FDA Lab-4

KHAN MOHD OWAIS RAZA
20BCD7138

ARRAYS

1] Create an array named "my_array" with the following elements: 1, 2, 3, 4, 5.

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Create the array
my_array <- array(c(1, 2, 3, 4, 5))
# Print the array
print(my_array)
```

```
R Console
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Create the array
> my_array <- array(c(1, 2, 3, 4, 5))
> # Print the array
> print(my_array)
[1] 1 2 3 4 5
>
```

2] Access and print the third element of the array "my_array"

```
#KHAN MOHD OWAIS RAZA
third_element <- my_array[3]
print(third_element)
```

```
> #KHAN MOHD OWAIS RAZA
> third_element <- my_array[3]
> print(third_element)
[1] 3
>
```

3] Update the fourth element of the array "my_array" to 10

```
#KHAN MOHD OWAIS RAZA
# Update the fourth element of the array
my_array[4] <- 10
my_array
```

```
> #KHAN MOHD OWAIS RAZA
> # Update the fourth element of the array
> my_array[4] <- 10
> my_array
[1]  1  2  3 10  5
>
```

4] Find the length of the array "my_array"

```
#KHAN MOHD OWAIS RAZA
# Find the length of the array
array_length <- length(my_array)
print(array_length)
```

```
> #KHAN MOHD OWAIS RAZA
> # Find the length of the array
> array_length <- length(my_array)
> print(array_length)
[1] 5
>
```

5] Create a new array named "my_matrix" with the following elements: 1, 2, 3, 4, 5, 6. Reshape the array into a 2x3 matrix

```
#KHAN MOHD OWAIS RAZA
# Create the array
my_matrix <- array(c(1, 2, 3, 4, 5, 6), dim = c(2, 3))
print(my_matrix)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create the array
> my_matrix <- array(c(1, 2, 3, 4, 5, 6), dim = c(2, 3))
> print(my_matrix)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

6] Calculate the sum of the elements in each column of the matrix "my_matrix"

```
#KHAN MOHD OWAIS RAZA
# Calculate the sum of elements in each column
column_sums <- colSums(my_matrix)
print(column_sums)
```

```
> #KHAN MOHD OWAIS RAZA
> # Calculate the sum of elements in each column
> column_sums <- colSums(my_matrix)
> print(column_sums)
[1]  3  7 11
```

7] Create a new array named "my_array2" with the following elements: 10, 20, 30, 40, 50. Concatenate "my_array" and "my_array2" into a single array.

```
#KHAN MOHD OWAIS RAZA
# Create my_array
my_array <- c(1, 2, 3, 4, 5)
# Create my_array2
my_array2 <- c(10, 20, 30, 40, 50)
# Concatenate my_array and my_array2
combined_array <- c(my_array, my_array2)
print(combined_array)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create my_array
> my_array <- c(1, 2, 3, 4, 5)
> # Create my_array2
> my_array2 <- c(10, 20, 30, 40, 50)
> # Concatenate my_array and my_array2
> combined_array <- c(my_array, my_array2)
> print(combined_array)
 [1]  1  2  3  4  5 10 20 30 40 50
>
```

8] Find the maximum value in the array "concat_array"

```
#KHAN MOHD OWAIS RAZA
# Create concat_array
concat_array <- c(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
# Find the maximum value
max_value <- max(concat_array)
print(max_value)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create concat_array
> concat_array <- c(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
> # Find the maximum value
> max_value <- max(concat_array)
> print(max_value)
[1] 50
>
```

9] Sort the elements in the array "concat_array" in ascending order

```
#KHAN MOHD OWAIS RAZA
# Create concat_array
concat_array <- c(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
# Sort the elements in ascending order
sorted_array <- sort(concat_array)
print(sorted_array)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create concat_array
> concat_array <- c(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
> # Sort the elements in ascending order
> sorted_array <- sort(concat_array)
> print(sorted_array)
 [1]  1  2  3  4  5 10 20 30 40 50
>
```

10] Calculate the average of the elements in the array "concat_array"

```
#KHAN MOHD OWAIS RAZA
# Create concat_array
concat_array <- c(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
# Calculate the average
average <- mean(concat_array)
print(average)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create concat_array
> concat_array <- c(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
> # Calculate the average
> average <- mean(concat_array)
> print(average)
[1] 16.5
>
```

