*** R Programming Language ***

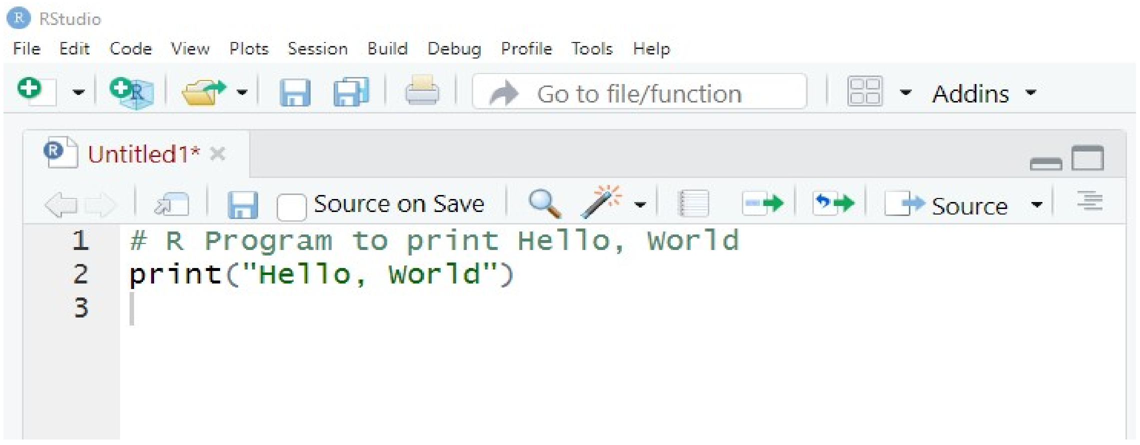
INDEX PAGE

01.=> Print "Hello World" to the Screen.

> Here a simple R Program that prints "Hello, World":

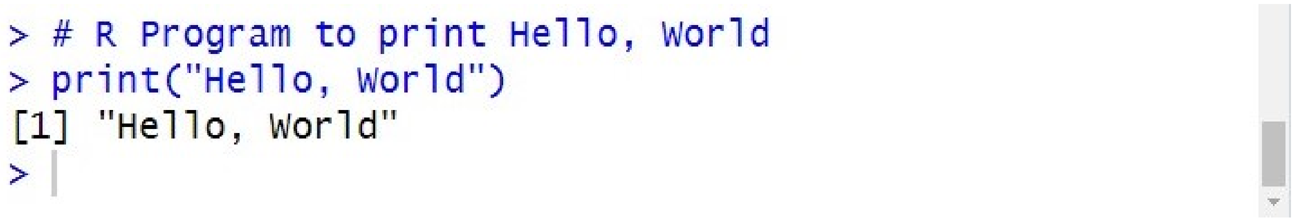
Syntax=> print("Hello, World")

Code >


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1*
Source on Save
Source

```
# R Program to print Hello, World
print("Hello, World")
```

Output >

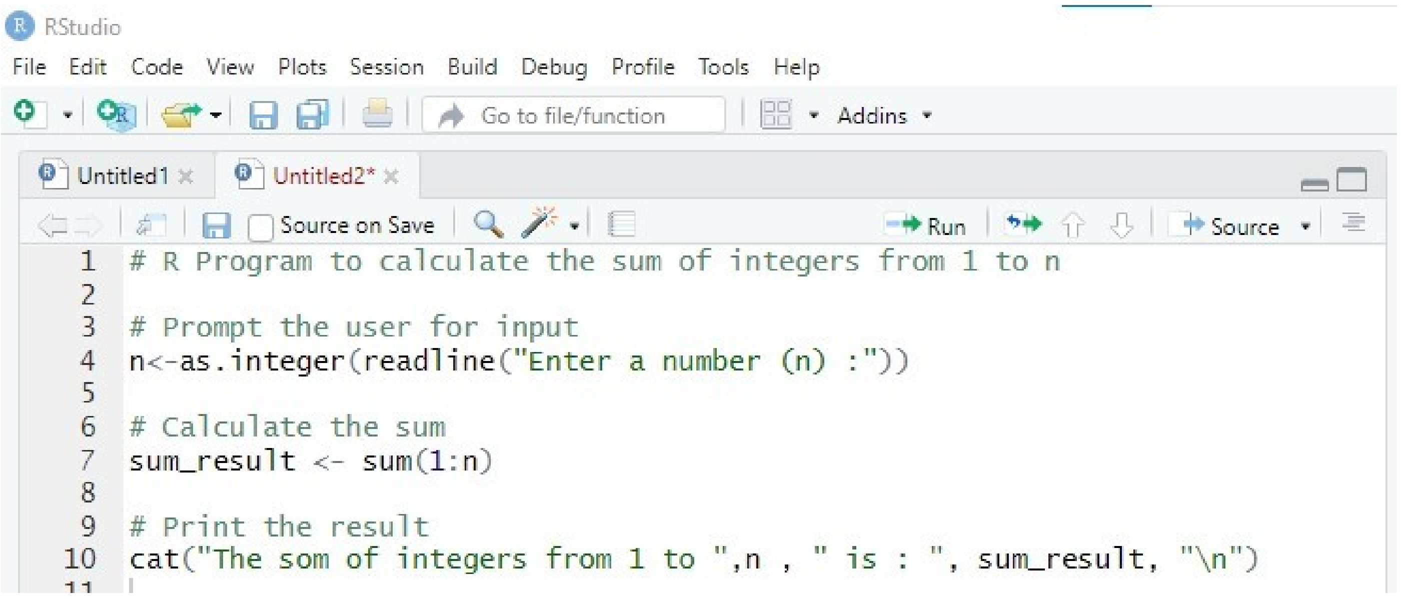


```
> # R Program to print Hello, World
> print("Hello, World")
[1] "Hello, World"
> 
```

Q2· > Write a program that asks the user for a number n and prints the sum of the 1 to n·

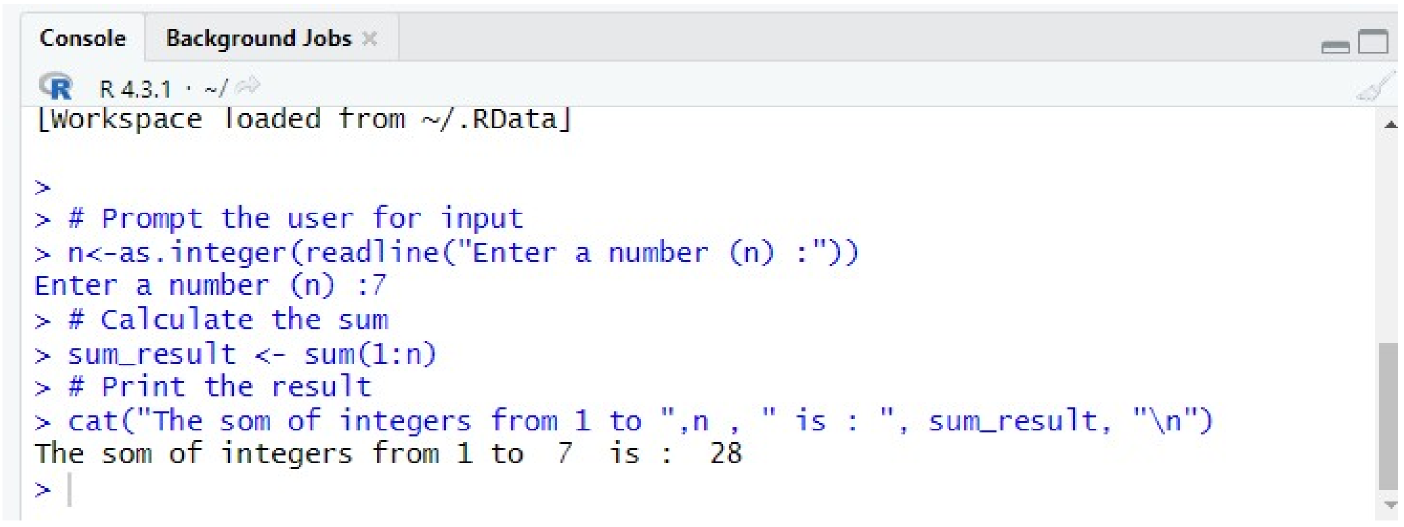
>that prompts the user for a number n and prints the sumb of integers from 1 to n;

Code >



```r
# R Program to calculate the sum of integers from 1 to n

# Prompt the user for input
n<-as.integer(readline("Enter a number (n) :"))

# Calculate the sum
sum_result <- sum(1:n)

# Print the result
cat("The som of integers from 1 to ",n , " is : ", sum_result, "\n")
```

