
PART A

1. R Program for Different Types of Data Structures

```r
# Vector
my_vector <- c(1, 2, 3, 4, 5)
print(my_vector)

# List
my_list <- list(name = "John", age = 30, city = "New York")
print(my_list)

# Matrix
my_matrix <- matrix(1:6, nrow = 2, ncol = 3)
print(my_matrix)

# Data Frame
my_df <- data.frame(Name = c("Alice", "Bob", "Charlie"), Age = c(25, 30, 22))
print(my_df)

# Array
my_array <- array(1:12, dim = c(2, 3, 2))
print(my_array)

# Factor
my_factor <- factor(c("High", "Low", "Medium", "High", "Low"))
print(my_factor)
```

```
# DataFrame with time-series
date <- as.Date(c("2024-09-23", "2024-09-24", "2024-09-25"))
value <- c(100, 110, 105)
df_time_series <- data.frame(Date = date, Value = value)
print(df_time_series)
```

Output:

```
[1] 1 2 3 4 5
$name
[1] "John"
$age
[1] 30
$city
[1] "New York"
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
     Name Age
1   Alice  25
2     Bob  30
3 Charlie  22
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
[1] High   Low    Medium High   Low
Levels: High Low Medium
```

```
        Date Value
1 2024-09-23   100
2 2024-09-24   110
3 2024-09-25   105
```

---

2. R Program for Variables, Constants, and Data Types

```r
# Variables
name <- "Alice"
age <- 25
height <- 165.5
is_student <- TRUE

# Constants
PI <- 3.14159265359
G <- 9.81

# Data Types
char_vector <- c("apple", "banana", "cherry")
int_vector <- c(1, 2, 3, 4, 5)
double_vector <- c(1.5, 2.7, 3.0)
logical_vector <- c(TRUE, FALSE, TRUE, FALSE)

# Print variables, constants, and data types
cat("Name:", name, "\n")
cat("Age:", age, "\n")
cat("Height:", height, "\n")
cat("Is Student:", is_student, "\n")
cat("PI Constant:", PI, "\n")
cat("Gravity Constant:", G, "\n")
```

cat("Character Vector:", char_vector, "\n")
cat("Integer Vector:", int_vector, "\n")
cat("Double Vector:", double_vector, "\n")
cat("Logical Vector:", logical_vector, "\n")
```

Output:

```
Name: Alice
Age: 25
Height: 165.5
Is Student: TRUE
PI Constant: 3.141593
Gravity Constant: 9.81
Character Vector: apple banana cherry
Integer Vector: 1 2 3 4 5
Double Vector: 1.5 2.7 3
Logical Vector: TRUE FALSE TRUE FALSE
```

---

3. R Program for Arithmetic Operations

```
# Function with default argument values and complex object return
Calculate_area <- function(shape = “rectangle”, length = 0, width = 0) {
  If (shape == “rectangle”) {
    Area <- length * width
  } else if (shape == “circle”) {
    Area <- pi * (length ^ 2)
  } else {
    Area <- NA
  }
```
```

```
  Return(list(shape = shape, area = area))
}

# Example 1: Calculate the area of rectangle
Result_rect <- calculate_area( "rectangle" , length = 5, width = 3)
Cat( "Area of rectangle is: " , result_rect$area, "\n" )

# Example 2: Calculate the area of a circle
Result_circle <- calculate_area( "circle" , length = 4)
Cat( "Area of circle is: " , result_circle$area, "\n" )

# Example 3: Calculate the area of an unknown shape
Result_unknown <- calculate_area( "triangle" , length = 4)
Cat( "Area of an unknown shape is: " , result_unknown$area, "\n" )

# Control Structures
If (result_rect$area > result_circle$area) {
  Cat( "The rectangle has a larger area than the circle.\n" )
} else if (result_rect$area < result_circle$area) {
  Cat( "The circle has a larger area than the rectangle.\n" )
} else {
  Cat( "The rectangle and the circle have the same area.\n" )
}

# Arithmetic operators
Addition <- result_rect$area + result_circle$area
Subtraction <- result_rect$area – result_circle$area
Multiplication <- result_rect$area * result_circle$area
Division <- result_rect$area / result_circle$area

Cat( "Addition:" , addition, "\n" )
Cat( "Subtraction:" , subtraction, "\n" )
Cat( "Multiplication:" , multiplication, "\n" )
Cat( "Division:" , division, "\n" )
```