LISTS

1] Create a list named "my_list" with the following elements: "apple", 10, TRUE

```
#KHAN MOHD OWAIS RAZA
# Create my_list
my_list <- list("apple", 10, TRUE)
print(my_list)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create my_list
> my_list <- list("apple", 10, TRUE)
> print(my_list)
[[1]]
[1] "apple"

[[2]]
[1] 10

[[3]]
[1] TRUE
```

2] Access and print the second element of the list "my_list"

```
#KHAN MOHD OWAIS RAZA
# Access and print the second element of my_list
second_element <- my_list[[2]]
print(second_element)
```

```
> # Access and print the second element of my_list
> second_element <- my_list[[2]]
> print(second_element)
[1] 10
>
```

3] Update the third element of the list "my_list" to FALSE

```
#KHAN MOHD OWAIS RAZA
# Update the third element of my_list to FALSE
my_list[[3]] <- FALSE
```

```
> #KHAN MOHD OWAIS RAZA
> # Update the third element of my_list to FALSE
> my_list[[3]] <- FALSE
> my_list
[[1]]
[1] "apple"

[[2]]
[1] 10

[[3]]
[1] FALSE
```

4] Find the length of the list "my_list"

```
#KHAN MOHD OWAIS RAZA
# Create the list
my_list <- list("apple", 10, FALSE)
# Find the length of the list
list_length <- length(my_list)
# Print the length of the list
print(list_length)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create the list
> my_list <- list("apple", 10, FALSE)
> # Find the length of the list
> list_length <- length(my_list)
> # Print the length of the list
> print(list_length)
[1] 3
>
```

5] Add a new element, "orange", to the end of the list "my_list"

```
#KHAN MOHD OWAIS RAZA
# Create the list
my_list <- list("apple", 10, FALSE)
# Add "orange" to the end of the list
my_list <- append(my_list, "orange")
# Print the updated list
print(my_list)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create the list
> my_list <- list("apple", 10, FALSE)
> # Add "orange" to the end of the list
> my_list <- append(my_list, "orange")
> # Print the updated list
> print(my_list)
[[1]]
[1] "apple"

[[2]]
[1] 10

[[3]]
[1] FALSE

[[4]]
[1] "orange"

>
```

6] Create a nested list named "nested_list" with two elements: a numeric vector (1, 2, 3) and a character vector ("a", "b", "c")

```
#KHAN MOHD OWAIS RAZA
# Create the nested list
nested_list <- list(
  numeric_vector = c(1, 2, 3),
  character_vector = c("a", "b", "c")
)
# Print the nested list
print(nested_list)
```

```
> #KHAN MOHD OWAIS RAZA
> # Create the nested list
> nested_list <- list(
+   numeric_vector = c(1, 2, 3),
+   character_vector = c("a", "b", "c")
+ )
> # Print the nested list
> print(nested_list)
$numeric_vector
[1] 1 2 3

$character_vector
[1] "a" "b" "c"
```

7] Access and print the second element of the numeric vector in the nested list "nested_list"

```
#KHAN MOHD OWAIS RAZA
# Access and print the second element of the numeric vector
second_element <- nested_list$numeric_vector[2]
print(second_element)
```

```
> #KHAN MOHD OWAIS RAZA
> # Access and print the second element of the numeric vector
> second_element <- nested_list$numeric_vector[2]
> print(second_element)
[1] 2
>
```

8] Add a new element, a logical vector (TRUE, FALSE, TRUE), to the nested list "nested_list"

```
#KHAN MOHD OWAIS RAZA
# Create a logical vector
logical_vector <- c(TRUE, FALSE, TRUE)
# Add the logical vector to the nested list
nested_list$log_vector <- logical_vector
```

```
> #KHAN MOHD OWAIS RAZA
> # Create a logical vector
> logical_vector <- c(TRUE, FALSE, TRUE)
> # Add the logical vector to the nested list
> nested_list$log_vector <- logical_vector
>
> logical_vector
[1]  TRUE FALSE  TRUE
```