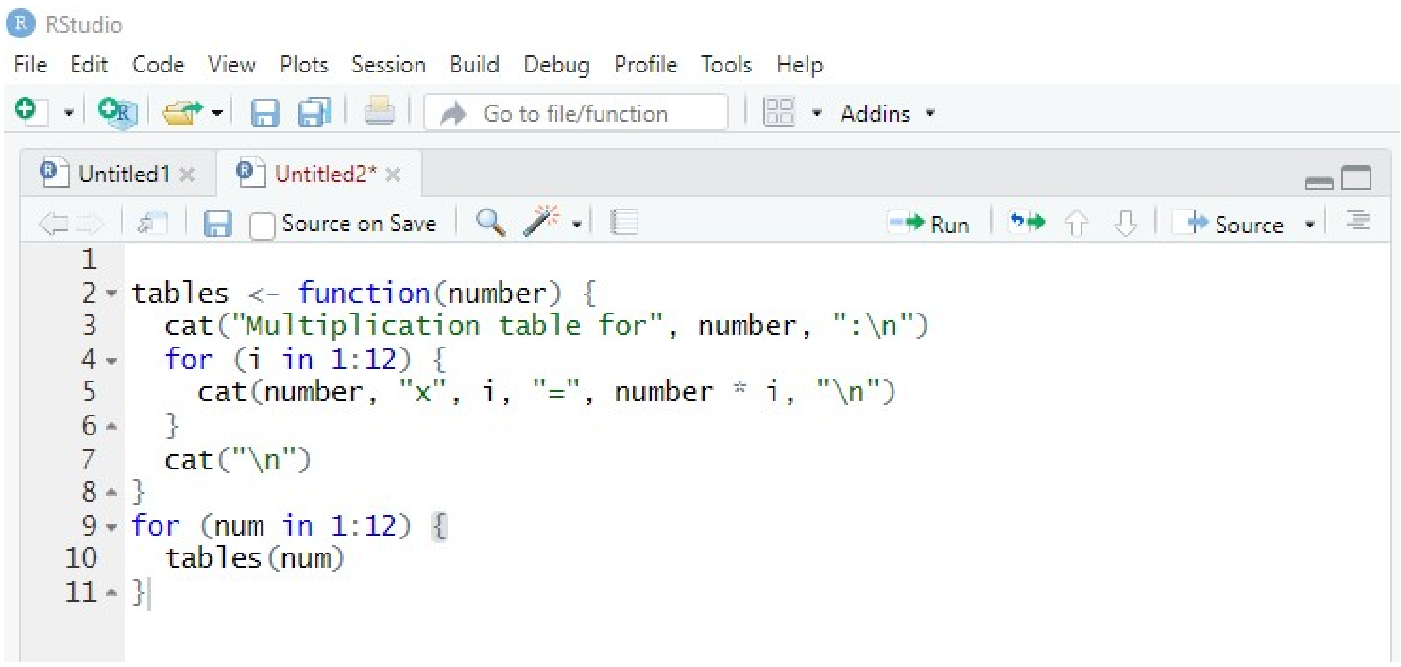
Output >



```
[Workspace loaded from ~/.RData]

>
> # Prompt the user for input
> n<-as.integer(readline("Enter a number (n) :"))
Enter a number (n) :7
> # Calculate the sum
> sum_result <- sum(1:n)
> # Print the result
> cat("The som of integers from 1 to ",n , " is : ", sum_result, "\n")
The som of integers from 1 to  7  is :   28
> 
```

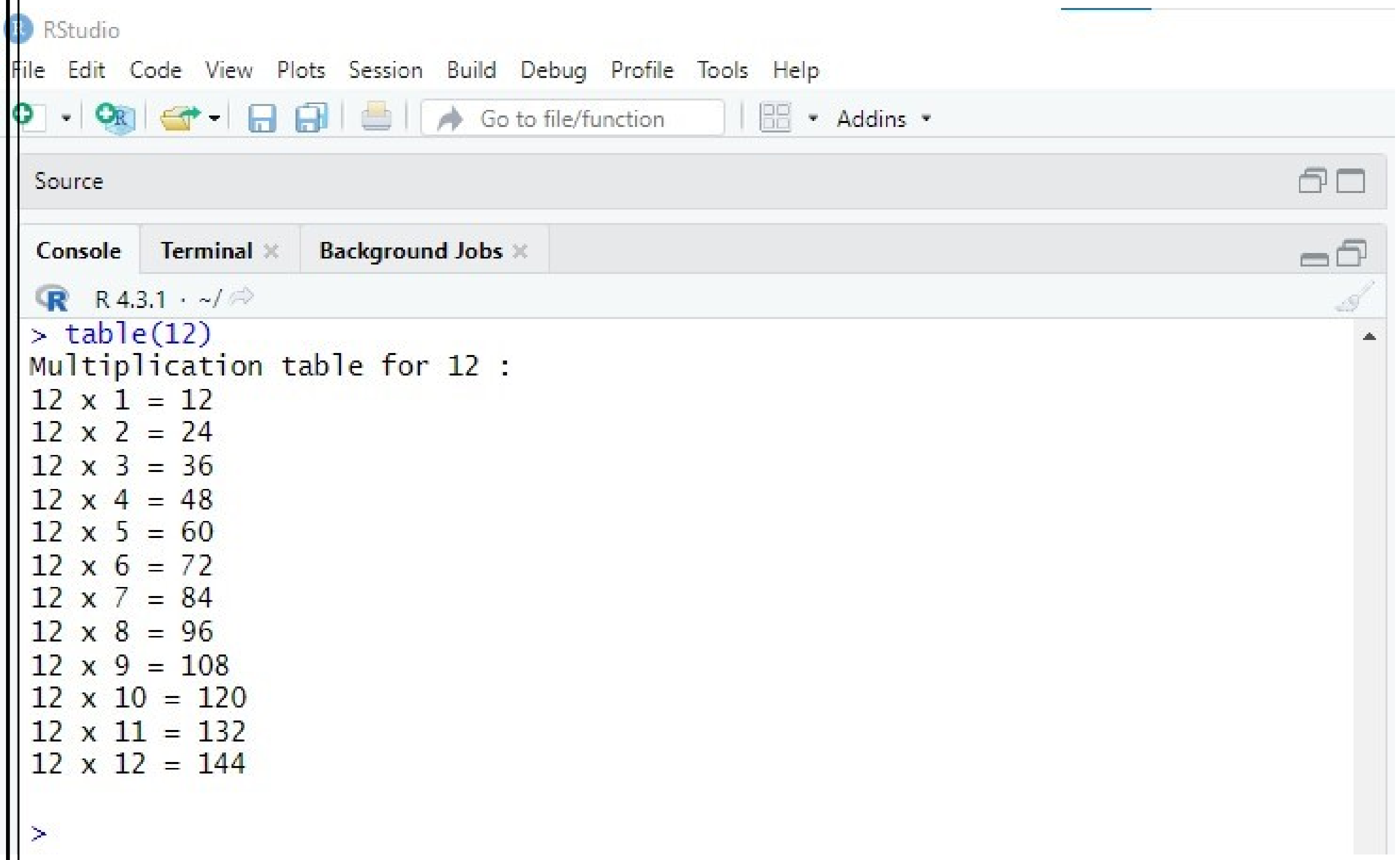
O3. > Write a program that prints a multiplication table for numbers up to 12.

