Lab Manual Day 1 X.M.Mary Sushmija
192124100

1.Write The Commands To Perform Basic Arithmetic In R.

Ans

In R, you can perform basic arithmetic using the following commands:

1. Addition (+)

To add two numbers in R, use the "+" operator. For example:

```r
# Adding two numbers
2 + 3
# Output: [1] 5

# Adding a sequence of numbers
1:5 + 3
# Output: [1] 4 5 6 7 8
```

2. Subtraction (-)

To subtract two numbers in R, use the "-" operator. For example:

```r
# Subtracting two numbers
10 - 3
# Output: [1] 7

# Subtracting a sequence of numbers
1:5 - 3
# Output: [1] -2 -1 0 1 2
```

~~~
```
~~~

3. Multiplication (*)

To multiply two numbers in R, use the "*" operator. For example:

~~~
```r
# Multiplying two numbers
2 * 3
# Output: [1] 6

# Multiplying a sequence of numbers
1:5 * 3
# Output: [1]  3  6  9 12 15
```
~~~

4. Division (/)

To divide two numbers in R, use the "/" operator. For example:

~~~
```r
# Dividing two numbers
10 / 5
# Output: [1] 2

# Dividing a sequence of numbers
1:5 / 2
# Output: [1] 0.5 1.0 1.5 2.0 2.5
```
~~~

5. Exponentiation (^)

To raise a number to a power in R, use the "^" operator. For example:

```r
# Raising a number to a power
2^3
# Output: [1] 8

# Raising a sequence of numbers to a power
1:5^2
# Output: [1]  1  4  9 16 25
```

6. Modulus (%%)

To find the remainder of a division in R, use the "%%" operator. For example:

```r
# Finding the remainder of a division
10 %% 3
# Output: [1] 1

# Finding the remainder of a sequence of divisions
1:5 %% 2
# Output: [1] 1 0 1 0 1
```

2. Display a String on R Console.

Ans

To display a string on the R console, you can use the `print()` function or simply type the string in quotes directly on the console. Here's an example:

```
print("Hello, World!")


# Output: [1] "Hello, World!"

"Hello, World!"

# Output: [1] "Hello, World!"
```

In both cases, the string "Hello, World!" will be displayed on the R console. Note that the `print()` function is useful when you want to display the value of a variable or the result of an expression.

3. Declare Variables In R And Also Write The Commands For Retrieving The Value Of

The Stored Variables In R Console.

Ans

To declare variables in R, you can use the assignment operator (`<-` or `=`). Here's an example:

```
x <- 10

y <- "hello"

x

# Output: [1] 10

y

# Output: [1] "hello"
```

In the example above, we declared two variables: `x` and `y`. `x` is assigned the value of 10, while `y` is assigned the string "hello". To retrieve the value of these variables in the R console, simply type their names and hit Enter. The console will display the value of each variable.

Note that you can also use the `print()` function to display the value of a variable. For example:

```
print(x)


# Output: [1] 10
```

This will display the value of the variable `x` on the console.

4. Write R script to calculate the area of Rectangle.

Ans

```
length <- 5

width <- 3

area <- length * width

print(paste("The area of the rectangle is", area))
```

OUTPUT

```
[1] "The area of the rectangle is 15"
```

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

Ans

```
x <- 5

y <- "hello"

z <- c(1, 2, 3)

class(x)

# Output: [1] "numeric"


class(y)

# Output: [1] "character"


class(z)

# Output: [1] "numeric"
```

6.Enumerate The Process To Check Whether A Given Input Is Numeric , Integer ,

Double, Complex in R.

Ans

```
a <- 5
b <- 3.14
c <- 2+3i
d <- "hello"
is.numeric(a)
# Output: [1] TRUE
is.double(b)
# Output: [1] TRUE
is.complex(c)
# Output: [1] TRUE
is.numeric(d)
# Output: [1] FALSE
```

7. Illustration of Vector Arithmetic.

Ans

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
z1 <- x + y
# Output: [1] 5 7 9
z2 <- x - y
# Output: [1] -3 -3 -3
z3 <- x * y
# Output: [1]  4 10 18
z4 <- x / y
# Output: [1] 0.25 0.4  0.5
```

```
z5 <- x^y

# Output: [1]   1  32 729
```

8. 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”

Ans

```
name <- readline(prompt = "Enter your name: ")

age <- as.numeric(readline(prompt = "Enter your age: "))

next_year_age <- age + 1

message("Hi, ", name, ". Next year you will be ", next_year_age, " years old.")
```

Output

```
Enter your name: Jack

Enter your age: 17

Hi, Jack. Next year you will be 18 years old.
```

9) Perform Matrix Addition &amp; Subtraction in R

Ans

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)

# Output:

#      [,1] [,2]

# [1,]    1    3

# [2,]    2    4


B <- matrix(c(5, 6, 7, 8), nrow = 2)
```

```
# Output:
#      [,1] [,2]
# [1,]    5    7
# [2,]    6    8

C <- A + B
# Output:
#      [,1] [,2]
# [1,]    6   10
# [2,]    8   12

D <- A - B
# Output:
#      [,1] [,2]
# [1,]   -4   -4
# [2,]   -4   -4
```

10.Perform Scalar multiplication and matrix multiplication in R

Ans

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)
# Output:
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4
B <- 2 * A
# Output:
#      [,1] [,2]
# [1,]    2    6
# [2,]    4    8
```

```
C <- matrix(c(5, 6, 7, 8), nrow = 2)

# Output:

#      [,1] [,2]

# [1,]    5    7

# [2,]    6    8

D <- A %*% C

# Output:

#      [,1] [,2]

# [1,]   23   31

# [2,]   34   46
```

11. Find Transpose of matrix in R.

Ans

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)

