Ex.No.1 **BASICS OF R**

Date: - 15-07-23

Aim

To implement the Basics of R programming in the experiments and learn about them.

Procedure

1. To do programming in R, first install “RStudio” and “R” in the system. RStudio is an integrated development environment [IDE] for R and python.
2. Select the File in taskbar →open New file →R script or use shortcut “ctrl+shift+N”
3. Write the program in the script and save it using the extension R.
4. Run the program by clicking Run option or use the shortcut “ctrl+enter”.
5. See the output in the console tab.

Concepts Applied

- Simple program

Script

"Hello World!I am Navadeep"

Output

```
[1] "Hello World!I am Navadeep"
```

ARITHMETIC OPERATIONS

The basic Arithmetic operations +,-,*,/ are used on Operands and values are printed.

Script

#Basic Arithmetic Operations
#Addition
2+2
#subtraction
2-2
#Product
2*2
#Division
2/2

Output

```
[1] 4
[1] 0
[1] 4
[1] 1
```

VARIABLE ASSIGNMENTS

Assigning the values to variables and printing them. Multiple variable assignments can also be done. <- Assigning a value to a variable

Script

```
#variable Assignment

name<- "navadeep"

age<-19

name

age

#Multiple variables

a<-b<-c<-5

a

b

c
```

Output

```
[1] "navadeep"
[1] 19
[1] 5
[1] 5
[1] 5
```

CONCATENATION USING PASTE ()

Concatenate the value and variable using paste() method.

Script

```
#concatenation using paste()

paste("Name is",name)
```

Output

```
[1] "Name is navadeep"
```

DATA TYPES

Variables can store

- Different datatypes like integer, numeric, complex, character etc.,
- Using class() to check the type of data

Script

```
#Data types

x <- 10L

class(x)

y <- 9.5

class(y)

z <- 10i + 5

class(z)

a <- "NavadeepDJ"

class(a)

b <- TRUE

class(b)
```

Output

```
[1] "integer"
[1] "numeric"
[1] "complex"
[1] "character"
[1] "logical"
```

TYPE CONVERSION

To convert one datatype to another datatype using the keywords like as.numeric(), as.integer() etc., for their respective datatypes

- as.integer() : to convert a datatype into integer type
- as.numeric() : to convert a datatype into numeric type

Script

```
#Type conversion

x <- 10L
```

```
y <- 9.5

a <- as.numeric(x)

b <- as.integer(y)

#print the values a and b

a

b

class(a)

class(b)
```

Output

```
[1] 10
[1] 9
[1] "numeric"
[1] "integer"
```

BUILT IN MATH FUNCTIONS

In R, there are some built in math functions which we can use without defining them. Like, max (), min (), sqrt (), abs () etc.,

Script

```
#Built-in-math functions

max(2,5,7)

min(2,5,7)

sqrt(9)

abs(-9.5)
```

Output

```
[1] 7
[1] 2
[1] 3
[1] 9.5
```

ROUNDING TO THE NEAREST

There are some built in functions which can be used for Rounding to the nearest like :

- ceiling() :it rounds off the values to their nearest greater number.
- floor() : it rounds off the values to their nearest lesser number.

Script

```
ceiling(2.5)

floor(2.5)

ceiling(3.4)

floor(3.4)
```

output

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

MULTILINE STRING

Multiline string is same as string but it has multi lines. The cat() method can break the lines too.

Script

```
#Multiline string

str <- "I am Navadeep

of speciliazation AIML

in Computer science and

engineering"


str#print the value of string


#Line breaks in multiline string using cat()

cat(str)


#string length using nchar()

nchar(str)
```

Output

```
[1] "I am Navadeep\nof speciliazation AIML\nin Computer science
and\nengineering"
I am Navadeep
of speciliazation AIML
in Computer science and
engineering
[1] 72
```

CHECK A STRING

For checking a string:

- Using grepl(): by using the grepl() method, we can find out whether the character or element is in the string or not.
- It returns the Boolean values i.e., True or False

Script

```
#check a string using grepl()

str1 <- "Navadeep"

grepl("N",str1)

grepl("A",str1)

grepl("D",str1)
```

Output

```
[1] TRUE
[1] FALSE
[1] FALSE
```

ESCAPE CHARACTERS

These escape characters are used when we are inserting a special character in another datatype and we don't want an error when printing it.