9] Remove the second element from the character vector in the nested list "nested_list"

```
#KHAN MOHD OWAIS RAZA
# Remove the second element from the character vector
nested_list[[2]] <- nested_list[[2]][-2]
nested_list
```

```
> #KHAN MOHD OWAIS RAZA
> # Remove the second element from the character vector
> nested_list[[2]] <- nested_list[[2]][-2]
> nested_list
$numeric_vector
[1] 1 2 3

$character_vector
[1] "a" "c"

$log_vector
[1]  TRUE FALSE  TRUE
```

10] Combine the elements of the nested list "nested_list" into a single vector

```
#KHAN MOHD OWAIS RAZA
# Combine elements of nested_list into a single vector
combined_vector <- unlist(nested_list)
```

```
> #KHAN MOHD OWAIS RAZA
> # Combine elements of nested_list into a single vector
> combined_vector <- unlist(nested_list)
>
> combined_vector
  numeric_vector1   numeric_vector2   numeric_vector3 character_vector1
              "1"               "2"               "3"               "a"
character_vector2       log_vector1       log_vector2       log_vector3
              "c"            "TRUE"           "FALSE"            "TRUE"
> |
```

FACTORS

Create a factor vector "f" with the following levels: "Low", "Medium", "High", "Low", "Medium".

a] Print the factor.

b] Check the levels of the factor vector "f".

c] Convert the factor vector "f" to numeric representation.

d] Count the frequency of each level in the factor vector "f".

e] Replace the level "Low" in the factor vector "f" with "Very Low".

f] Sort the levels of the factor vector "f" in alphabetical order.

g] Reorder the factor vector "f" based on the frequency of each level.

h] Find the most frequent level in the factor vector "f".

i] Create a factor vector "f2" with levels "Yes", "No", and "Maybe" in a specified order.

j] Convert the factor vector "f2" to a character vector.

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Create the factor vector
f <- factor(c("Low", "Medium", "High", "Low", "Medium"), levels =
c("Low", "Medium", "High"))
# Print the factor vector
print(f)
```

```
# A] Print the factor
f <- factor(c("Low", "Medium", "High", "Low", "Medium"), levels =
c("Low", "Medium", "High"))
print(f)
# B] Check the levels of the factor vector "f"
f <- factor(c("Low", "Medium", "High", "Low", "Medium"), levels =
c("Low", "Medium", "High"))
levels(f)
# C] Convert the factor vector "f" to numeric representation
numeric_f <- as.numeric(f)
numeric_f
# D] Count the frequency of each level in the factor vector "f"
frequency <- table(f)
frequency
# E] Replace the level "Low" in the factor vector "f" with "Very
Low"
levels(f) <- c("Very Low", "Medium", "High")
f
# F] Sort the levels of the factor vector "f" in alphabetical
order
f <- factor(f, levels = sort(levels(f)))
f
# G] Reorder the factor vector "f" based on the frequency of each
level
f <- reorder(f, FUN = function(x) -length(x))
f
# H] Find the most frequent level in the factor vector "f"
freq <- table(f)
most_frequent_index <- which.max(freq)
most_frequent_level <- levels(f)[most_frequent_index]
most_frequent_level
# I] Create a factor vector "f2" with levels "Yes", "No", and
"Maybe" in a specified order
f2 <- factor(c("Yes", "No", "Maybe"), levels = c("Yes", "No",
"Maybe"))
f2
# J] Convert the factor vector "f2" to a character vector
```

