**Functions in R Programming**

**Introduction**

Functions in R are a fundamental part of programming, enabling code reuse, modularity, and efficiency. Functions help encapsulate operations, making code easier to read and maintain.

**Defining a Function**

A function in R is created using the function keyword. The basic syntax is:

function\_name <- function(arg1, arg2, ...) {

# Function body

return(value)

}

**Example**

add\_numbers <- function(a, b) {

result <- a + b

return(result)

}

# Calling the function

sum\_value <- add\_numbers(5, 7)

print(sum\_value) # Output: 12

**Types of Functions**

1. **Built-in Functions**: R provides a wide range of built-in functions, such as sum(), mean(), sd(), and length().
   1. x <- c(1, 2, 3, 4, 5)
   2. mean\_value <- mean(x)
   3. print(mean\_value) # Output: 3
2. **User-defined Functions**: Custom functions created by the user for specific tasks.
3. **Anonymous Functions (Lambda Functions)**: Functions without a name, often used within apply() family functions.
   1. (function(x) x^2)(4) # Output: 16

**Function Arguments**

Functions in R can take various types of arguments:

* **Required Arguments**: Must be provided.
* **Default Arguments**: Assigned default values.
* **Variable Arguments**: ... allows passing multiple arguments.

Example with default arguments:

power\_function <- function(x, power=2) {

return(x^power)

}

print(power\_function(3)) # Output: 9 (default power=2)

print(power\_function(3, 3)) # Output: 27

**Scope of Variables**

R has two types of variable scopes:

* **Local Scope**: Variables defined within a function are not accessible outside.
* **Global Scope**: Variables defined outside functions are accessible globally.

Example:

my\_function <- function() {

local\_var <- 10

return(local\_var)

}

print(my\_function()) # Output: 10

print(local\_var) # Error: object 'local\_var' not found

**Recursive Functions**

A function can call itself, useful for tasks like computing factorial.

factorial\_func <- function(n) {

if (n == 0) return(1)

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

}

print(factorial\_func(5)) # Output: 120

**Problems to Solve**

1. Write a function to calculate the Fibonacci sequence up to a given number n.

|  |
| --- |
| fibonacci\_sequence <- function(n) {  if (n < 0) return("Input must be a non-negative integer")  fib <- c(0, 1)  while (TRUE) {  next\_fib <- sum(tail(fib, 2))  if (next\_fib > n) break  fib <- c(fib, next\_fib)  }  return(fib)  }  # Example  print(fibonacci\_sequence(20)) |

1. Create a function that takes a numeric vector and returns the sum of its squares.

|  |
| --- |
| sum\_of\_squares <- function(vec) {  return(sum(vec^2))  }  # Example  print(sum\_of\_squares(c(1, 2, 3))) |

1. Write a function that checks whether a number is prime.

|  |
| --- |
| is\_prime <- function(num) {  if (num <= 1) return(FALSE)  for (i in 2:sqrt(num)) {  if (num %% i == 0) return(FALSE)  }  return(TRUE)  }  # Example  print(is\_prime(17)) # TRUE  print(is\_prime(18)) # FALSE |

1. Implement a function to normalize a numeric vector (scale values between 0 and 1).

|  |
| --- |
| normalize\_vector <- function(vec) {  return((vec - min(vec)) / (max(vec) - min(vec)))  }  # Example  print(normalize\_vector(c(10, 20, 30, 40, 50))) |

1. Write a recursive function to compute the greatest common divisor (GCD) of two numbers.

|  |
| --- |
| gcd <- function(a, b) {  if (b == 0) return(a)  return(gcd(b, a %% b))  }  # Example  print(gcd(48, 18)) |

1. Write a function to count the number of vowels in a given string.

|  |
| --- |
| count\_vowels <- function(s) {  vowels <- c('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')  s\_chars <- strsplit(s, NULL)[[1]]  return(sum(s\_chars %in% vowels))  }  # Example  print(count\_vowels("Hello World")) |

1. Create a function that reverses a character string.

|  |
| --- |
| reverse\_string <- function(s) {  return(paste(rev(strsplit(s, NULL)[[1]]), collapse = ""))  }  # Example  print(reverse\_string("R programming")) |

1. Write a function that returns the factorial of a number using a loop (iterative approach).

|  |
| --- |
| factorial\_iter <- function(n) {  if (n == 0) return(1)  result <- 1  for (i in 1:n) {  result <- result \* i  }  return(result)  }  # Example  print(factorial\_iter(5)) |

1. Implement a function to find the maximum value in a numeric vector without using built-in max().

|  |
| --- |
| find\_max <- function(vec) {  max\_val <- vec[1]  for (val in vec) {  if (val > max\_val) max\_val <- val  }  return(max\_val)  }  # Example  print(find\_max(c(3, 7, 2, 9, 4))) |

1. Write a function to calculate the nth triangular number (sum of first n natural numbers).

|  |
| --- |
| triangular\_number <- function(n) {  return(n \* (n + 1) / 2)  }  # Example  print(triangular\_number(7)) |

1. Create a function that takes two numeric vectors and returns their dot product.

|  |
| --- |
| dot\_product <- function(vec1, vec2) {  if (length(vec1) != length(vec2)) stop("Vectors must be of equal length")  return(sum(vec1 \* vec2))  }  # Example  print(dot\_product(c(1, 2, 3), c(4, 5, 6))) |

1. Write a function that removes NA values from a vector and returns the cleaned vector.

|  |
| --- |
| remove\_na <- function(vec) {  return(vec[!is.na(vec)])  }  # Example  print(remove\_na(c(1, NA, 3, NA, 5))) |

1. Implement a function that checks if a string is a palindrome.

|  |
| --- |
| is\_palindrome <- function(s) {  s\_clean <- tolower(gsub("[^a-zA-Z0-9]", "", s))  return(s\_clean == paste(rev(strsplit(s\_clean, NULL)[[1]]), collapse = ""))  }  # Example  print(is\_palindrome("A man, a plan, a canal, Panama"))  print(is\_palindrome("Hello")) |

1. Write a function that converts temperature from Celsius to Fahrenheit.

|  |
| --- |
| celsius\_to\_fahrenheit <- function(c) {  return(c \* 9/5 + 32)  }  # Example  print(celsius\_to\_fahrenheit(25)) |

1. Create a function that takes a numeric vector and returns a named list with mean, median, and mode.

|  |
| --- |
| calculate\_stats <- function(vec) {  mode\_val <- function(x) {  ux <- unique(x)  ux[which.max(tabulate(match(x, ux)))]  }    return(list(  mean = mean(vec),  median = median(vec),  mode = mode\_val(vec)  ))  }  # Example  print(calculate\_stats(c(1, 2, 2, 3, 4, 4, 4, 5))) |