Code >



```r

tables <- function(number) {
  cat("Multiplication table for", number, ":\n")
  for (i in 1:12) {
    cat(number, "x", i, "=", number * i, "\n")
  }
  cat("\n")
}
for (num in 1:12) {
  tables(num)
}
```

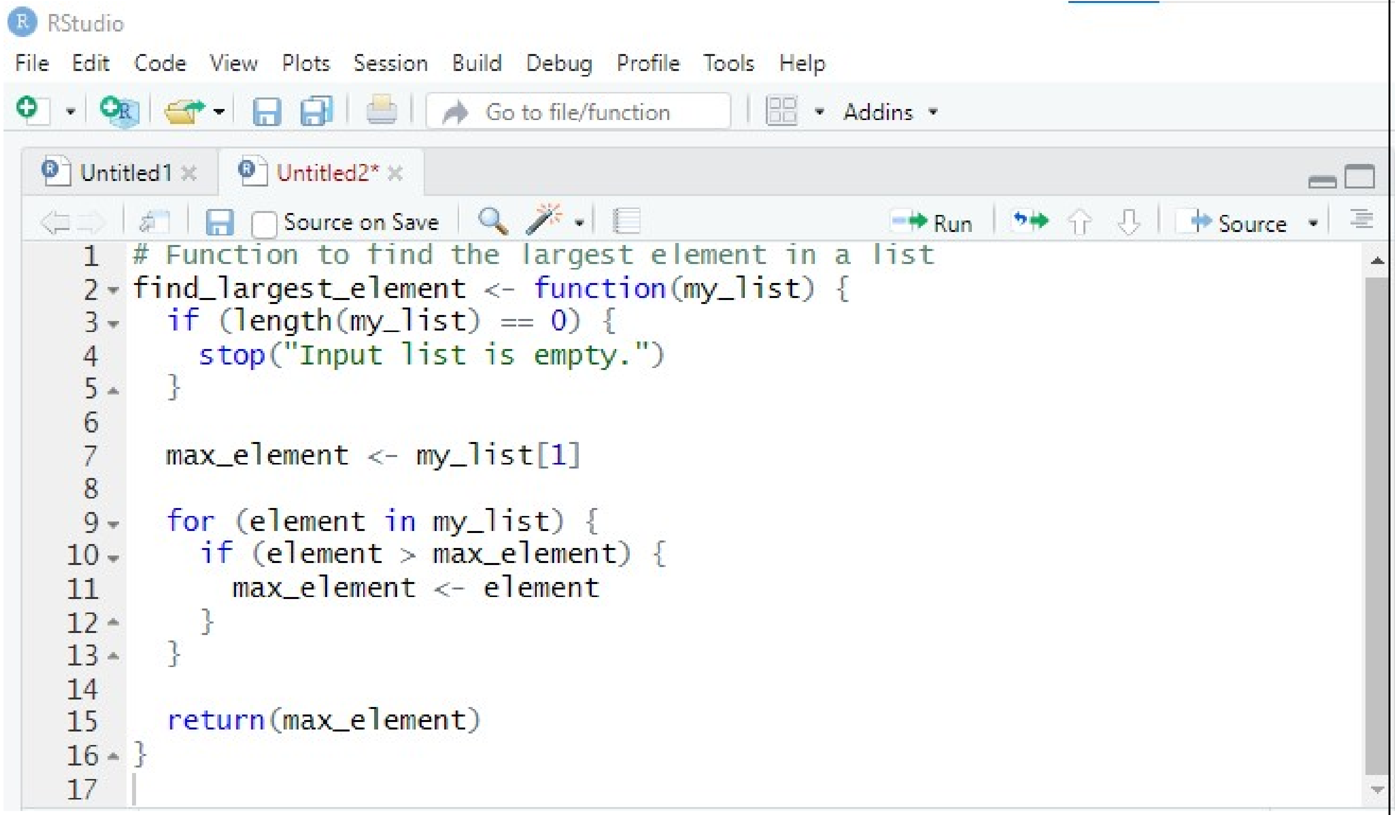
Output >



```
> table(12)
Multiplication table for 12 :
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120
12 x 11 = 132
12 x 12 = 144

>
```

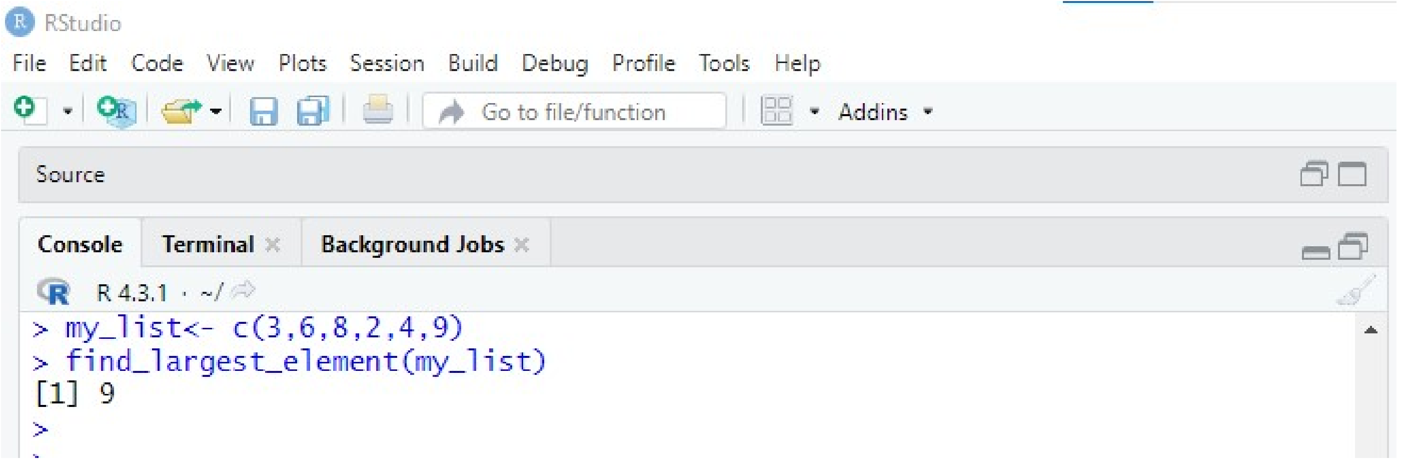
04. > Write a function that returns the largest element in a list.

Code >



```
# Function to find the largest element in a list
find_largest_element <- function(my_list) {
  if (length(my_list) == 0) {
    stop("Input list is empty.")
  }

  max_element <- my_list[1]

  for (element in my_list) {
    if (element > max_element) {
      max_element <- element
    }
  }

  return(max_element)
}
```

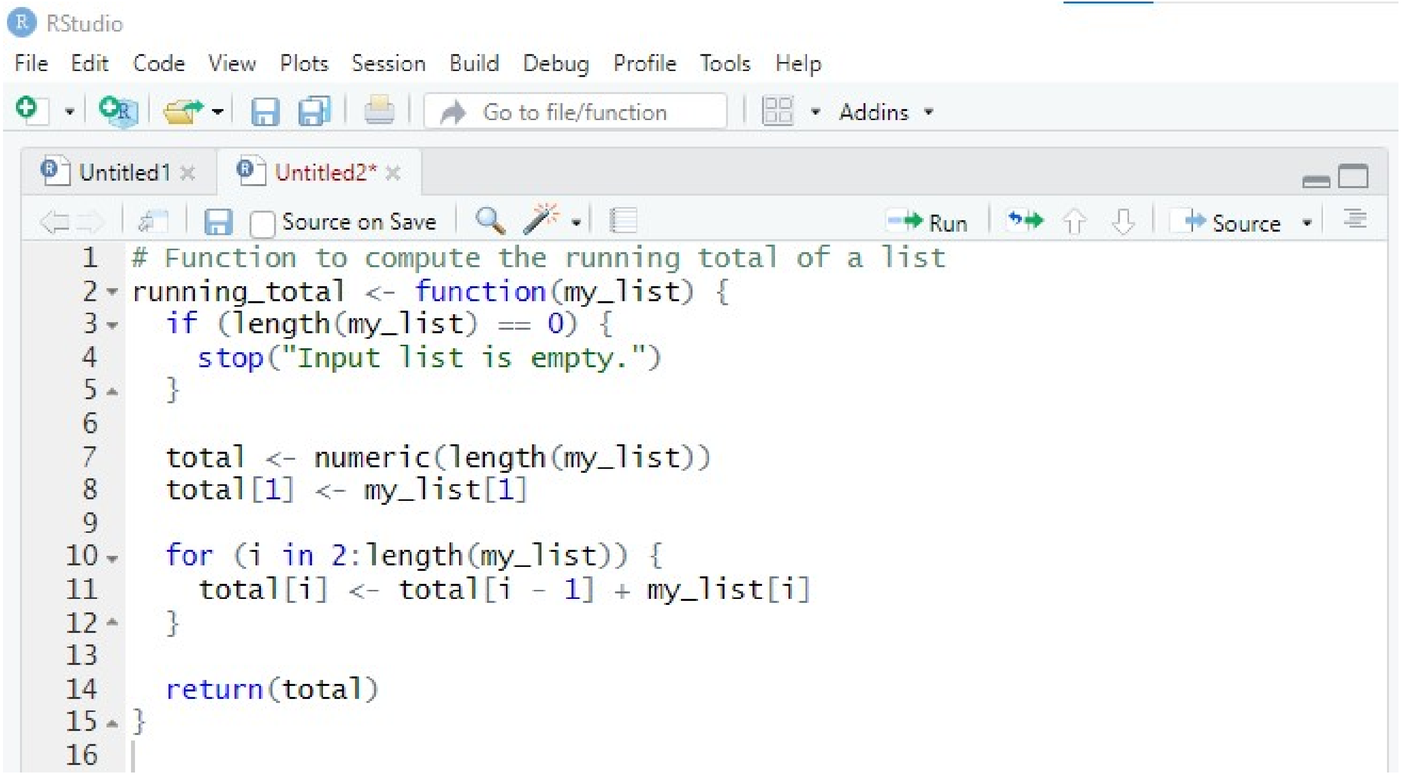
Output >



```
> my_list<- c(3,6,8,2,4,9)
> find_largest_element(my_list)
[1] 9
>
```