```
f2_char <- as.character(f2)
f2_char
```

R Console

```
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Create the factor vector
> f <- factor(c("Low", "Medium", "High", "Low", "Medium"), levels = c("Low", "Medium", "High"))
> # Print the factor vector
> print(f)
[1] Low    Medium High   Low    Medium
Levels: Low Medium High
> # A] Print the factor
> f <- factor(c("Low", "Medium", "High", "Low", "Medium"), levels = c("Low", "Medium", "High"))
> print(f)
[1] Low    Medium High   Low    Medium
Levels: Low Medium High
>
> # B] Check the levels of the factor vector "f"
> f <- factor(c("Low", "Medium", "High", "Low", "Medium"), levels = c("Low", "Medium", "High"))
> levels(f)
[1] "Low"    "Medium" "High"
>
> # C] Convert the factor vector "f" to numeric representation
> numeric_f <- as.numeric(f)
> numeric_f
[1] 1 2 3 1 2
>
> # D] Count the frequency of each level in the factor vector "f"
> frequency <- table(f)
> frequency
f
   Low Medium   High
     2      2      1
> # E] Replace the level "Low" in the factor vector "f" with "Very Low"
> levels(f) <- c("Very Low", "Medium", "High")
> f
[1] Very Low Medium   High     Very Low Medium
Levels: Very Low Medium High
>
> # F] Sort the levels of the factor vector "f" in alphabetical order
> f <- factor(f, levels = sort(levels(f)))
> f
[1] Very Low Medium   High     Very Low Medium
Levels: High Medium Very Low
>
> # G] Reorder the factor vector "f" based on the frequency of each level
> f <- reorder(f, FUN = function(x) -length(x))
Error in tapply(X = X, INDEX = x, FUN = FUN, ...) :
  argument "X" is missing, with no default
> f
[1] Very Low Medium   High     Very Low Medium
Levels: High Medium Very Low
>
> # H] Find the most frequent level in the factor vector "f"
> freq <- table(f)
> most_frequent_index <- which.max(freq)
> most_frequent_level <- levels(f)[most_frequent_index]
> most_frequent_level
[1] "Medium"
```

```
>
> # I] Create a factor vector "f2" with levels "Yes", "No", and "Maybe" in a specified order
> f2 <- factor(c("Yes", "No", "Maybe"), levels = c("Yes", "No", "Maybe"))
> f2
[1] Yes   No    Maybe
Levels: Yes No Maybe
>
> # J] Convert the factor vector "f2" to a character vector
> f2_char <- as.character(f2)
> f2_char
[1] "Yes"   "No"    "Maybe"
>
```

FUNCTIONS

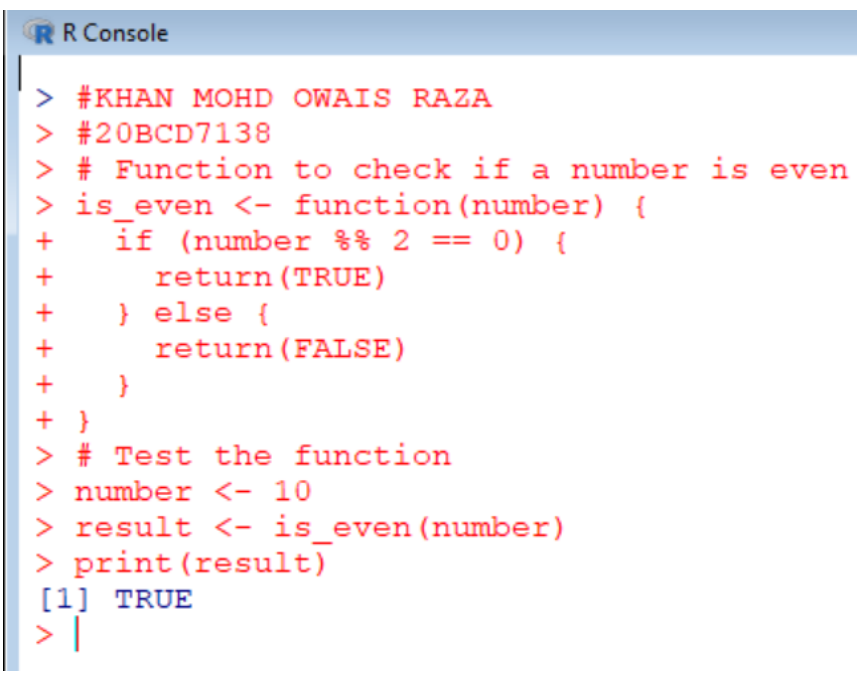
1] Write a function in R called "calculate_area" that takes two arguments, "length" and "width", and calculates the area of a rectangle.

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to calculate the area of a rectangle
calculate_area <- function(length, width) {
  area <- length * width
  return(area)
}
# Test the function
length <- 5
width <- 10
rectangle_area <- calculate_area(length, width)
print(rectangle_area)
```

```
R Console
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Function to calculate the area of a rectangle
> calculate_area <- function(length, width) {
+   area <- length * width
+   return(area)
+ }
> # Test the function
> length <- 5
> width <- 10
> rectangle_area <- calculate_area(length, width)
> print(rectangle_area)
[1] 50
>
```

