Practical – 05

Title: - Introduction to R programming and Data acquisition

Aim: - To understand basic concepts in R programming.

Lab Objectives: -

Students will understand following R programming concepts:

I. What is R?

II. Installation of R

III. R syntax, comments, variables

IV. R Datatypes

V. R Functions

VI. R Vectors, Factors, Dataframes

VII. Installing and loading packages

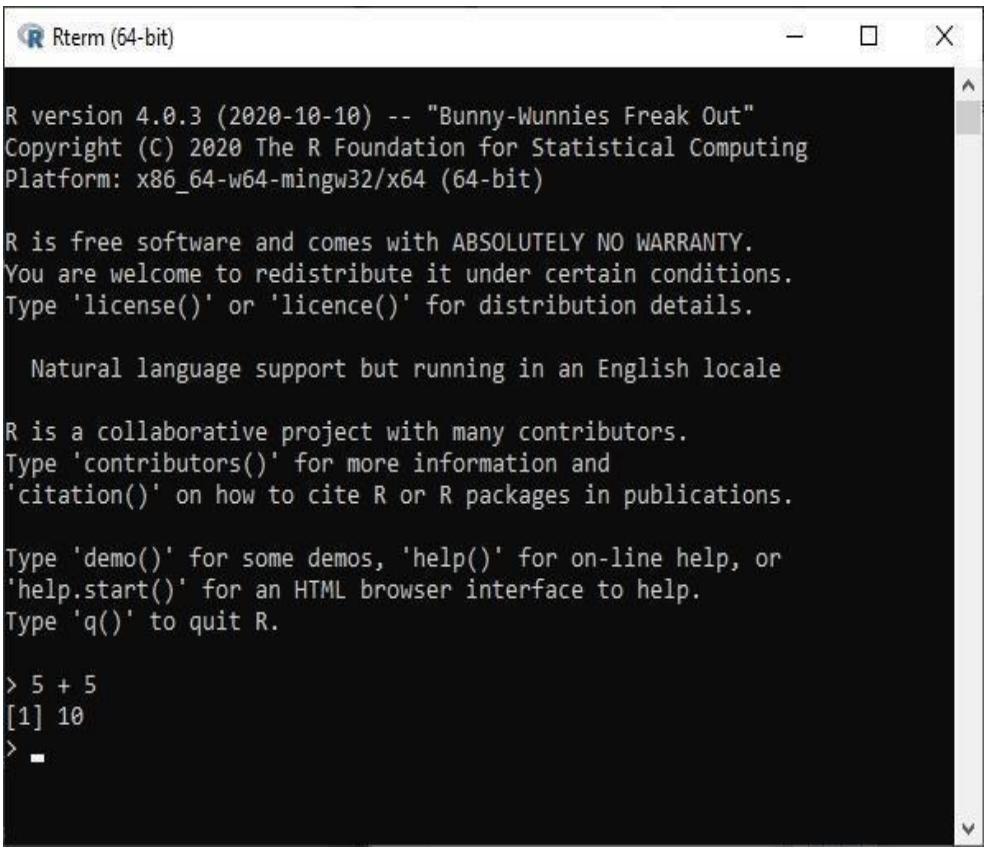
Description:

I. What is R?

- R is a popular programming language used for statistical computing and graphical presentation.
- Its most common use is to analyze and visualize data.
- It is a great resource for data analysis, data visualization, data science and machine learning
- It provides many statistical techniques (such as statistical tests, classification, clustering and data reduction)
- It is easy to draw graphs in R, like pie charts, histograms, box plot, scatter plot, etc++
- It works on different platforms (Windows, Mac, Linux)
- It is open-source and free
- It has a large community support
- It has many packages (libraries of functions) that can be used to solve different problems

II. Installation of R

- To install R, go to https://cloud.r-project.org/ and download the latest version of R for Windows, Mac or Linux.
- When you have downloaded and installed R, you can run R on your computer.
- The screenshot below shows how it may look like when you run R on a Windows PC:



III. R syntax, comments, variables

Syntax

- To output text in R, use single or double quotes:
  Example
  "Hello World!"
- To output numbers, just type the number (without quotes):
  Example
  5
  10
  25
- To do simple calculations, add numbers together:
  Example

```
5 + 5
```

Print

- R does have a print() function available if you want to use it. This might be useful if you are familiar with other programming languages, such as Python, which often uses the print() function to output code. Example
  print("Hello World!")
- And there are times you must use the print() function to output code, for example when working with for loops (which you will learn more about in a later chapter):
  Example for (x in 1:10) { print(x)
  }