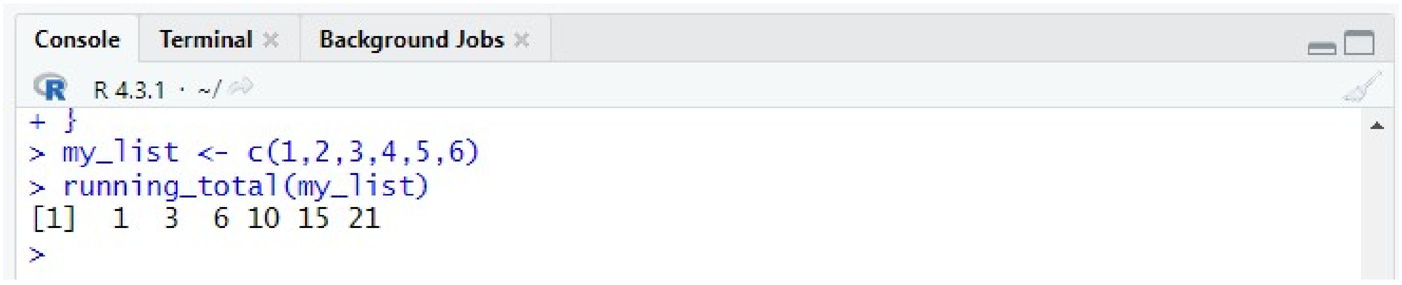
05. > Write a function that computers the running total of a list.

Code >



```r
# Function to compute the running total of a list
running_total <- function(my_list) {
  if (length(my_list) == 0) {
    stop("Input list is empty.")
  }

  total <- numeric(length(my_list))
  total[1] <- my_list[1]

  for (i in 2:length(my_list)) {
    total[i] <- total[i - 1] + my_list[i]
  }

  return(total)
}

```

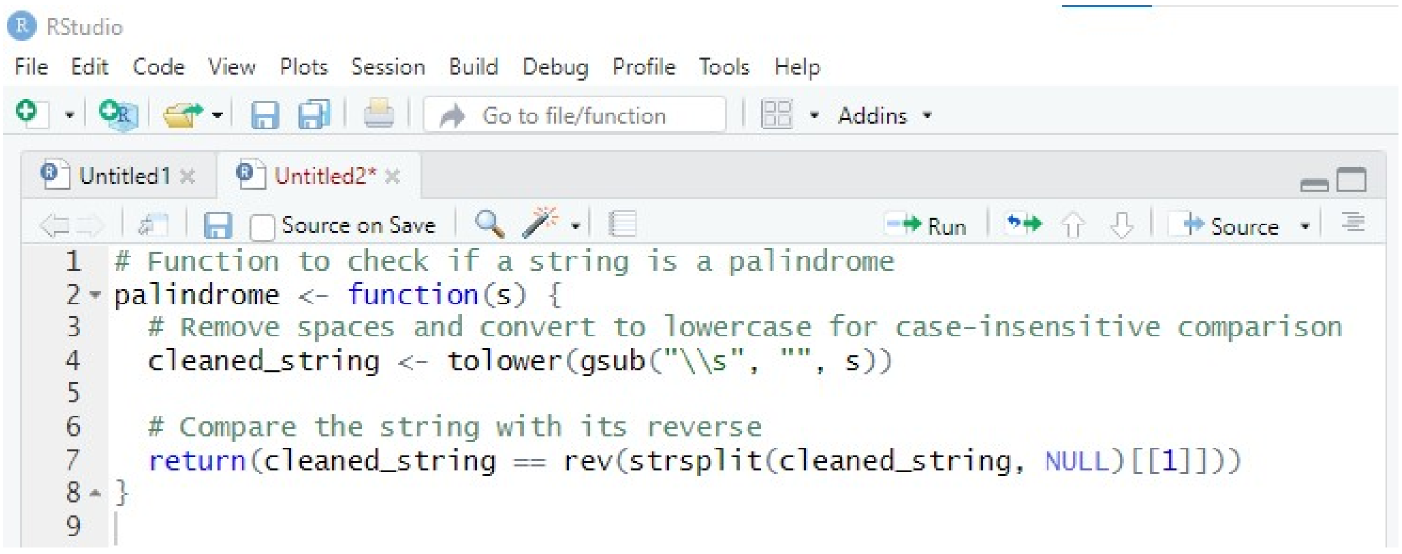
Output >



```
+ }
> my_list <- c(1,2,3,4,5,6)
> running_total(my_list)
[1]  1  3  6 10 15 21
>
```

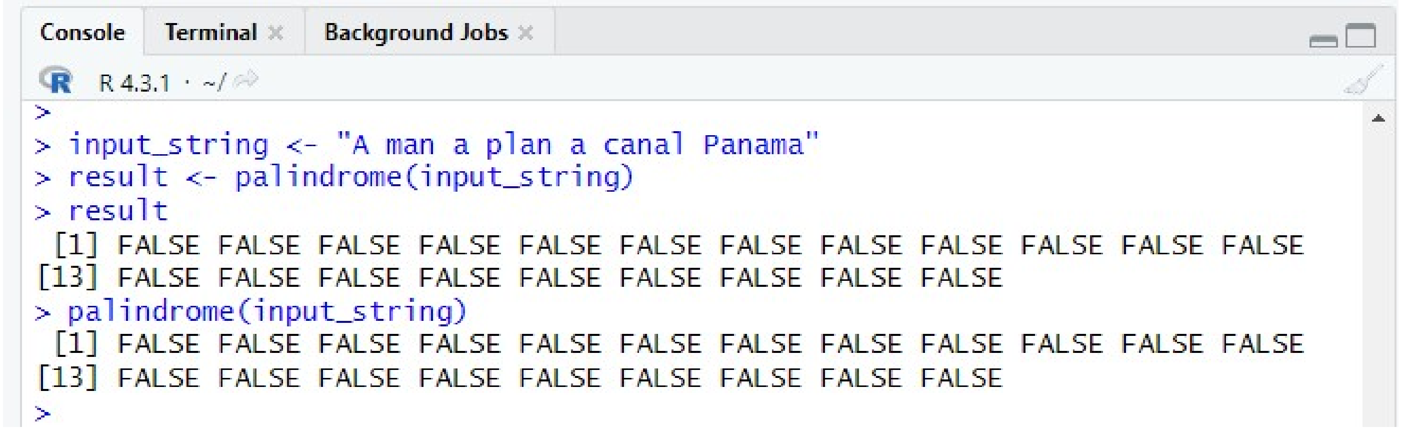
06. > Write a function that tests whether a string is Palindrome.