2] Write a function in R called "is_even" that takes a single argument "number" and returns TRUE if the number is even, and FALSE otherwise

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to check if a number is even
is_even <- function(number) {
  if (number %% 2 == 0) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}
# Test the function
number <- 10
result <- is_even(number)
print(result)
```



3] Write a function in R called "calculate_factorial" that calculates the factorial of a given positive integer

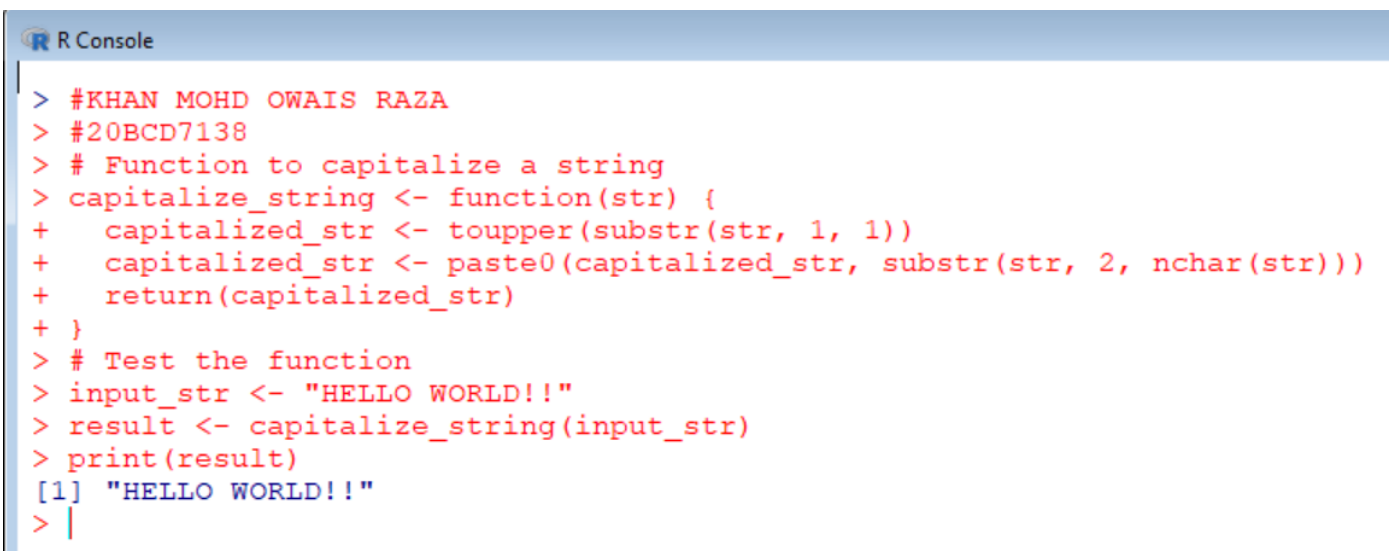
```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to calculate factorial
calculate_factorial <- function(number) {
  if (number < 0) {
    stop("Factorial is not defined for negative numbers.")
  }
  if (number == 0) {
    return(1)
  } else {
    factorial <- 1
    for (i in 1:number) {
      factorial <- factorial * i
    }
    return(factorial)
  }
}
# Test the function
number <- 5
result <- calculate_factorial(number)
print(result)
```

R Console