Output:
Area of rectangle is: 15
Area of circle is: 50.26548
Area of an unknown shape is: NA
The circle has a larger area than the rectangle.
Addition: 65.26548
Subtraction: -35.26548
Multiplication: 753.9822
Division: 0.2984155

---

4. R Program for Cumulative Sums, Products, Minima, Maxima, and Calculus

```r
# Create a sample numeric vector
data_vector <- c(3, 5, 7, 2, 10, 8)

# Cumulative Sum
cumulative_sum <- cumsum(data_vector)
cat("Cumulative Sum:", cumulative_sum, "\n")

# Cumulative Product
cumulative_product <- cumprod(data_vector)
cat("Cumulative Product is:", cumulative_product, "\n")

# Minimum and Maximum
minimum_value <- min(data_vector)
maximum_value <- max(data_vector)

cat("Minimum Value: ", minimum_value, "\n")
cat("Maximum Value: ", maximum_value, "\n")
```

# Calculus Differentiation
differentiate <- diff(data_vector)
cat("Differentiation (First Difference):", differentiate, "\n")
```

Output:

```
Cumulative Sum: 3 8 15 17 27 35
Cumulative Product is: 3 15 105 210 2100 16800
Minimum Value:  2
Maximum Value:  10
Differentiation (First Difference): 2 2 -5 8 -2
```

---

5. R Program for Finding Stationary Distribution of Markov Chains

```r
# Install and load the markovchain package
install.packages("markovchain")
library(markovchain)

# Define the transition probability matrix for the markov chain
p <- matrix(c(0.8, 0.2, 0.1, 0.9), nrow = 2, byrow = TRUE)

# Set the initial probability distribution
initial_distribution <- c(0.6, 0.4)

# Number of iterations for convergence
num_iterations <- 1000

# Initialize the distribution vector

```
current_distribution <- initial_distribution

# Perform iteration to estimate the stationary distribution
for(i in 1:num_iterations) {
  current_distribution <- current_distribution %*% p
}

# The final distribution is the estimated stationary distribution
stationary_distribution <- current_distribution

# Print the result
cat("Estimated Stationary Distribution:\n")
print(stationary_distribution)
```

Output:

```
Estimated Stationary Distribution:
          [,1]      [,2]
[1,] 0.3333333 0.6666667
```

---

6. R Program for Linear Algebra Operations

```r
# Create vectors
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)

# Perform vector addition
vector_sum <- vector1 + vector2
```

```
cat("Vector Addition:", vector_sum, "\n")

# Perform vector subtraction
vector_diff <- vector1 - vector2
cat("Vector Subtraction:", vector_diff, "\n")

# Perform vector multiplication
scalar <- 2
vector_scalar_product <- scalar * vector1
cat("Vector Scalar Product:", vector_scalar_product, "\n")

# Create matrices
matrix1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
matrix2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)

# Perform matrix addition
matrix_sum <- matrix1 + matrix2
cat("Matrix Addition:\n")
print(matrix_sum)

# Perform matrix subtraction
matrix_diff <- matrix1 - matrix2
cat("Matrix Subtraction:\n")
print(matrix_diff)

# Perform matrix multiplication
matrix_product <- matrix1 %*% matrix2
cat("Matrix Multiplication:\n")
print(matrix_product)

# Calculate matrix determinant
matrix_det <- det(matrix1)
cat("Matrix Determinant:", matrix_det, "\n")
```

```
# Calculate Matrix Transpose
matrix_transpose <- t(matrix1)
cat("Matrix Transpose:\n")
print(matrix_transpose)

# Calculate matrix inverse
matrix_inv <- solve(matrix1)
cat("Matrix Inverse:\n")
print(matrix_inv)
```

