# **DT:8-2-23 LAB EXERCISES**

# ITA0443-STATISTICS WITH R-PROGRAMMING

Name:D.Tej kiran

Reg.no:192110625

**1.BASIC OPERATIONS IN R**

**Q1)Write The Commands To Perform Basic Arithmetic In R.**

**# addition**

x <- 2 + 3

print(x)

**# subtraction**

y <- 5 - 2

print(y)

**# multiplication**

z <- 2 \* 3

print(z)

**# division**

a <- 8 / 4

print(a)

**# exponentiation**

b <- 2 ^ 3

print(b)

**Q2) Display a String on R Console.**

> cat("Hello World")

Hello World

Alternatively, you can use the **print** function:

> print("Hello World")

[1] "Hello World"

**Q3) Declare Variables In R And Also Write The Commands For Retrieving The Value Of The Stored Variables In R Console.**

**INPUT:**

x <- 5

y <- 10

z <- "Hello World"

**OUTPUT:**

> x

[1] 5

> y

[1] 10

> z

[1] "Hello World"

**Q4. Write R script to calculate the area of Rectangle.**

**INPUT:**

rectangle\_area <- function(l, w) {

area <- length \* width

return(area)

}

l<- 5

w<- 10

rectangle <- rectangle\_area(length, width)

cat("The area of the rectangle is", rectangle)

**OUTPUT:**

The area of the rectangle is 50

**Q5.Write Commands In R Console To Determine The Type Of Variable**

To determine the type of a variable in R, you can use the class function. For example, if you have a variable x:

**INPUT:**

x <- 10

class(x)

**OUTPUT:**

[1] "numeric"

**Q6.Enumerate The Process To Check Whether A Given Input Is Numeric , Integer ,Double, Complex in R.**

1.Check if it's numeric:

INPUT:

x <- 10

is.numeric(x)

OUTPUT:

TRUE

2.Check if it's an integer:

INPUT:

x <- 10

is.integer(x)

OUTPUT:

TRUE

3.Check if it's a double:

INPUT:

x <- 10

is.double(x)

OUTPUT:

TRUE

4.Check if it's complex:

INPUT:

x <- 10 + 4i

is.complex(x)

OUTPUT:

TRUE

**Q7)Illustration of vector arithmetic:**

INPUT:

vector1 <- c(1, 2, 3)

vector2 <- c(4, 5, 6)

result <- vector1 + vector2

result

OUTPUT:

[1] 5 7 9

**Q8)Write an R Program to Take Input From User.Input name as “Jack” and age as 17.The program should display the output as“Hai , Jack next year you will be 18 years old”**

INPUT:

x<-readline(prompt = "name:")

y<- readline(prompt = "age")

paste("Hai",x,"next year you will be ",y+1,"years old")

OUTPUT:

Hai jack next year you will be 18 years old.

----------------------------------------------------------------------------------------------------------------------

**2.DATA STRUCTURES IN R**

**1) Perform Matrix Addition &amp; Subtraction in R**

A <- matrix(1:4, nrow = 2, ncol = 2)

B <- matrix(5:8, nrow = 2, ncol = 2)

C <- A + B

print(C)

D <- A - B

print(D)

**OUTPUT:**

[,1] [,2]

[1,] 6 8

[2,] 10 12

[,1] [,2]

[1,] -4 -4

[2,] -4 -4

**2) Perform Scalar multiplication and matrix multiplication in R**

INPUT:

A <- matrix(1:4, nrow = 2, ncol = 2)

B <- 2 \* A

print(B)

C <- A %\*% t(A)

print(C)

OUTPUT:

[,1] [,2]

[1,] 2 4

[2,] 6 8

[,1] [,2]

[1,] 10 14

[2,] 14 20

**Q3) Find Transpose of matrix in R**.

A <- matrix(1:4, nrow = 2, ncol = 2)

B <- t(A)

print(B)