Code >



```r
# Function to check if a string is a palindrome
palindrome <- function(s) {
  # Remove spaces and convert to lowercase for case-insensitive comparison
  cleaned_string <- tolower(gsub("\\s", "", s))

  # Compare the string with its reverse
  return(cleaned_string == rev(strsplit(cleaned_string, NULL)[[1]]))
}
```

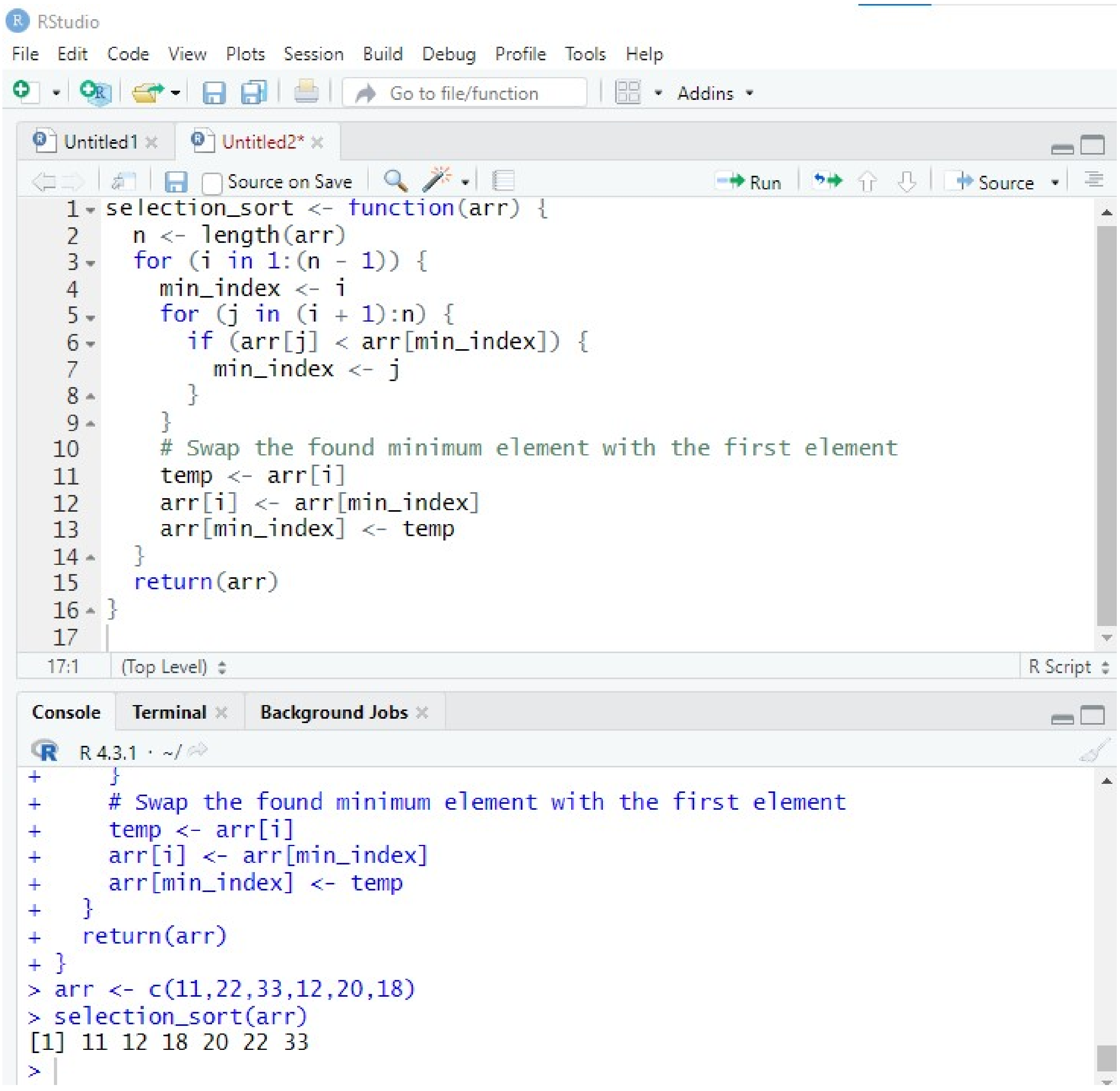
Output >



```
>
> input_string <- "A man a plan a canal Panama"
> result <- palindrome(input_string)
> result
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> palindrome(input_string)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
>
```

07. > Implement the following sorting algorithms: Selection sort, Insertion sort, Bubble sort.

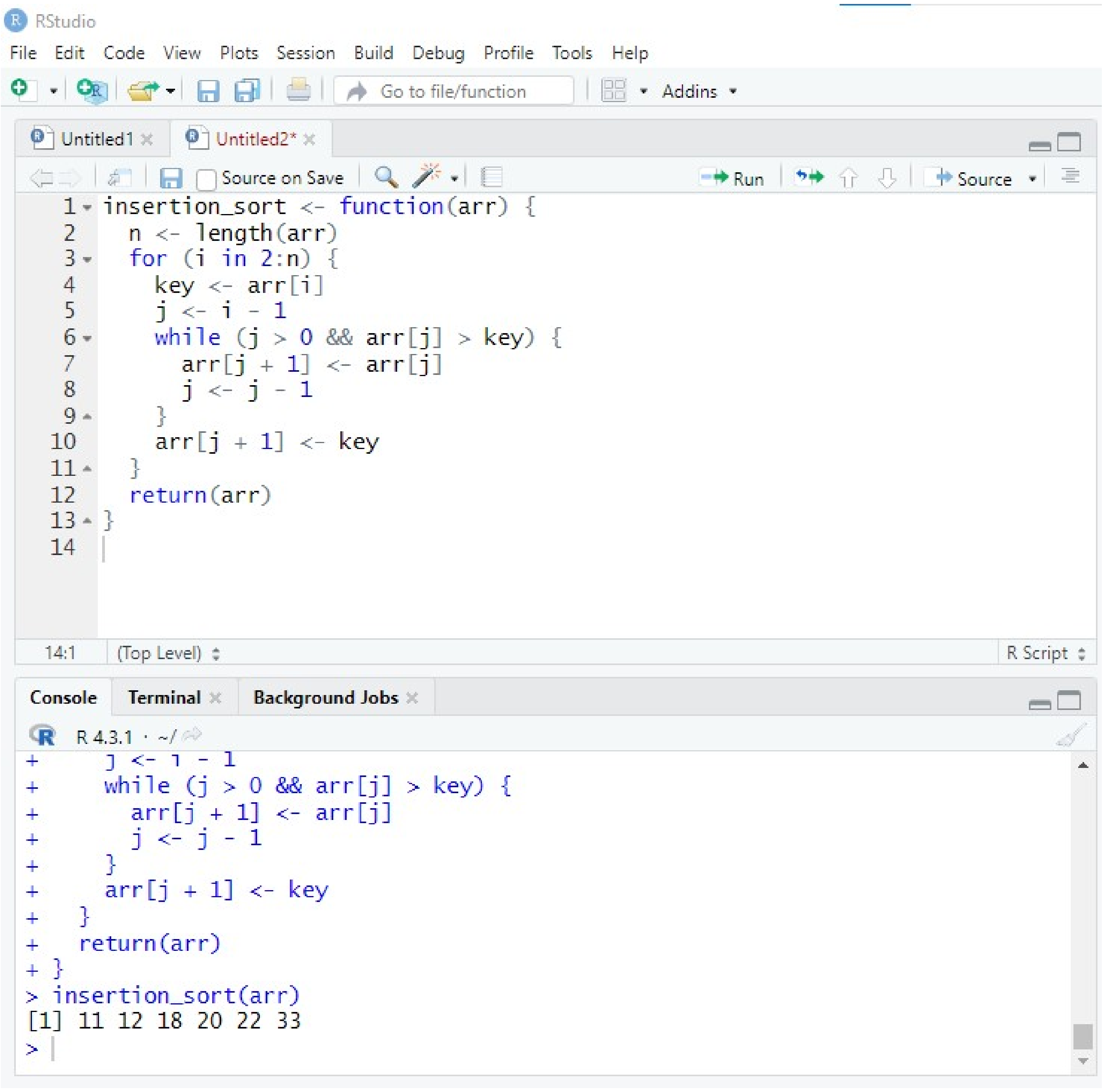
Code > Selection Sort



```r
selection_sort <- function(arr) {
  n <- length(arr)
  for (i in 1:(n - 1)) {
    min_index <- i
    for (j in (i + 1):n) {
      if (arr[j] < arr[min_index]) {
        min_index <- j
      }
    }
    # Swap the found minimum element with the first element
    temp <- arr[i]
    arr[i] <- arr[min_index]
    arr[min_index] <- temp
  }
  return(arr)
}

```

```
+     }
+     # Swap the found minimum element with the first element
+     temp <- arr[i]
+     arr[i] <- arr[min_index]
+     arr[min_index] <- temp
+   }
+   return(arr)
+ }
> arr <- c(11,22,33,12,20,18)
> selection_sort(arr)
[1] 11 12 18 20 22 33
> 
```