Output:

```
Vector Addition: 5 7 9
Vector Subtraction: -3 -3 -3
Vector Scalar Product: 2 4 6
Matrix Addition:
     [,1] [,2]
[1,]    6   10
[2,]    8   12
Matrix Subtraction:
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
Matrix Multiplication:
     [,1] [,2]
[1,]   23   31
[2,]   34   46
Matrix Determinant: -2
Matrix Transpose:
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
Matrix Inverse:
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

---

7. R Program for Visual Representations

```r
# Creating a vector of data
data <- c(10, 15, 20, 25, 30, 35, 40, 45, 50)

# Plotting a line chart
plot(data, type = "l", main = "Line Chart", xlab = "X-axis", ylab = "Y-axis")

# Creating a histogram
hist(data, main = "Histogram", xlab = "Values", ylab = "Frequency")

# Line Chart
time <- 1:10
values <- rnorm(10)
plot(time, values, type = "o", main = "Line Chart", xlab = "Time", ylab = "Value", col
= "green")

# Creating a pie chart
categories <- c("A", "B", "C")
values <- c(30, 40, 50)
pie(values, labels = categories, main = "Pie Chart", col = c("red", "green", "blue"))

# Creating a boxplot
data <- list(A = c(2, 4, 6, 8), B = c(1, 3, 5, 7))
boxplot(data, main = "Boxplot", xlab = "Groups", ylab = "Values")
```

```r
# Creating a scatter plot
x <- c(1, 2, 3, 4, 5)
y <- c(10, 20, 30, 20, 50)
plot(x, y, pch = 19, col = "blue", main = "Scatter Plot", xlab = "X-axis", ylab =
"Y-axis")
```

Output: Various plots will be displayed in the graphics device.

---

8. R Program for Data Frame Operations

```r
# Create a sample employee dataset as a data frame
emp_data <- data.frame(
  EmployeeId = c(1, 2, 3, 4, 5),
  FirstName = c("John", "Alice", "Bob", "Carol", "David"),
  LastName = c("Smith", "Johnson", "Johnson", "Smith", "Davis"),
  Age = c(30, 25, 28, 35, 32),
  Department = c("HR", "Marketing", "Finance", "HR", "IT"),
  Salary = c(50000, 55000, 60000, 52000, 70000)
)

# Print the entire employee dataset
cat("Employee Data:\n")
print(emp_data)

# Subset and index the data frame
cat("\nSubset and Indexing:\n")

# Select employees in the HR department
hr_emp <- emp_data[emp_data$Department == "HR", ]
```

```
cat("HR Employees:\n")
print(hr_emp)

# Select employees aged 30 or older
old_emp <- emp_data[emp_data$Age >= 30, ]
cat("Employees aged 30 or Older:\n")
print(old_emp)

# Select employees with salary greater than $55000
high_sal_emp <- emp_data[emp_data$Salary > 55000, ]
cat("Employees with Salary > $55000:\n")
print(high_sal_emp)

# Manipulate and analyze the data
cat("\nData Manipulation and Analysis:\n")

# Calculate the average salary
avg_sal <- mean(emp_data$Salary)
cat("Average Salary:", avg_sal, "\n")

# Calculate the Maximum age
max_age <- max(emp_data$Age)
cat("Maximum Age:", max_age, "\n")

# Calculate the no. of employees in each department
dept_count <- table(emp_data$Department)
cat("Number of Employees in each Department:\n")
print(dept_count)