**OUTPUT:**

**[,1] [,2]**

**[1,] 1 3**

**[2,] 2 4**

**Q4) Perform the operation of combining matrices in R using cbind() and rbind() functions.**

A <- matrix(1:4, nrow = 2, ncol = 2)

B <- matrix(5:8, nrow = 2, ncol = 2)

C <- cbind(A, B)

print(C)

D <- rbind(A, B)

print(D)

**OUTPUT:**

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

**[1,] 1 2 5 6**

**[2,] 3 4 7 8**

**[,1] [,2]**

**[1,] 1 2**

**[2,] 3 4**

**[3,] 5 6**

**[4,] 7 8**

**5) Deconstruct a matrix in R**

A <- matrix(1:4, nrow = 2, ncol = 2)

a1 <- A[1,1]

a2 <- A[1,2]

a3 <- A[2,1]

a4 <- A[2,2]

print(a1)

print(a2)

print(a3)

print(a4)

**OUTPUT:**

**[1] 1**

**[1] 2**

**[1] 3**

**[1] 4**

**6) Perform array manipulation in R**

x <- c(1, 2, 3, 4)

y <- matrix(rep(x, times = 2), ncol = 2, byrow = TRUE)

z <- array(1:24, dim = c(2, 3, 4))

print(x)

print(y)

print(z)

**OUTPUT:**

[1] 1 2 3 4

[,1] [,2]

[1,] 1 1

[2,] 2 2

[3,] 3 3

[4,] 4 4

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

[1,] 1 3 5

[2,] 2 4 6

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

[1,] 7 9 11

**7) Perform calculations across array elements in an array using the apply() function.**

x <- matrix(1:6, nrow = 2, ncol = 3)

col\_sums <- apply(x, 2, sum)

print(col\_sums)

row\_means <- apply(x, 1, mean)

print(row\_means)

**OUTPUT:**

**[1] 3 5 7**

**[1] 2.5 3.5**

**8) Demonstrate Factor data structure in R.**

x <- c("apple", "banana", "cherry", "banana", "apple")

x\_factor <- factor(x)

print(x\_factor)

**OUTPUT:**

[1] apple banana cherry banana apple

Levels: apple banana cherry

**9) Create a data frame and print the structure of the data frame in R.**

df <- data.frame(Name = c("SHASHI", "TAKESH", "SAI"),

Age = c(19, 20, 21),

Gender = c("Male", "male", "Male"))

str(df)

**OUTPUT:**

$ Name : Factor w/ 3 levels "SHASHI","TAKESH","SAI": 3 1 2

$ Age : num 19 20 21

$ Gender: Factor w/ 2 levels "male","Male": 2 1 2

**10) Demonstrate the creation of S3 class in R.**

# Define a class

Person <- function(name, age) {

structure(list(name = name, age = age), class = "Person")

}

# Define a method for the class

print.Person <- function(person) {

cat(paste("Name:", person$name, "\nAge:", person$age, "\n"))

}

# Create an object of the class

p1 <- Person("John", 30)

# Call the method for the object

print(p1)

**OUTPUT:**

**Name: John**

**Age: 30**

**11) Demonstrate the creation of S4 class in R.**

setClass("Person", representation(name = "character", age = "numeric"))

setMethod("print", "Person", function(object) {

cat(paste("Name:", object@name, "\nAge:", object@age, "\n"))

})

p1 <- new("Person", name = "John", age = 30)

print(p1)

**OUTPUT:**

**Name: John**

**Age: 30**

**12) Demonstrate the creation of Reference class in R by defining a class called students with fields – Name, Age , GPA. Also illustrate how the fields of the object can be accessed using the $ operator. Modify the Name field by reassigning the name to Paul.**

library(methods)

students <- setRefClass("students",

fields = list(

Name = "character",

Age = "numeric",

GPA = "numeric"

)

)

s1 <- students$new(Name = "John", Age = 25, GPA = 3.5)

cat("Name:", s1$Name, "\nAge:", s1$Age, "\nGPA:", s1$GPA, "\n")

s1$Name <- "Paul"

cat("Name:", s1$Name, "\nAge:", s1$Age, "\nGPA:", s1$GPA, "\n")