```
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Function to calculate factorial
> calculate_factorial <- function(number) {
+   if (number < 0) {
+     stop("Factorial is not defined for negative numbers.")
+   }
+   if (number == 0) {
+     return(1)
+   } else {
+     factorial <- 1
+     for (i in 1:number) {
+       factorial <- factorial * i
+     }
+     return(factorial)
+   }
+ }
> # Test the function
> number <- 5
> result <- calculate_factorial(number)
> print(result)
[1] 120
>
```

4] Write a function in R called "capitalize_string" that takes a string as an argument and returns the capitalized version of the string

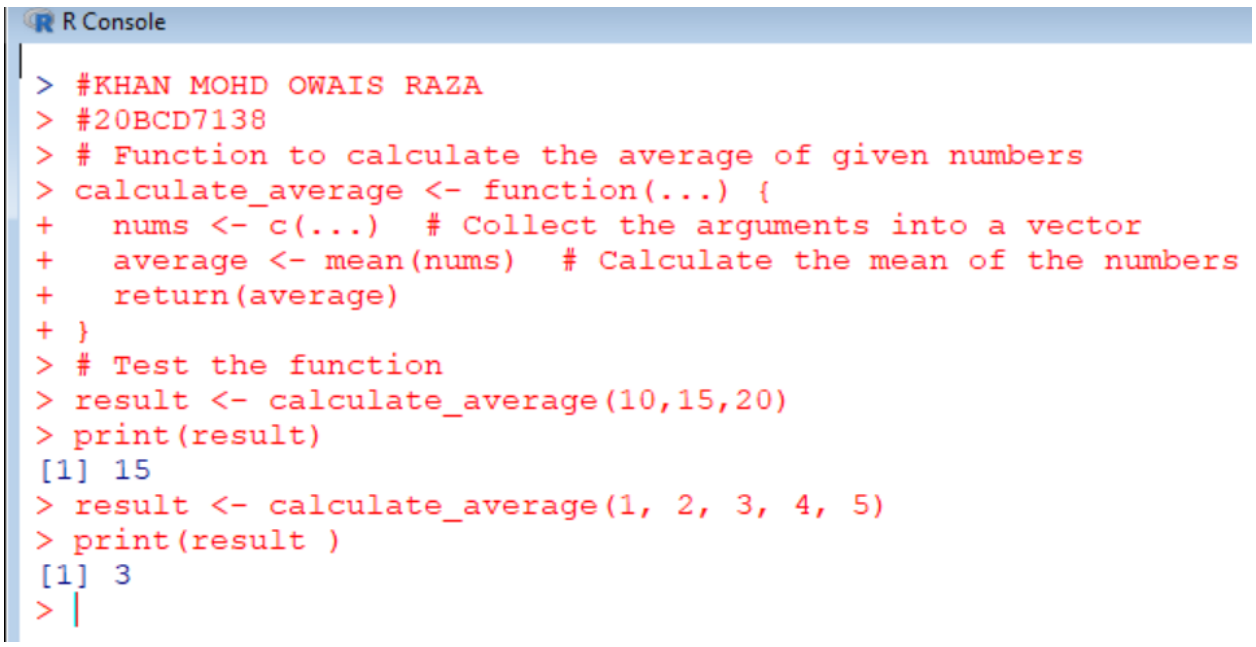
```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to capitalize a string
capitalize_string <- function(str) {
  capitalized_str <- toupper(substr(str, 1, 1))
  capitalized_str <- paste0(capitalized_str, substr(str, 2,
nchar(str)))
  return(capitalized_str)
}
# Test the function
input_str <- "HELLO WORLD!!"
result <- capitalize_string(input_str)
print(result)
```



5] Write a function in R called "calculate_average" that takes a variable number of arguments and calculates the average of the given numbers

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to calculate the average of given numbers
calculate_average <- function(...) {
```