# Calculate the total Payroll for each department
dept_payroll <- tapply(emp_data$Salary, emp_data$Department, sum)
cat("Total Payroll in Each Department:\n")
print(dept_payroll)
```

Output:

```
Employee Data:
  EmployeeId FirstName LastName Age Department Salary
1          1      John    Smith  30         HR  50000
2          2     Alice  Johnson  25  Marketing  55000
3          3       Bob  Johnson  28    Finance  60000
4          4     Carol    Smith  35         HR  52000
5          5     David    Davis  32         IT  70000

Subset and Indexing:
HR Employees:
  EmployeeId FirstName LastName Age Department Salary
1          1      John    Smith  30         HR  50000
4          4     Carol    Smith  35         HR  52000
Employees aged 30 or Older:
  EmployeeId FirstName LastName Age Department Salary
1          1      John    Smith  30         HR  50000
4          4     Carol    Smith  35         HR  52000
5          5     David    Davis  32         IT  70000
Employees with Salary > $55000:
  EmployeeId FirstName LastName Age Department Salary
3          3       Bob  Johnson  28    Finance  60000
5          5     David    Davis  32         IT  70000

Data Manipulation and Analysis:
Average Salary: 57400
Maximum Age: 35
Number of Employees in each Department:
  Finance        HR        IT Marketing
        1         2         1         1
Total Payroll in Each Department:
```

```
  Finance       HR       IT Marketing
    60000   102000    70000    55000
```

---

9. R Program for Multivariate Linear Regression

```r
# Sample dataset: Salary, Years of Experience, Education Level
data <- data.frame(
  Salary = c(50000, 60000, 75000, 80000, 95000, 110000, 120000, 130000),
  Experience = c(1, 2, 3, 4, 5, 6, 7, 8),
  Education = c(12, 14, 16, 16, 18, 20, 20, 22)
)

# Perform multivariate linear regression
model <- lm(Salary ~ Experience + Education, data = data)

# Predict salaries for new data
new_data <- data.frame(
  Experience = c(9, 10),
  Education = c(22, 24)
)

predicted_salaries <- predict(model, newdata = new_data)

# Print the predicted salaries
cat("Predicted Salaries:\n")
print(predicted_salaries)
```

Output:

```
Predicted Salaries:
       1        2
139333.3 152500.0
```

PART B

1. R Program to Find Factorial of a Number

Method 1: Using factorial() Function

```r
# Using factorial() method
answer1 <- factorial(4)
answer2 <- factorial(-3)
answer3 <- factorial(0)

print(answer1)
print(answer2)
print(answer3)

# Compute factorial of multiple values
answer1 <- factorial(c(0, 1, 2, 3, 4))
print(answer1)
```

Output:

```
[1] 24
[1] NaN
```

[1] 1
[1] 1 1 2 6 24
```

Method 2: Using if-else

```r
# take input from the user
num = as.integer(readline(prompt = "Enter a number: "))
factorial = 1

# check if the number is negative or positive
if(num < 0) {
  print("Not possible for negative numbers")
} else if(num == 0) {
  print("The factorial of 0 is 1")
} else {
  for(i in 1:num) {
    factorial = factorial * i
  }
  print(paste("The factorial of", num, "is", factorial))
}
```

Output (for input 5):

```
[1] "The factorial of 5 is 120"
```

---

2. R Program to Check Armstrong Number
```

```r
# Function to check for an Armstrong number
is_armstrong_number <- function(number) {
  num <- number
  num_of_digits <- nchar(as.character(num))
  sum_of_digits <- 0

  while (num > 0) {
    digit <- num %% 10
    sum_of_digits <- sum_of_digits + digit^num_of_digits
    num <- num %/% 10
  }

  return(sum_of_digits == number)
}

# Example usage
number_to_check <- 153
if (is_armstrong_number(number_to_check)) {
  cat(number_to_check, "is an Armstrong number.")
} else {
  cat(number_to_check, "is not an Armstrong number.")
}
```

Output:

```
153 is an Armstrong number.
```

---

3. R Program to Add Two Vectors

```r
# Create a vector 'x' of integer type and length 3
x = c(10, 20, 30)

# Create another vector 'y' of integer type and length 3
y = c(20, 10, 40)

# Print message indicating the original vectors
print("Original Vectors:")

# Print the contents of vector 'x'
print(x)

# Print the contents of vector 'y'
print(y)

# Print message indicating the result after adding the vectors
print("After adding two Vectors:")

# Add vectors 'x' and 'y' element-wise and store in 'z'
z = x + y

# Print the resulting vector 'z'
print(z)
```

Output:

```
[1] "Original Vectors:"
[1] 10 20 30
[1] 20 10 40
[1] "After adding two Vectors:"
```

```
[1] 30 30 70
```

---

4. Fibonacci Sequence Using Recursion

```r
fibonacci <- function(n) {
  if (n <= 0) {
    return(NULL)
  } else if (n == 1) {
    return(0)
  } else if (n == 2) {
    return(1)
  } else {
    return(fibonacci(n - 1) + fibonacci(n - 2))
  }
}

print_fibonacci_sequence <- function(n) {
  if (n <= 0) {
    cat("Invalid input. Please enter a positive integer.\n")
    return()
  }

  cat("Fibonacci Sequence:")
  for (i in 1:n) {
    cat(" ", fibonacci(i))
  }
  cat("\n")
}

# Change the value of 'n' to the desired number of terms in the sequence
```

n <- 10
print_fibonacci_sequence(n)
```

Output:

```
Fibonacci Sequence: 0 1 1 2 3 5 8 13 21 34
```

---

5. R Program to Find HCF or GCD

```r
hcf <- function(x, y) {
  while(y) {
    temp = y
    y = x %% y
    x = temp
  }
  return(x)
}

num1 = as.integer(readline(prompt = "Enter first number: "))
num2 = as.integer(readline(prompt = "Enter second number: "))
print(paste("The H.C.F. of", num1, "and", num2, "is", hcf(num1, num2)))
```

Output (for inputs 12 and 18):

```
[1] "The H.C.F. of 12 and 18 is 6"
```
```

---

6. R Program to Check for Leap Year

```r
# Function to check for a leap year
is_leap_year <- function(year) {
  if ((year %% 4 == 0 && year %% 100 != 0) || year %% 400 == 0) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

# Input year
input_year = as.integer(readline(prompt = "Enter the valid year: "))

# Check if it's a leap year
if (is_leap_year(input_year)) {
  print(paste(input_year, "is a leap year."))
} else {
  print(paste(input_year, "is not a leap year."))
}
```

Output (for input 2020):

```
[1] "2020 is a leap year."
```

---

7. R Program for Multiplication Table

```r
# program to print the multiplication table

# take input from user
number <- as.numeric(readline("Enter a number: "))
range <- as.numeric(readline("Enter the end range: "))

# Check if the input is a valid number
if (is.numeric(number)) {
  for (i in 1:range) {
    result <- number * i
    cat(number, "x", i, "=", result, "\n")
  }
} else {
  cat("Please enter a valid numeric value.\n")
}
```

Output (for inputs 5 and 10):

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

---

8. R Program to Check Prime Number

```r
Find_Prime_No <- function(n1) {
  if (n1 == 2) {
    return(TRUE)
  }
  if (n1 <= 1) {
    return(FALSE)
  }
  for (i in 2:(n1-1)) {
    if (n1 %% i == 0) {
      return(FALSE)
    }
  }
  return(TRUE)
}

numb_1 = as.numeric(readline("Enter a number: "))
if (Find_Prime_No(numb_1)) {
  print(paste(numb_1, "is a prime number"))
} else {
  print("It is not a prime number")
}
```

Output (for input 7):

```
[1] "7 is a prime number"
```