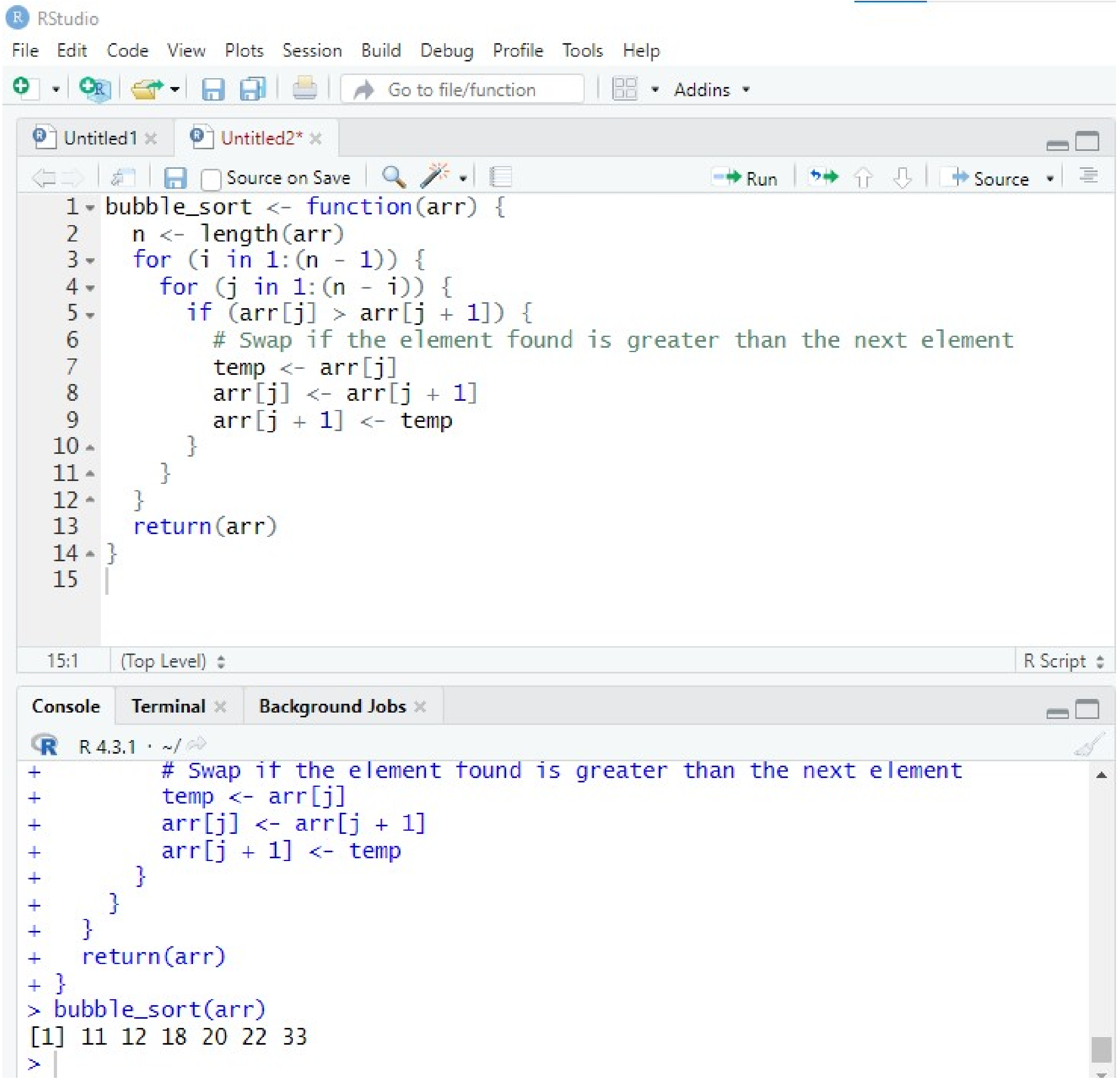
Code > Insertion Sort



```r
insertion_sort <- function(arr) {
  n <- length(arr)
  for (i in 2:n) {
    key <- arr[i]
    j <- i - 1
    while (j > 0 && arr[j] > key) {
      arr[j + 1] <- arr[j]
      j <- j - 1
    }
    arr[j + 1] <- key
  }
  return(arr)
}
```

```
+     j <- i - 1
+     while (j > 0 && arr[j] > key) {
+       arr[j + 1] <- arr[j]
+       j <- j - 1
+     }
+     arr[j + 1] <- key
+   }
+   return(arr)
+ }
> insertion_sort(arr)
[1] 11 12 18 20 22 33
> 
```

Code > Bubble Sort

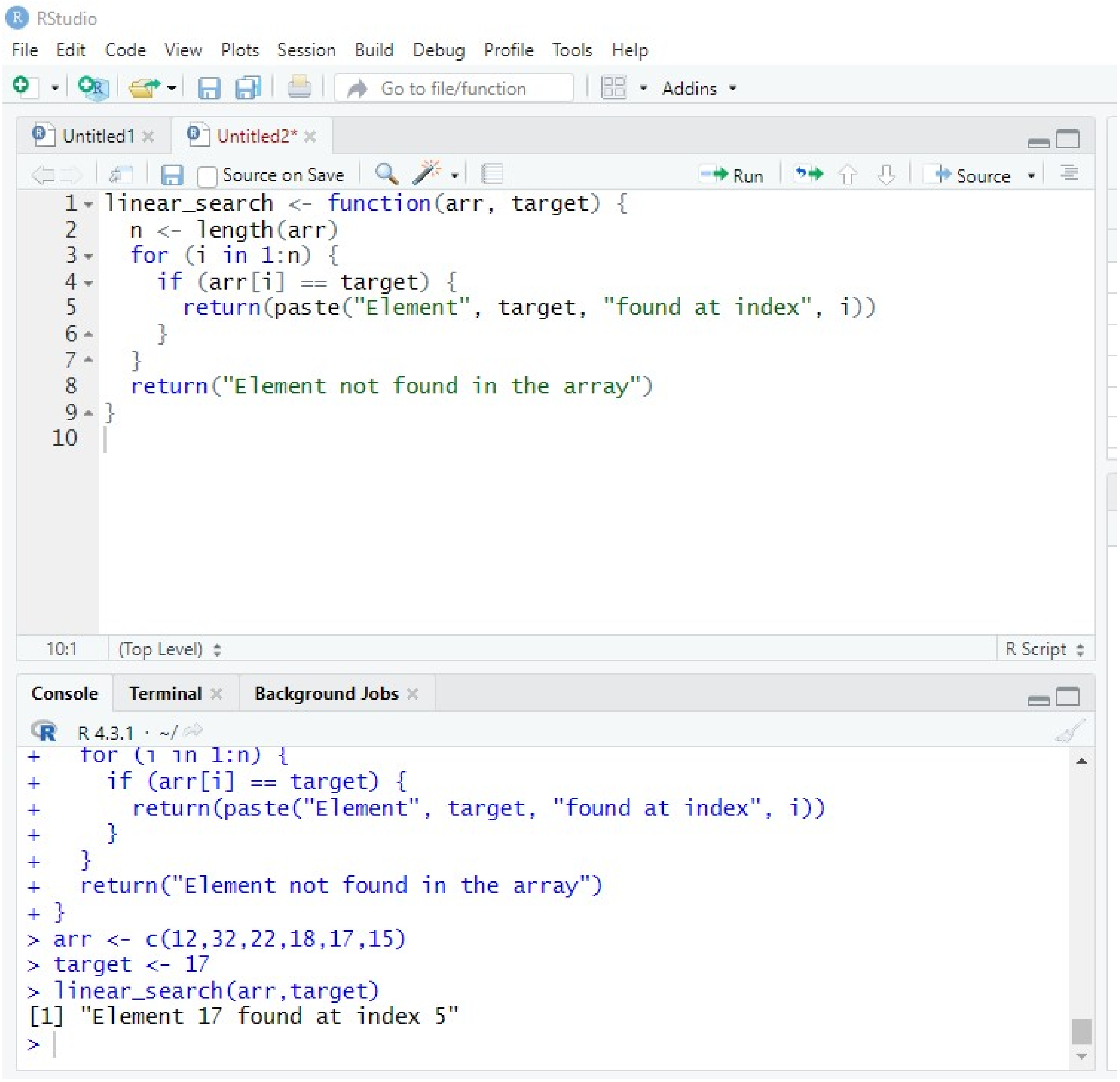


```r
bubble_sort <- function(arr) {
  n <- length(arr)
  for (i in 1:(n - 1)) {
    for (j in 1:(n - i)) {
      if (arr[j] > arr[j + 1]) {
        # Swap if the element found is greater than the next element
        temp <- arr[j]
        arr[j] <- arr[j + 1]
        arr[j + 1] <- temp
      }
    }
  }
  return(arr)
}
```

```
+         # Swap if the element found is greater than the next element
+         temp <- arr[j]
+         arr[j] <- arr[j + 1]
+         arr[j + 1] <- temp
+       }
+     }
+   }
+   return(arr)
+ }
> bubble_sort(arr)
[1] 11 12 18 20 22 33
> 
```

08. > Implement linear search.