# Output:

#      [,1] [,2]

# [1,]    1    3

# [2,]    2    4

B <- t(A)

# Output:

#      [,1] [,2]

# [1,]    1    2

# [2,]    3    4
```

12. Perform the operation of combining matrices in R using cbind() and rbind()

functions.

Ans

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)

# Output:
```

```
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4

B <- matrix(c(5, 6, 7, 8), nrow = 2)
# Output:
#      [,1] [,2]
# [1,]    5    7
# [2,]    6    8

C <- cbind(A, B)
# Output:
#      [,1] [,2] [,3] [,4]
# [1,]    1    3    5    7
# [2,]    2    4    6    8

D <- rbind(A, B)
# Output:
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4
# [3,]    5    7
# [4,]    6    8
```

13. Deconstruct a matrix in R

Ans

```
A <- matrix(c(1, 2, 3, 4), nrow = 2)
# Output:
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4


col1 <- A[, 1]

# Output: [1] 1 2


col2 <- A[, 2]

# Output: [1] 3 4


row1 <- A[1, ]

# Output: [1] 1 3


row2 <- A[2, ]

# Output: [1] 2 4
```

14. Perform array manipulation in R

Ans

In R, you can perform array manipulation using various functions and operators. Here are some examples:

1. Creating an array:

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

print(arr)
```

Output:

```
, , 1


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

[1,]    1    3    5

[2,]    2    4    6


, , 2
```

~~~
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```
~~~

2. Retrieving specific elements:

```
elem <- arr[2, 2, 3]
print(elem)
```

Output:

```
[1] 16
```

3. Subsetting arrays:

```
subset_arr <- arr[, 1:2, 1:2]
print(subset_arr)
```

Output:

```
, , 1

     [,1] [,2]
```

```
[1,]    1    3

[2,]    2    4


, , 2


     [,1] [,2]

[1,]    7    9

[2,]    8   10
```

4. Applying a function to an array:

```
sum_arr <- apply(arr, 3, sum)

print(sum_arr)
```

Output:

```
[1] 54 90 126 162
```

5. Reshaping an array:

```
reshaped_arr <- array(arr, dim = c(4, 6))

print(reshaped_arr)
```

Output:

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

[1,]    1   13    7   19    9   21

[2,]    2   14    8   20   10   22

[3,]    3   15    9   21   11   23

[4,]    4   16   10   22   12   24
```

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

Ans

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

layer_sum <- function(x) {

  sum(x)

}

sums <- apply(arr, MARGIN = 3, FUN = layer_sum)

print(sums)
```

OUTPUT

```
[1] 54 90 126 162
```

16. Demonstrate Factor data structure in R.

Ans

The `factor` data structure in R is used to represent categorical data, such as data that can take on a limited number of distinct values. Here's an example of how to create and manipulate a `factor` in R:

```
colors <- c("red", "blue", "green", "red", "blue", "green", "green", "red")

color_factor <- factor(colors)

print(color_factor)
```

In this example, we first create a vector `colors` that contains categorical data (in this case, the names of different colors). We then use the `factor()` function to convert this vector to a `factor` data structure, which assigns a unique integer value to each distinct category. Finally, we print the resulting `factor` using the `print()` function.

The output of this code would be:

```
[1] red   blue  green red   blue  green green red

Levels: blue green red
```

This output shows that the `factor` has correctly identified the distinct categories in the `colors` vector and assigned each one a unique integer value. It also shows the levels of the factor, which are the distinct categories in alphabetical order.

We can also manipulate the levels of a factor using the `levels()` function:

```
color_factor2 <- factor(colors, levels = c("red", "green", "blue"))

print(color_factor2)
```

In this example, we create a new `factor` called `color_factor2`, which has the same categories as the original `color_factor`, but with the levels specified in a specific order using the `levels` argument. The resulting output would be:

```
[1] red   blue  green red   blue  green green red

Levels: red green blue
```

This output shows that the levels of the `color_factor2` factor have been changed to the specified order.

In summary, the `factor` data structure in R is used to represent categorical data and can be created using the `factor()` function. The `levels()` function can be used to manipulate the levels of a `factor`.

18. Create a data frame and print the structure of the data frame in R.

Ans

```
my_df <- data.frame(

  name = c("Alice", "Bob", "Charlie"),

  age = c(25, 30, 35),

  married = c(TRUE, TRUE, FALSE),

  stringsAsFactors = FALSE

)

print(my_df)

str(my_df)
```

OUTPUT

```
'data.frame':	3 obs. of  3 variables:

 $ name   : chr  "Alice" "Bob" "Charlie"

 $ age    : num  25 30 35
 $ married: logi  TRUE TRUE FALSE
```

19. Demonstrate the creation of S3 class in R.

Ans

```
my_class <- function(x, y) {
  obj <- list(x = x, y = y)
  class(obj) <- "my_class"
  obj
}
print.my_class <- function(obj) {
  cat("x: ", obj$x, "\n")
  cat("y: ", obj$y, "\n")
}
my_obj <- my_class(1, 2)
print(my_obj)
```

OUTPUT

```
x:  1
y:  2
```

19. Demonstrate the creation of S4 class in R.

Ans

```
setClass("my_class",
  slots = list(
    x = "numeric",
    y = "character"
  )
)
```

```
my_obj <- new("my_class", x = 1, y = "hello")
print(my_obj)
```

OUTPUT

```
An object of class "my_class"
Slot "x":
[1] 1

Slot "y":
[1] "hello"
```

20. 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.

Ans

```
setRefClass("students",
  fields = list(
    Name = "character",
    Age = "numeric",
    GPA = "numeric"
  )
)
student1 <- new("students", Name = "John", Age = 20, GPA = 3.5)
student1$Name
student1$Age
student1$GPA
student1$Name <- "Paul"
```