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))
```

2. 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)))
```

3. 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
```

4. 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)))
```

5. 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))
```

6. 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"))
```

7. 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"))
```

8. 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))
```

9. 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)))
```

10. 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))
```

11. 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)))
```

12. 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)))
```

13. 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"))
```

14. 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))
```

15. 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)))
```