Comments

- Comments can be used to explain R code, and to make it more readable. It can also be used to prevent execution when testing alternative code.
- Comments starts with a #. When executing the R-code, R will ignore anything that starts with #.
- This example uses a comment before a line of code:

```
Example
# This is a comment
"Hello World!"
```

▷ **Multiline Comments**

- ○ Unlike other programming languages, such as Java, there are no syntax in R for multiline comments.
- ○ However, we can just insert a # for each line to create multiline comments:
- ○ Example

```
# This is a comment
# written in
# more than just one line
"Hello World!"
```

Creating Variables in R

- Variables are containers for storing data values.
- R does not have a command for declaring a variable.
- A variable is created the moment you first assign a value to it. To assign a value to a variable, use the <- sign.
- To output (or print) the variable value, just type the variable name:
- Example name <- "John" age <- 40 name # output "John" age # output 40

▷ **Concatenate Elements**

- You can also concatenate, or join, two or more elements, by using the paste() function.
- To combine both text and a variable, R uses comma (,):
- Example

```
text <- "awesome"
paste("R is", text)
```

- You can also use , to add a variable to another variable:
- Example

```
text1 <- "R is"
text2 <-
"awesome"
paste(text1, text2)
```

- For numbers, the + character works as a mathematical operator:
  Example num1

```
<- 5 num2 <-
10 num1 +
num2
```

- If you try to combine a string (text) and a number, R will give you an error:
  Example

```
num <-
5
text <- "Some text"
num + text
```

▷ **Multiple Variables**

- R allows you to assign the same value to multiple variables in one line:
- Example

```
# Assign the same value to multiple variables in one
line var1 <- var2 <- var3 <- "Orange" # Print variable
values var1 var2 var3
```

▷ **Variable Names**

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for R variables are:
  - A variable name must start with a letter and can be a combination of letters, digits, period(.) and underscore(_). If it starts with period(.), it cannot be followed by a digit.
  - A variable name cannot start with a number or underscore (_)
  - Variable names are case-sensitive (age, Age and AGE are three different variables)
  - Reserved words cannot be used as variables (TRUE, FALSE, NULL, if...)

IV. Data Types

- In programming, data type is an important concept.
- Variables can store data of different types, and different types can do different things.
- In R, variables do not need to be declared with any particular type, and can even change type after they have been set:
- Example my_var <- 30 # my_var is type of numeric
  my_var <- "Sally" # my_var is now of type character (aka string)

○ **Basic Data Types**

- Basic data types in R can be divided into the following types:
  - numeric - (10.5, 55, 787)
  - integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
  - complex - (9 + 3i, where "i" is the imaginary part)
  - character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
  - logical (a.k.a. boolean) - (TRUE or FALSE)
- We can use the class() function to check the data type of a variable:
  # numeric
  x <- 10.5
  class(x)

▷ **Type Conversion**

- You can convert from one type to another with the following functions:
  - as.numeric()
  - as.integer()
  - as.complex()

▷ **Built-in Math and String Functions** ○ **min() and max()**

- R also has many built-in math functions that allows you to perform mathematical tasks on numbers.
- For example, the min() and max() functions can be used to find the lowest or highest number in a set:
- Example
  max(5, 10, 15)
  min(5, 10, 15)

○ **sqrt()**

- The sqrt() function returns the square root of a number:
- Example
  sqrt(16)

- **abs()**
  - The abs() function returns the absolute (positive) value of a number:
  - Example
    abs(-4.7)
- **ceiling() and floor()**
  - The ceiling() function rounds a number upwards to its nearest integer, and the floor() function rounds a number downwards to its nearest integer, and returns the result:
  - Example

  ceiling(1.4)
  floor(1.4)
- **String Length**
  - For example, to find the number of characters in a string, use the nchar() function:
  - Example
    str <- "Hello World!"
    nchar(str)
- **Check a String**
  - Use the grepl() function to check if a character or a sequence of characters are present in a string:
  - Example
    str <- "Hello World!"
    grepl("H", str)
    grepl("Hello", str)
    grepl("X", str)
- **Concatenating Strings - paste() function**
  - Many strings in R are combined using the paste() function. It can take any number of arguments to be combined together.
  - Syntax
  - The basic syntax for paste function is −
  - paste(..., sep = " ", collapse = NULL)
  - Following is the description of the parameters used − ▪ ...

    represents any number of arguments to be combined.

    ▪ sep represents any separator between the arguments. It is optional.