**OUTPUT:**

**Name: John**

**Age: 25**

**GPA: 3.5**

**Name: Paul**

**Age: 25**

**GPA: 3.5**

**3.WORKING WITH LOOPING AND FUNCTIONS IN R**

**1.Write a program to check whether an integer (entered by the user) is a prime number or not using control statements.**

INPUT:

num = as.integer(readline(prompt="Enter a number: "))

flag = 0

# prime numbers are greater than 1

if(num > 1) {

# check for factors

flag = 1

for(i in 2:(num-1)) {

if ((num %% i) == 0) {

flag = 0

break

}

}

}

if(num == 2) flag = 1

if(flag == 1) {

print(paste(num,"is a prime number"))

} else {

print(paste(num,"is not a prime number"))

}

OUTPUT:

Enter a number: 25

[1] "25 is not a prime number"

**2.Write a program to check whether a number entered by the user is positive number or a negative number or zero.**

INPUT:

num <- as.integer(readline(prompt="Enter a number: "))

if(num > 0) {

cat("The entered number is a positive number.")

} else if(num < 0) {

cat("The entered number is a negative number.")

} else {

cat("The entered number is zero.")

}

**OUTPUT:**

**Enter a number:9**

**[1] The entered number is a positive number**

**3.Write a program to check whether a number is an Armstrong number or not using a while loop.**

num <- as.integer(readline(prompt="Enter a number: "))

digits <- nchar(as.character(num))

sum\_cubes <- 0

temp\_num <- num

while(temp\_num > 0) {

digit <- temp\_num %% 10

sum\_cubes <- sum\_cubes + (digit^digits)

temp\_num <- floor(temp\_num / 10)

}

if(sum\_cubes == num) {

cat("The entered number is an Armstrong number.")

} else {

cat("The entered number is not an Armstrong number.")

}

**OUTPUT:**

**Enter a number:153**

**[1] entered number is an Armstrong number**

**4.Write a program to demonstrate Repeat Loop in R**

count <- 1

repeat{

print(count)

count <- count + 1

if (count > 5) {

break

}

}

**OUTPUT:**

**[1] 1**

**[1] 2**

**[1] 3**

**[1] 4**

**[1] 5**

**5.Using functions develop a simple calculator in R.**

calculate <- function(num1, num2, operator) {

if (operator == "+") {

return(num1 + num2)

} else if (operator == "-") {

return(num1 - num2)

} else if (operator == "\*") {

return(num1 \* num2)

} else if (operator == "/") {

return(num1 / num2)

} else {

return("Invalid operator")

}

}

result <- calculate(5, 3, "+")

print(result)

result <- calculate(5, 3, "-")

print(result)

result <- calculate(5, 3, "\*")

print(result)

result <- calculate(5, 3, "/")

print(result)

result <- calculate(5, 3, "^")

print(result)

**OUTPUT:**

**[1] 8**

**[1] 2**

**[1] 15**

**[1] 1.666667**

**[1] "Invalid operator"**

**6. Demonstrate the creation of a complex number in R.**

z1 <- complex(real = 1, imaginary = 2)

print(z1)

z2 <- 3 + 4i

print(z2)

**OUTPUT:**

**[1] 1+2i**

**[1] 3+4i**

**7.Write a program to multiply two numbers using a function with a default value.Assume default value as NULL.**

multiply <- function(x, y = NULL) {

if (is.null(y)) {

y <- 1

}

return (x \* y)

}

result <- multiply(5)

print(result)

result <- multiply(5, 3)

print(result)

**OUTPUT:**

**[1] 5**

**[1] 15**

**8.Find sum, mean and product of vector elements using built-in functions.**

vec <- c(1, 2, 3, 4, 5)

sum\_of\_elements <- sum(vec)

print(sum\_of\_elements)

mean\_of\_elements <- mean(vec)

print(mean\_of\_elements)

product\_of\_elements <- prod(vec)

print(product\_of\_elements)