Code >

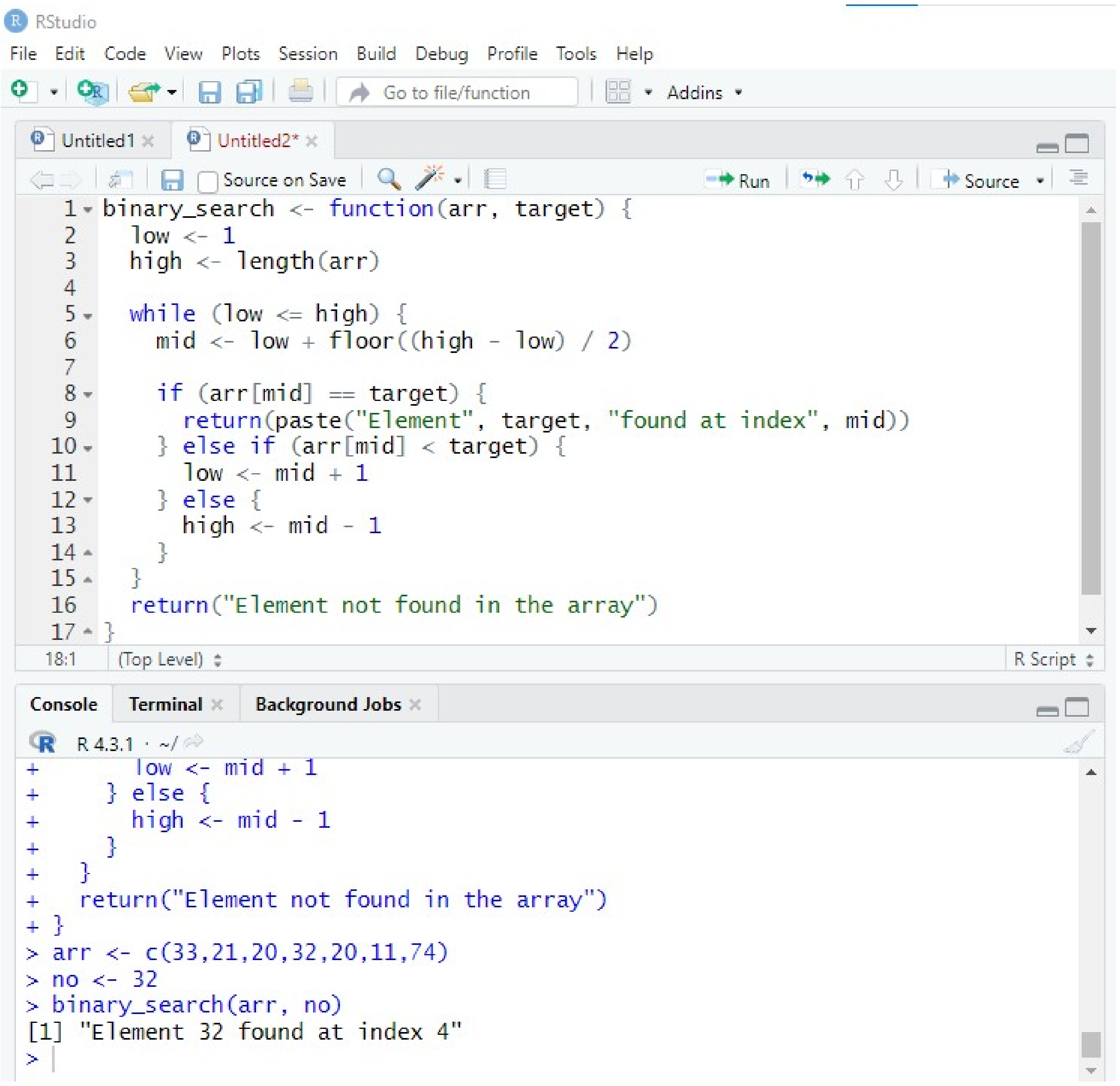


```r
linear_search <- function(arr, target) {
  n <- length(arr)
  for (i in 1:n) {
    if (arr[i] == target) {
      return(paste("Element", target, "found at index", i))
    }
  }
  return("Element not found in the array")
}

```

```
+   for (i in 1:n) {
+     if (arr[i] == target) {
+       return(paste("Element", target, "found at index", i))
+     }
+   }
+   return("Element not found in the array")
+ }
> arr <- c(12,32,22,18,17,15)
> target <- 17
> linear_search(arr,target)
[1] "Element 17 found at index 5"
> 
```

09· > Implement binary search·

Code >

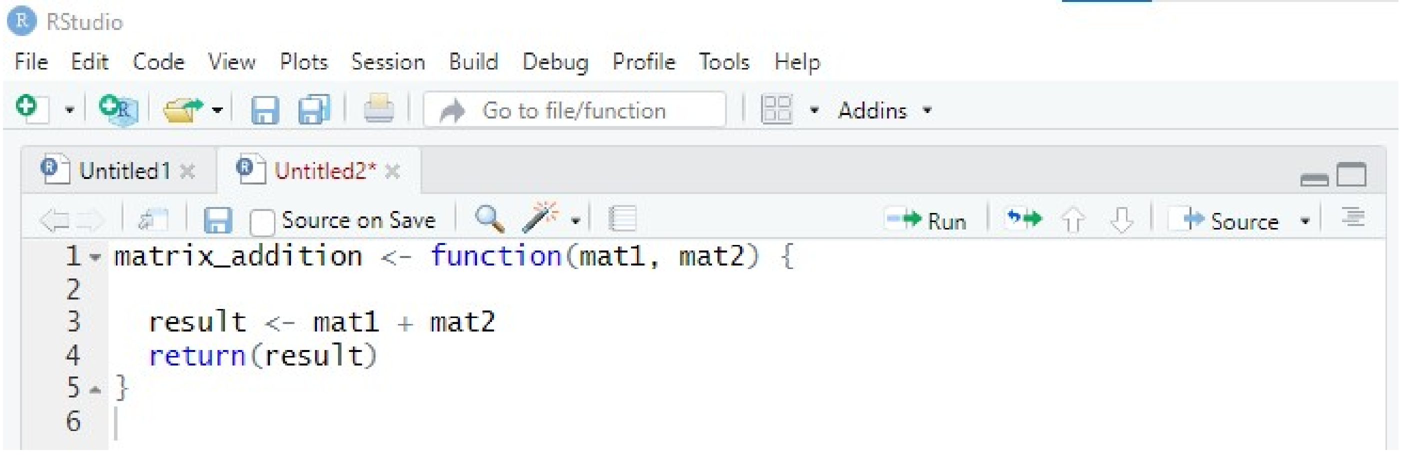


```
binary_search <- function(arr, target) {
  low <- 1
  high <- length(arr)

  while (low <= high) {
    mid <- low + floor((high - low) / 2)

    if (arr[mid] == target) {
      return(paste("Element", target, "found at index", mid))
    } else if (arr[mid] < target) {
      low <- mid + 1
    } else {
      high <- mid - 1
    }
  }
  return("Element not found in the array")
}
```

```
+       low <- mid + 1
+     } else {
+       high <- mid - 1
+     }
+   }
+   return("Element not found in the array")
+ }
> arr <- c(33,21,20,32,20,11,74)
> no <- 32
> binary_search(arr, no)
[1] "Element 32 found at index 4"
> 
```

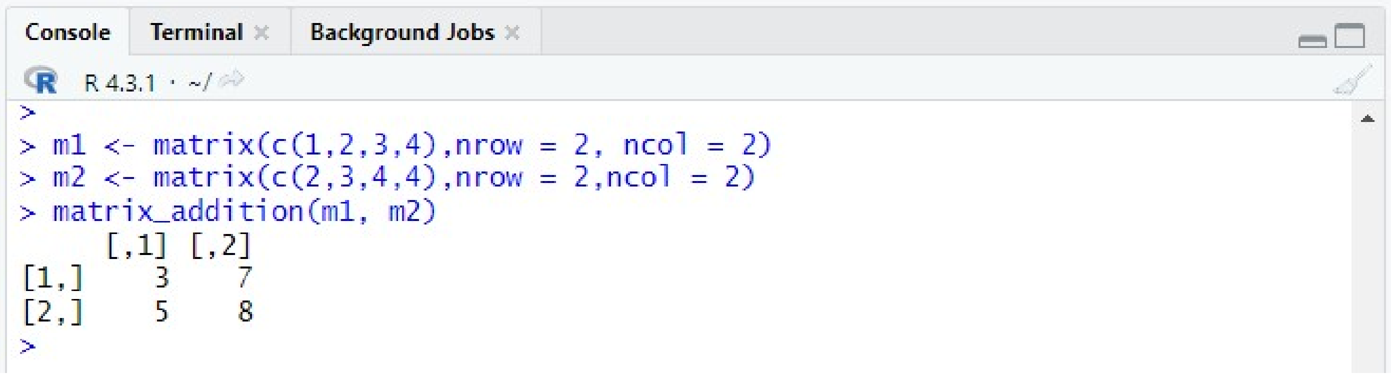
10. > Implement matrices addition, subtraction and Multiplication.