    ▪ collapse is used to eliminate the space in between two strings. But not the space within two words of one string.
- **Changing the case - toupper() & tolower() functions**
  - These functions change the case of characters of a string.
  - Syntax

- The basic syntax for toupper() & tolower() function is −
  toupper(x)
  tolower(x)
- Following is the description of the parameters used −
  x is the vector input.

○ **Extracting parts of a string - substring()**

function ■ This function extracts parts of a String.

- Syntax
- The basic syntax for substring() function is −
  substring(x,first,last)
- Following is the description of the parameters used −
  x is the character vector input.
- first is the position of the first character to be extracted.
- last is the position of the last character to be extracted.

Example

```
# Extract characters from 5th to 7th position.
result <- substring("Extract", 5, 7)
print(result)
```

V. R Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

▷ **Creating a Function**

○ To create a function, use the function() keyword:

○ Example

```
my_function <- function() { # create a function with the name my_function
print("Hello World!")
}
```

▷ **Call a Function**

○ To call a function, use the function name followed by parenthesis, like my_function():

○ Example

```
my_function <- function() {
  print("Hello World!")
}
my_function() # call the function named my_function
```

▷ **Arguments**

○ Information can be passed into functions as arguments.

- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:
- Example

```
my_function <- function(fname) {
  paste(fname, "Griffin")
}
my_function("Peter")
my_function("Lois")
my_function("Stewie")
```

▷ **Default Parameter Value**

- The following example shows how to use a default parameter value.
- If we call the function without an argument, it uses the default value:
- Example

```
my_function <- function(country = "Norway") {
  paste("I am from", country)
}
my_function("Sweden") my_function("India")
my_function() # will get the default value, which is
Norway my_function("USA")
```

▷ **Return Values**

- To let a function return a result, use the return() function:
- Example

```
my_function <- function(x) {
  return (5 * x)
}
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

VI. R Vectors, Factors, Dataframes

- A vector is simply a list of items that are of the same type.
- To combine the list of items to a vector, use the c() function and separate the items by a comma.
- In the example below, we create a vector variable called fruits, that combine strings:
- Example

```
# Vector of strings
fruits <- c("banana", "apple", "orange")
```

```
# Print fruits
fruits
```

⩥ Using sequence (Seq.) operator

```
# Create vector with elements from 5 to 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```

▷ When we execute the above code, it produces the following result −

```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

▷ Example numbers <- seq(from = 0, to =
100, by = 20) numbers

⩥ Using the c() function

- The non-character values are coerced to character type if one of the elements is a character.

```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5,TRUE)
print(s)
```

- When we execute the above code, it produces the following result −

```
[1] "apple" "red"   "5"   "TRUE"
```

- To create a vector with numerical values in a sequence, use the : operator
- Example

```
# Vector with numerical values in a
sequence numbers <- 1:10 numbers
```

- You can also create numerical values with decimals in a sequence, but note that if the last element does not belong to the sequence, it is not used:
- Example

```
# Vector with numerical decimals in a
sequence numbers1 <- 1.5:6.5 numbers1
# Vector with numerical decimals in a sequence where the last element is not
used numbers2 <- 1.5:6.3 numbers2
```

▷ **Vector Length**

- To find out how many items a vector has, use the length() function:
- Example

```
fruits <- c("banana", "apple", "orange")
length(fruits)
```

▷ **Sort a Vector**

- To sort items in a vector alphabetically or numerically, use the sort() function:
- Example

```
fruits <- c("banana", "apple", "orange", "mango",
"lemon") numbers <- c(13, 3, 5, 7, 20, 2) sort(fruits) #
Sort a string sort(numbers) # Sort numbers
```

▷ **Access Vectors**

- You can access the vector items by referring to its index number inside brackets []. The first item has index 1, the second item has index 2, and so on:
- Example

```
fruits <- c("banana", "apple", "orange")
# Access the first item (banana)
fruits[1]
```

- You can also access multiple elements by referring to different index positions with the c() function:
- Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")
# Access the first and third item (banana and orange)
fruits[c(1, 3)]
```

▷ **Change an Item**

- To change the value of a specific item, refer to the index number:
- Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")
        # Change "banana" to
        "pear" fruits[1] <- "pear" #
        Print fruits fruits
```

▷ **Repeat Vectors**

- To repeat vectors, use the rep() function:
- Example
- Repeat each value: repeat_each <- rep(c(1,2,3), each = 3) repeat_each
- Example
- Repeat the sequence of the vector: repeat_times <- rep(c(1,2,3), times = 3) repeat_times

▷ **Vector arithmetic**

- Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
        v1 <-
        c(3,8,4,5,0,11) v2
        <- c(4,11,0,8,1,2)
        # Vector addition.
        result <- v1+v2
```

```
print(result) result
<- v1-v2
print(result) result
<- v1*v2
print(result) result
<- v1/v2
print(result)
```

