**Experiment No.4**

**Problem Statement:**

**Note: The students should perform for following statements using functions and recursive**

**functions in R.**

**Q1. A function that returns both the mean and standard deviation of a vector.**

* **Standard Function Approach**

**Code :**

calculate\_stats <- function(vec) {

if (length(vec) == 0) {

stop("The vector is empty.")

}

mean\_value <- mean(vec)

sd\_value <- sd(vec)

return(list(mean = mean\_value, sd = sd\_value))

}

# Example usage

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

stats <- calculate\_stats(vec)

print(stats)

**Output:**

$mean

[1] 3

$sd

[1] 1.581139

* **Recursive Function Approach**

**Code:**

# Helper function to calculate sum recursively

recursive\_sum <- function(vec) {

if (length(vec) == 0) {

return(0)

} else {

return(vec[1] + recursive\_sum(vec[-1]))

}

}

# Helper function to calculate sum of squares recursively

recursive\_sum\_of\_squares <- function(vec) {

if (length(vec) == 0) {

return(0)

} else {

return(vec[1]^2 + recursive\_sum\_of\_squares(vec[-1]))

}

}

# Main function to calculate mean and standard deviation recursively

calculate\_stats\_recursive <- function(vec) {

n <- length(vec)

if (n == 0) {

stop("The vector is empty.")

}

total\_sum <- recursive\_sum(vec)

total\_sum\_of\_squares <- recursive\_sum\_of\_squares(vec)

mean\_value <- total\_sum / n

variance <- (total\_sum\_of\_squares / n) - (mean\_value^2)

sd\_value <- sqrt(variance)

return(list(mean = mean\_value, sd = sd\_value))

}

# Example usage

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

stats\_recursive <- calculate\_stats\_recursive(vec)

print(stats\_recursive)

**Output:**

$mean

[1] 3

$sd

[1] 1.414214

**Q2. A function with arguments and without arguments.**

* **Standard Function Approach**

**Code :**

**# Function with arguments**

calculate\_sum <- function(a, b) {

return(a + b)

}

# Example usage

result\_with\_args <- calculate\_sum(5, 10)

print(result\_with\_args)

**Output:**

[1] 15

**# Function without arguments**

get\_default\_message <- function() {

return("Hello, Good Morning!")

}

# Example usage

message <- get\_default\_message()

print(message)

**Output:**

[1] "Hello, Good Morning!"

* **Recursive Function Approach**

**Code:**

**# Recursive Function with arguments**

# Recursive function to calculate factorial

factorial\_recursive <- function(n) {

if (n <= 1) {

return(1)

} else {

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

}

}

# Example usage

fact\_result <- factorial\_recursive(5)

print(fact\_result)

**Output:**

[1] 120

**# Recursive Function without arguments**

**Code:**

# Global variable

counter <- 0

# Recursive function to count down

count\_down\_recursive <- function(n) {

if (n < 0) {

return()

} else {

print(n)

count\_down\_recursive(n - 1)

}

}

# Example usage

count\_down\_recursive(5)

**Output:**

[1] 5

[1] 4

[1] 3

[1] 2

[1] 1

[1] 0

NULL

**Q3. Calculate reverse of given number using functions.**

* **Standard Function Approach**

**Code:**

# Standard function to reverse a number

reverse\_number <- function(num) {

reversed <- 0

while (num > 0) {

digit <- num %% 10

reversed <- reversed \* 10 + digit

num <- num %/% 10

}

return(reversed)

}

# Example usage

number <- 13579

reversed\_number <- reverse\_number(number)

print(reversed\_number)

**Output:**

[1] 97531

* **Recursive Function Approach**

**Code:**

# Recursive function to reverse a number

reverse\_number\_recursive <- function(num, reversed = 0) {

if (num == 0) {

return(reversed)

} else {

digit <- num %% 10

reversed <- reversed \* 10 + digit

return(reverse\_number\_recursive(num %/% 10, reversed))

}

}

# Example usage

number <- 12345

reversed\_number\_recursive <- reverse\_number\_recursive(number)

print(reversed\_number\_recursive)

**Output:**

[1] 54321

**Q4. Calculate Fibinocci Series using Recursion function in R**

* **Standard Function Approach**

**Code:**

# Standard function to generate Fibonacci series

fibonacci\_series <- function(n) {

fib <- numeric(n) # Create a numeric vector of length n

fib[1] <- 0

if (n > 1) fib[2] <- 1

for (i in 3:n) {

fib[i] <- fib[i - 1] + fib[i - 2]

}

return(fib)

}

# Example usage

num\_terms <- 10

fib\_series <- fibonacci\_series(num\_terms)

print(fib\_series)

**Output:**

[1] 0 1 1 2 3 5 8 13 21 34

* **Recursive Function Approach**

**Code:**

# Recursive function to calculate nth Fibonacci number

fibonacci\_recursive <- function(n) {

if (n <= 0) {

return(0)

} else if (n == 1) {

return(1)

} else {

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

}

}

# Function to generate Fibonacci series using recursion

fibonacci\_series\_recursive <- function(n) {

fib\_series <- numeric(n)

for (i in 0:(n - 1)) {

fib\_series[i + 1] <- fibonacci\_recursive(i)

}

return(fib\_series)

}

# Example usage

num\_terms <- 10

fib\_series\_rec <- fibonacci\_series\_recursive(num\_terms)

print(fib\_series\_rec)

**Output:**

[1] 0 1 1 2 3 5 8 13 21 34

**Q5. Calculate factorial of given number using recursion in R.**

* **Standard Function Approach**

**Code:**

# Standard function to calculate factorial

factorial <- function(n) {

if (n < 0) {

stop("Factorial is not defined for negative numbers.")

}

result <- 1

for (i in 1:n) {

result <- result \* i

}

return(result)

}

# Get input from user

user\_input <- as.integer(readline(prompt = "Enter a non-negative integer: "))

factorial\_result <- factorial(user\_input)

print(paste("Factorial of", user\_input, "is", factorial\_result))

**Output:**

Enter a non-negative integer: 8

Factorial of 8 is 40320

* **Recursive Function Approach**

**Code:**

# Recursive function to calculate factorial

factorial\_recursive <- function(n) {

if (n < 0) {

stop("Factorial is not defined for negative numbers.")

} else if (n == 0 || n == 1) {

return(1)

} else {

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

}

}

# Get input from user

user\_input\_rec <- as.integer(readline(prompt = "Enter a non-negative integer: "))

factorial\_result\_rec <- factorial\_recursive(user\_input\_rec)

print(paste("Factorial of", user\_input\_rec, "is", factorial\_result\_rec))

**Output:**

Enter a non-negative integer: 8

Factorial of 8 is 40320