Code > Addition matrices



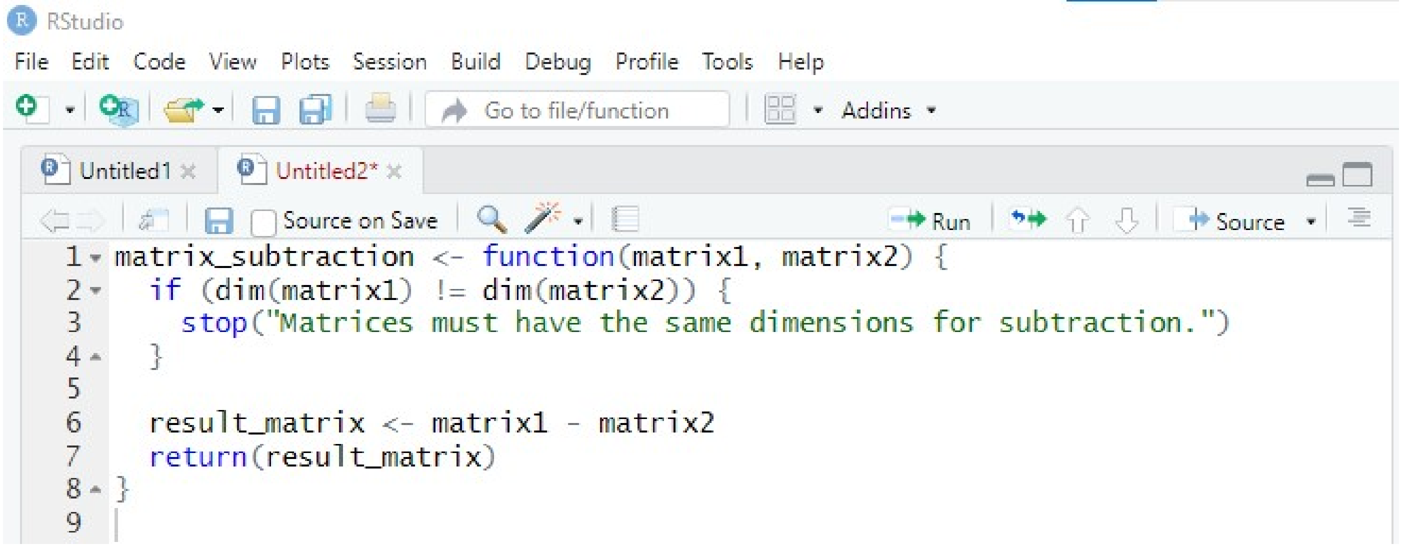
```
matrix_addition <- function(mat1, mat2) {

  result <- mat1 + mat2
  return(result)
}

```

Output >



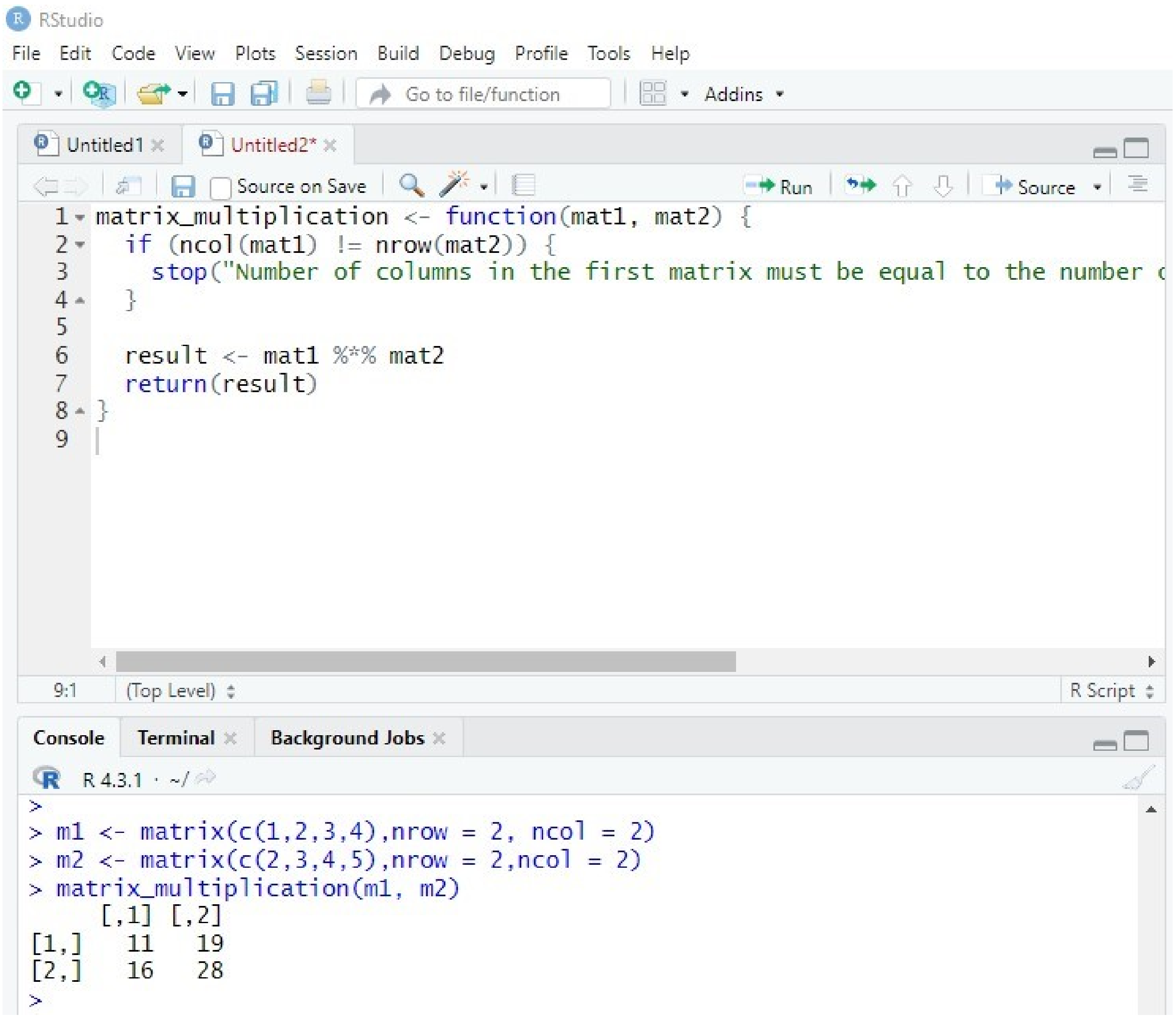
```
R 4.3.1 · ~/
>
> m1 <- matrix(c(1,2,3,4),nrow = 2, ncol = 2)
> m2 <- matrix(c(2,3,4,4),nrow = 2,ncol = 2)
> matrix_addition(m1, m2)
     [,1] [,2]
[1,]    3    7
[2,]    5    8
>
```

Code > Matrix Subtraction



```
matrix_subtraction <- function(matrix1, matrix2) {
  if (dim(matrix1) != dim(matrix2)) {
    stop("Matrices must have the same dimensions for subtraction.")
  }

  result_matrix <- matrix1 - matrix2
  return(result_matrix)
}

```

Code > Matrix Multiplication



```r
matrix_multiplication <- function(mat1, mat2) {
  if (ncol(mat1) != nrow(mat2)) {
    stop("Number of columns in the first matrix must be equal to the number c
  }

  result <- mat1 %*% mat2
  return(result)
}

```

```
>
> m1 <- matrix(c(1,2,3,4),nrow = 2, ncol = 2)
> m2 <- matrix(c(2,3,4,5),nrow = 2,ncol = 2)
> matrix_multiplication(m1, m2)
     [,1] [,2]
[1,]   11   19
[2,]   16   28
>
```