R Factors

- Factors are the data objects which are used to categorize the data and store it as levels.
- They can store both strings and integers.
- Examples of factors are:
  - Demography: Male/Female
  - Music: Rock, Pop, Classic, Jazz
  - Training: Strength, Stamina
- To create a factor, use the factor() function and add a vector as argument:
- Example

```
# Create a factor music_genre <- factor(c("Jazz", "Rock", "Classic",
"Classic", "Pop", "Jazz", "Rock", "Jazz"))
# Print the factor
music_genre
```

- To only print the levels, use the levels() function:
- Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz",
"Rock",
"Jazz"))
levels(music_genre)
```

- You can also set the levels, by adding the levels argument inside the factor() function:
- Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz",
"Rock",
"Jazz"), levels = c("Classic", "Jazz", "Pop", "Rock", "Other"))
levels(music_genre)
```

▷ **Factor Length**

- Use the length() function to find out how many items there are in the factor:
- Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz",
"Rock",
"Jazz"))
length(music_genre)
```

▷ **Access Factors**

- To access the items in a factor, refer to the index number, using [] brackets:
- Example - Access the third item: music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))
  music_genre[3]

R Data Frames

- Data Frames are data displayed in a format as a table.
- Each column contains values of one variable and each row contains one set of values from each column.
- Data Frames can have different types of data inside it.
- Following are the characteristics of a data frame.
  - The column names should be non-empty.
  - The row names should be unique.
  - The data stored in a data frame can be of numeric, factor or character type.
  - Each column should contain same number of data items.
- Use the data.frame() function to create a data frame:
- Example

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Print the data frame
Data_Frame
```

▷ **Summarize the Data**

- Use the summary() function to summarize the data from a Data Frame:
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data_Frame
summary(Data_Frame)
```

▷ **Access Items**

- We can use single brackets [ ], double brackets [[ ]] or $ to access columns from a data frame:
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data_Frame[1]
Data_Frame[["Training"]]
Data_Frame$Training
```

▷ **Add Rows**

- Use the rbind() function to add new rows in a Data Frame:
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Add a new row
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))
# Print the new row
New_row_DF
```

▷ **Add Columns**

- Use the cbind() function to add new columns in a Data Frame:
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Add a new column
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000, 2000))
# Print the new column
New_col_DF
```

▷ **Remove Rows and Columns**

- Use the c() function to remove rows and columns in a Data Frame:

- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Remove the first row and column
Data_Frame_New <- Data_Frame[-c(1), -c(1)]
# Print the new data frame
Data_Frame_New
```

▷ **Amount of Rows and Columns**

- Use the dim() function to find the amount of rows and columns in a Data Frame:
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
dim(Data_Frame)
```

- You can also use the ncol() function to find the number of columns and nrow() to find the number of rows:
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
ncol(Data_Frame)
nrow(Data_Frame)
```

▷ **Get the Structure of the Data Frame**

- The structure of the data frame can be seen by using str() function.
- Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
str(Data_Frame)
```

R – Packages

- R packages are a collection of R functions, complied code and sample data.
- They are stored under a directory called "library" in the R environment.
- By default, R installs a set of packages during installation.
- More packages are added later, when they are needed for some specific purpose.
- When we start the R console, only the default packages are available by default.
- Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.

⮚ Check Available R Packages

- Get library locations containing R packages
  .libPaths()
- When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.
- [2] "C:/Program Files/R/R-3.2.2/library“

⮚ Get the list of all the packages installed

library()

⮚ Get all packages currently loaded in the R environment

search()

▷ What Are Repositories?