- **\ :** Backslash is used as a escape character when it is used to escape a special character.
- Using cat() along with the backslash to get the line breakers exactly as we want.

Script

```
#escape characters

str2 <- "I am the \"King\" of the kings"

str2
```

cat(str2)# cat() gives the Exact statement we prompted

output

```
[1] "I am the \"King\" of the kings"
I am the "King" of the king
```

BOOLEAN VALUES

The True and False are Boolean values.

Script

```
10 > 9

10 == 9

10 < 9
```

Output

```
[1] TRUE
[1] FALSE
[1] FALSE
```

CONDITIONAL STATEMENTS

The conditional statements consist of if else, else if etc.,

Script

```
# if else

a <- 10

b <- 20

if (b > a){

  print("b is greater than a")

 } else{

    print("a is greater than b")

  }
```

Output

```
[1] "b is greater than a"
```

ASSIGNMENT OPERATORS

Variable assignment with values using equality.Comparison of the different assignment operators: <- and<<-

Script

```
#Assignment operators(<- or <<-)

a <- 10

a <<- 10

a

(a <- 10) == (a <<- 10)
```

Output

```
[1] 10
[1] TRUE
```

LOGICAL OPERATORS

There are majorly two logical operators which are used for comparing two or more expressions. They are:

- & : logical AND operator
- | : logical OR operator

Script

```
#logical operators

#logical AND

a <- 100

b <- 50

c <- 20

if (a>b & a>c){

  print("Both conditions are true")

}else{

  print("Both are not true")

}

# logical Or
```

```
a <- 100

b <- 50

c <- 20

if (a>b | c>a){

  print("one of the conditions is true")

}
```

Output

```
[1] "Both conditions are true"
[1] "one of the conditions is true"
```

LOOPS

Loops are useful when one has to repeat a same thing over and over again since they are more suited for them. Here, the loops are:

- While loop
- For loop

Script

```
#while loop

i <- 1

while (i < 6){

  print(i)

  i <- i + 1

}

#for loop

for (x in 1:7){

  print(x)

}
```

Output

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

[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
```

CREATING AND CALLING FUNCTIONS

Functions can be created by using Keyword function () and they are called by using the function's name

Script

```
#creating a function
my_function<-function(){
  print("I am Navadeep")
}
my_function()#calling a function
```

Output

```
[1] "I am Navadeep"
```

ARGUMENTS

Information can be passed into functions as arguments.

Script

```
#Arguments
my_function<-function(fname,lname){
  paste(fname,lname,"is good")
}
my_function("M","Navadeep")
```

Output

```
"M Navadeep is good"
```

NUMBER OF ARGUMENTS

By default, a function must be called with the correct number of arguments.The number of arguments in the function call must be same as the number of arguments the function expects.

Script

```
#Number of Arguments
my_function<-function(fname,lname){
  paste(fname,lname,"is good")
}
my_function("M","Navadeep","N")
```

Output

```
Error in my_function("M", "Navadeep", "N") : unused argument ("N
")
```

DEFAULT PARAMETER VALUE

Default parameter values are printed when the argument is not present in the function call.

Script

```
my_function<-function(fname = "NK"){
  paste(fname,"is good")
}
my_function()
```

Output

```
[1] "NK is good"
```

RETURN VALUES

To return the values from the function.

Script

```
my_function<-function(x){
  print(2+x)
```

```
}
my_function(1)
my_function(2)
```

Output

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

NESTED FUNCTIONS

A function within a function is basically called as Nested functions.

Script

```
#Nested functions
nested_function<-function(a,b){
  x<-a*b
  return(x)
}
nested_function(nested_function(2,3),nested_function(2,3))
```

Output

```
[1] 36
```

RECURSION

R also accepts recursion which means a defined function call itself.

Script

```
#recursion
tri_recursion <- function(k) {
  if (k > 0) {
    result <- k + tri_recursion(k - 1)
    print(result)
  } else {
    result = 0
    return(result)
```

```
  }
}
tri_recursion(7)
```

Output

```
[1] 1
[1] 3
[1] 6
[1] 10
```

GLOBAL VARIABLES

The variables that are defined outside of the function are called global variables. They can be used anywhere in the program.

Script

```
#global variables
f<- "Nk"
my_function<-function(){
  print(f)
}
my_function()
```

Output

```
[1] "Nk"
```

Result: -
Thus the basics of R programming experiments have been executed successfully.