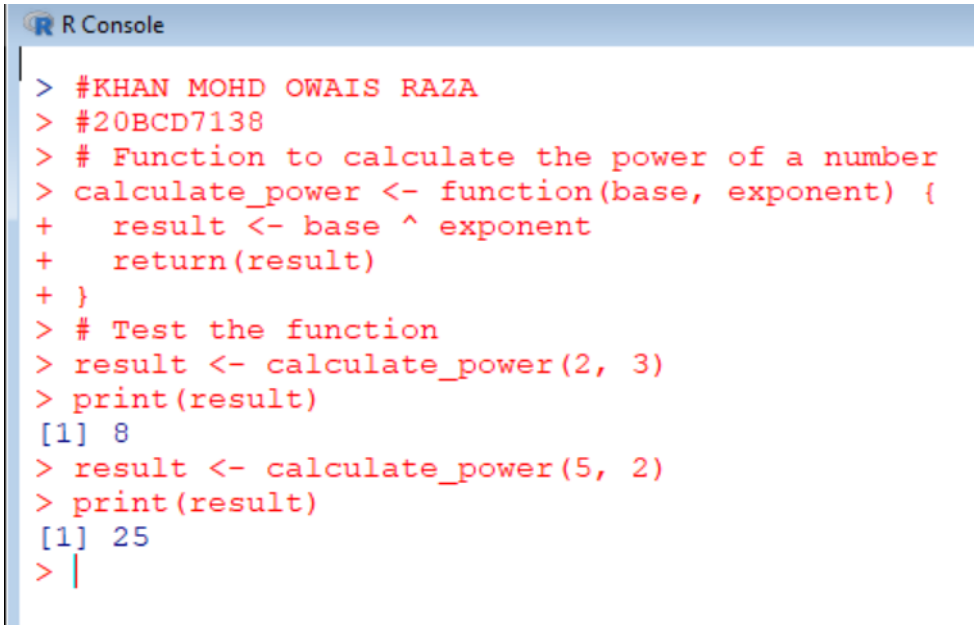
```
  nums <- c(...)  # Collect the arguments into a vector
  average <- mean(nums)  # Calculate the mean of the numbers
  return(average)
}
# Test the function
result <- calculate_average(10,15,20)
print(result)
result <- calculate_average(1, 2, 3, 4, 5)
print(result)
```



6] Write a function in R called "calculate_power" that takes two arguments, "base" and "exponent", and calculates the result of raising the base to the exponent

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to calculate the power of a number
calculate_power <- function(base, exponent) {
  result <- base ^ exponent
  return(result)
}
# Test the function
result <- calculate_power(2, 3)
```

```
print(result)
result <- calculate_power(5, 2)
print(result)
```



7] Write a function in R called "check_prime" that takes a positive integer as an argument and checks if it is a prime number. The function should return TRUE if the number is prime, and FALSE otherwise

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to check if a number is prime
check_prime <- function(number) {
  if (number <= 1) {
    return(FALSE)  # Numbers less than or equal to 1 are not
prime
  }
  # Check divisibility by numbers up to the square root of the
given number
  for (i in 2:sqrt(number)) {
    if (number %% i == 0) {
      return(FALSE)  # Number is divisible by i, hence not prime
    }
```

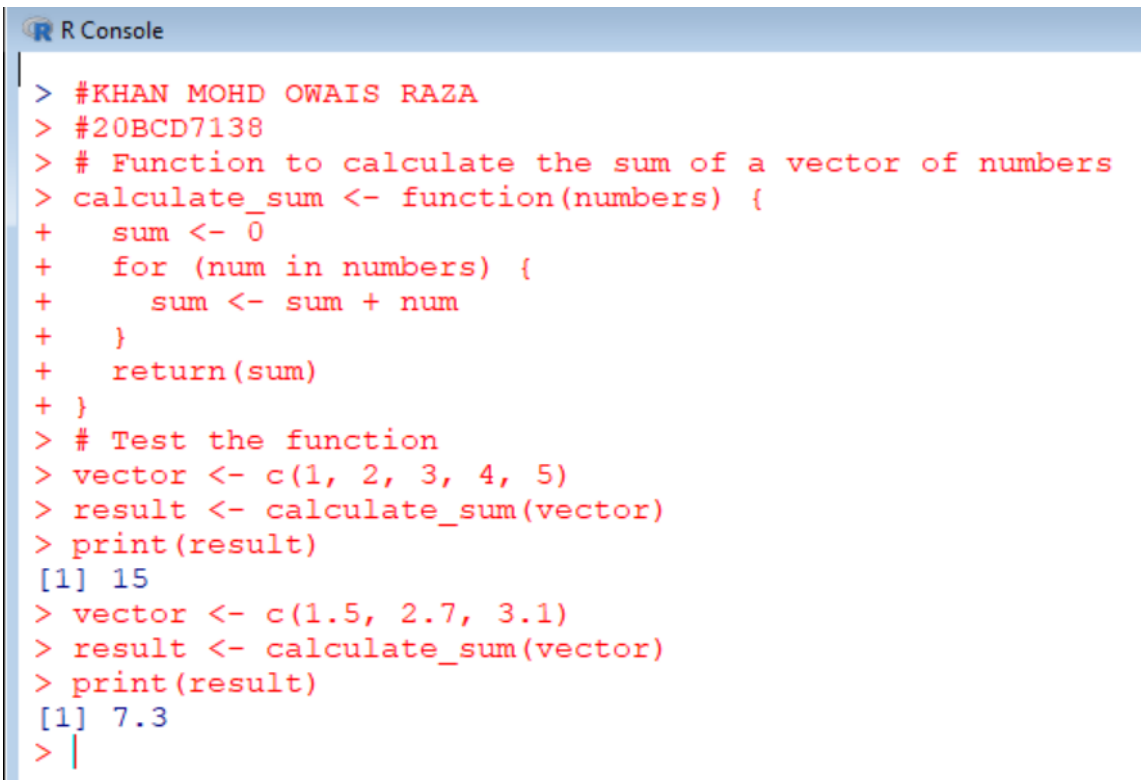
```
  }

  return(TRUE)  # Number is prime
}
# Test the function
result <- check_prime(7)
print(result)
```

```
R Console
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Function to check if a number is prime
> check_prime <- function(number) {
+   if (number <= 1) {
+     return(FALSE)  # Numbers less than or equal to 1 are not prime
+   }
+   # Check divisibility by numbers up to the square root of the given number
+   for (i in 2:sqrt(number)) {
+     if (number %% i == 0) {
+       return(FALSE)  # Number is divisible by i, hence not prime
+     }
+   }
+
+   return(TRUE)  # Number is prime
+ }
> # Test the function
> result <- check_prime(7)
> print(result)
[1] TRUE
>
```

