeep", "Subham", "Shiba")

numberOfEmp = 4

empList = list(empId, empName, numberOfEmp)

print(empList)

**R program to access components of a list**

# Creating a list by naming all its components

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

empName = c("Debi", "Sandeep", "Subham", "Shiba")

numberOfEmp = 4

empList = list("ID" = empId, "Names" = empName, "Total Staff" = numberOfEmp)

print(empList)

# Accessing components by names

cat("Accessing name components using $ command\n")

print(empList$Names)

**Creating an Array**

array(data, dim = (nrow, ncol, nmat), dimnames=names)

nrow : Number of rows, ncol : Number of columns

nmat : Number of matrices of dimensions nrow \* ncol, dimnames : Default value = NULL.

**What is difference between array and list in R?**

An array in R can have one, two or more dimensions.

It is simply a vector which is stored with additional attributes giving the dimensions (attribute “dim”) and optionally names for those dimensions (attribute “dimnames”).

array(list()) is a list with an additional dim attribute.

The list is one-dimensional in which each data element is the data structure itself.

A list can contain data elements of the same and different data types.

An array is a data structure that can beone-dimensional or multidimensional. It stores the data

elements of similar data types.

**Multi-Dimensional Array**

A two-dimensional matrix is an array specified by a fixed number of rows and columns, each containing the same data type. A matrix is created byusingarray() function to which the values and the dimensions are passed.

**Example:**

# arranges data from 2 to 13

# in two matrices of dimensions 2x3

arr = array(2:13, dim = c(2, 3, 2))

print(arr)

**Output:**

, , 1

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

[1,] 2 4 6

[2,] 3 5 7

, , 2

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

[1,] 8 10 12

[2,] 9 11 13

**Naming of Arrays**

row\_names<- c("row1", "row2")

col\_names<- c("col1", "col2", "col3")

mat\_names<- c("Mat1", "Mat2")

The naming of the various elements is specified in a list and fed to the function

arr= array(2:14, dim = c(2, 3, 2), dimnames = list(row\_names, col\_names, mat\_names))

print (arr)

**Output:**

,, Mat1

col1 col2 col3

row1 24 6

row2 35 7

,, Mat2

col1 col2 col3

row1 810 12

row2 911 13

R packages - R packages contain code, data, and documentation in a standardised collection format that can be installed by users of R, typically via a centralised software repository such as CRAN (the Comprehensive R Archive Network). One needs to install a package in the computer once and load it in every session by using the library(“package”). library() is the command used to load a package. It is similar to install a light, and then switch it on every time you use it.

“swirl” is a very useful package for interactive learning R language.

install.packages(“swirl”)

library(“swirl’) # Quote is optional

swirl()

Select 1 to learn R programming.

Please choose a course, or type 0 to exit swirl.

1: R Programming

2: Take me to the swirl course repository!

Choose 1 to learn basic building blocks.

Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks 2: Workspace and Files

3: Sequences of Numbers 4: Vectors

5: Missing Values 6: Subsetting Vectors

7: Matrices and Data Frames 8: Logic

9: Functions 10: lapply and sapply

11: vapply and tapply 12: Looking at Data

13: Simulation 14: Dates and Times

15: Base Graphics

In this practice session, follow 1 and 2 one after the other to learn upto workspace and Files. It is interesting. The remaining lessons you will gradually learn in other practice sessions and assignments.