**OUTPUT:**

**[1] 15**

**[1] 3**

**[1] 120**

**9.Sort a vector in R using sort() function. Also find the index of the sorted vector.**

vec <- c(5, 3, 2, 4, 1)

sorted\_vec <- sort(vec)

print(sorted\_vec)

index\_sorted\_vec <- order(vec)

print(index\_sorted\_vec)

**OUTPUT:**

**[1] 1 2 3 4 5**

**[1] 5 4 3 2 1**

**10.Find the L.C.M of two numbers entered by the user by creating a user-defined function.**

find\_lcm <- function(x, y) {

return (x \* y / gcd(x, y))

}

x <- as.integer(readline(prompt = "Enter the first number: "))

y <- as.integer(readline(prompt = "Enter the second number: "))

lcm <- find\_lcm(x, y)

print(paste("The LCM of", x, "and", y, "is", lcm))

**OUTPUT:**

**[1]Enter the first number: 12**

**[1]Enter the second number:4**

**[1]The LCM of 12 and 4 is 12**

**4.IMPLEMENTATION OF VECTOR RECYCLING,APPLY FAMILY &RECURSION**

**Q1. Demonstrate Vector Recycling in R.**

INPUT:

vec1 <- c(1, 2, 3)

vec2 <- c(4, 5)

sum\_of\_vectors <- vec1 + vec2

print(sum\_of\_vectors)

OUTPUT:

[1] 5 7 7

**Q2. Demonstrate the usage of apply function in R**

INPUT:

mat <- matrix(1:6, ncol = 2)

row\_sums <- apply(mat, 1, sum)

print(row\_sums)

OUTPUT:

[1] 3 7 11

**Q3. Demonstrate the usage of lapply function in R**

INPUT:

list\_example <- list(c(1, 2, 3), c(4, 5, 6), c(7, 8, 9))

sum\_of\_squares <- function(x) {

sum(x^2)

}

result <- lapply(list\_example, sum\_of\_squares)

result

OUTPUT:

[[1]]

[1] 14

[[2]]

[1] 77

[[3]]

[1] 194

**Q4. Demonstrate the usage of sapply function in R**

INPUT:

list\_example <- list(c(1, 2, 3), c(4, 5, 6), c(7, 8, 9))

sum\_of\_squares <- function(x) {

sum(x^2)

}

result <- sapply(list\_example, sum\_of\_squares)

result

OUTPUT:

[1] 14 77 194

**Q5. Demonstrate the usage of tapply function in R**

INPUT:

values <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)

grouping <- c("A", "B", "A", "B", "A", "B", "A", "B", "A")

result <- tapply(values, grouping, mean)

result

OUTPUT:

A B

5.0 6.0

**Q6. Demonstrate the usage of mapply function in R**

INPUT:

a <- c(1, 2, 3)

b <- c(4, 5, 6)

multiply\_values <- function(x, y) {

x \* y

}

result <- mapply(multiply\_values, a, b)

result

OUTPUT:

[1] 4 10 18

**Q7)Sum of Natural Numbers using Recursion**

INPUT:

sum\_of\_numbers <- function(n) {

if (n == 1) {

return(1)

} else {

return(n + sum\_of\_numbers(n - 1))

}

}

result <- sum\_of\_numbers(10)

result

OUTPUT:

[1] 55

**Q8. Write a program to generate Fibonacci sequence using Recursion in R**

INPUT:

fibonacci <- function(n) {

if (n == 1 || n == 2) {

return(1)

} else {

return(fibonacci(n - 1) + fibonacci(n - 2))

}

}

result <- sapply(1:10, fibonacci)

result

OUTPUT:

[1] 1 1 2 3 5 8 13 21 34 55

**Q9. Write a program to find factorial of a number in R using recursion.**

INPUT:

factorial <- function(n) {

if (n == 0) {

return(1)

} else {

return(n \* factorial(n-1))

}

}

factorial(5)

OUTPUT:

[1] 120