8] Write a function in R called "calculate_sum" that takes a vector of numbers as an argument and calculates the sum of the numbers. The function should handle both integer and decimal numbers

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to calculate the sum of a vector of numbers
calculate_sum <- function(numbers) {
  sum <- 0
  for (num in numbers) {
    sum <- sum + num
  }
```

```
  return(sum)
}
# Test the function
vector <- c(1, 2, 3, 4, 5)
result <- calculate_sum(vector)
print(result)
vector <- c(1.5, 2.7, 3.1)
result <- calculate_sum(vector)
print(result)
```



9] Write a function in R called "convert_temperature" that takes a temperature value in Celsius and converts it to Fahrenheit. The formula for conversion is: Fahrenheit = (Celsius * 9/5) + 32

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to convert temperature from Celsius to Fahrenheit
convert_temperature <- function(celsius) {
  fahrenheit <- (celsius * 9/5) + 32
```

```
  return(fahrenheit)
}
# Test the function
celsius_temp <- 25
fahrenheit_temp <- convert_temperature(celsius_temp)
print(fahrenheit_temp)
```

```
R Console
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Function to convert temperature from Celsius to Fahrenheit
> convert_temperature <- function(celsius) {
+   fahrenheit <- (celsius * 9/5) + 32
+   return(fahrenheit)
+ }
> # Test the function
> celsius_temp <- 25
> fahrenheit_temp <- convert_temperature(celsius_temp)
> print(fahrenheit_temp)
[1] 77
>
```

10] Write a function in R called "reverse_string" that takes a string as an argument and returns the reverse of the string

```
#KHAN MOHD OWAIS RAZA
#20BCD7138
# Function to reverse a string
reverse_string <- function(input_string) {
  reversed_string <- strsplit(input_string, "")[[1]]
  reversed_string <-
paste(reversed_string[length(reversed_string):1], collapse = "")
  return(reversed_string)
}
# Test the function
input <- "Hello, World!"
reversed <- reverse_string(input)
print(reversed)
```

R Console

```
> #KHAN MOHD OWAIS RAZA
> #20BCD7138
> # Function to reverse a string
> reverse_string <- function(input_string) {
+   reversed_string <- strsplit(input_string, "")[[1]]
+   reversed_string <- paste(reversed_string[length(reversed_string):1], collapse = "")
+   return(reversed_string)
+ }
> # Test the function
> input <- "Hello, World!"
> reversed <- reverse_string(input)
> print(reversed)
[1] "!dlroW ,olleH"
>
```