- A repository is a place where packages are located so you can install them from it.
- Three of the most popular repositories for R packages are:
  ○ CRAN:
  ○ Bioconductor: ○
  Github :

▷ How To Install An R Package

- The most common way is to use the CRAN repository, then you just need the name of the package and use the command install.packages("package")

- An example is given below for the ggplot2 package that will be required for some plots we will create later on. Run this code to install ggplot2.
  install.packages("ggplot2")
- Package installation from source
- Finally, R packages can also be installed from source. This is useful when you do not have an internet connection (and have the source files locally)
- To install from source, we use the same install.packages function but we have additional arguments that provide specifications to change from defaults:
  install.packages("~/Downloads/ggplot2_1.0.1.tar.gz", type="source", repos=NULL)

▷ **Loading libraries**

- Once you have the package installed, you can load the library into your R session for use.
- Any of the functions that are specific to that package will be available for you to use by simply calling the function as you would for any of the base functions.
- Note that quotations are not required here.

```
library(ggplot2)
```

- A package might have fifteen functions and the other might have hundred, it totally depends on the necessity.
- We can find the functions inside a package by using lsf.str function but we need to load the package prior to knowing the functions inside.

```
ls("package:ggplot2")
```

Roll no. : 40 Name : Anish Ramakant Karlekar

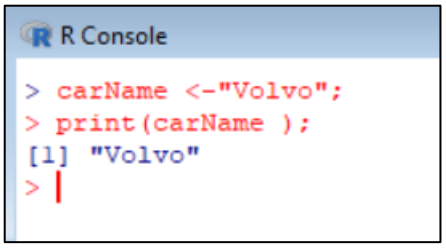
Exercise:

1. **Create a variable named carName and assign the value Volvo to it.**

   CODE :

   ```
   carName <- “Volvo”
   print(carName)
   ```

   OUTPUT :

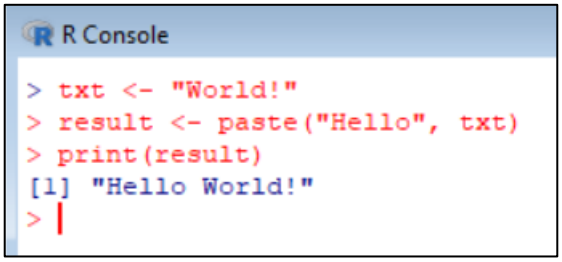
   

2. **Use the correct function to combine the text "Hello" with the txt variable, to output "Hello World!".**

   CODE:

   ```
   txt <- "World!"
   result <- paste("Hello", txt)
   print(result)
   ```

   OUTPUT :

3. **What data type is myVar and x? x <- 10.5 myVar <- 30**

CODE :

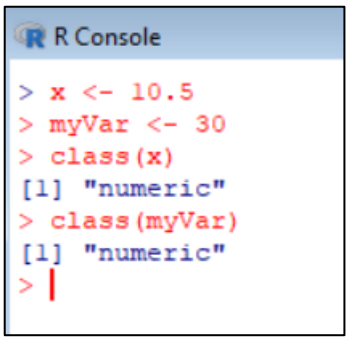
```
x <- 10.5

myVar <- 30

class(x)

class(myVar)
```

OUTPUT :

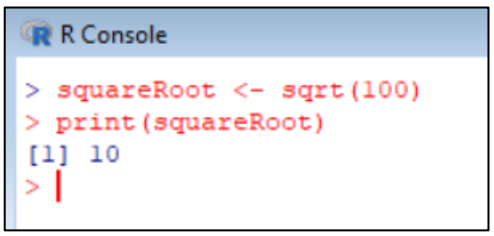


4. **Use the correct function to find the square root of the number 100.**

CODE :

```
squareRoot <- sqrt(100)
print(squareRoot)
```

OUTPUT :

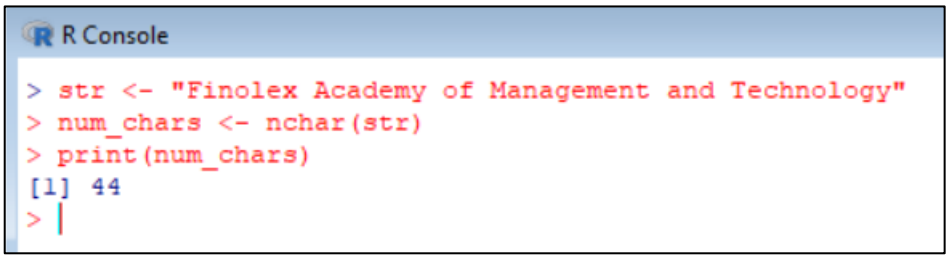


5. **Use the correct function to find the number of characters in the str variable: str<- "Finolex Academy of Management and Technology"**

   CODE:

```
str <- "Finolex Academy of Management and Technology"
num_chars <- nchar(str)
print(num_chars)
```

   OUTPUT :



6. **Write a R program to create a sequence of numbers from 20 to 50 and find the mean of numbers from 20 to 60 and sum of numbers from 51 to 91.**

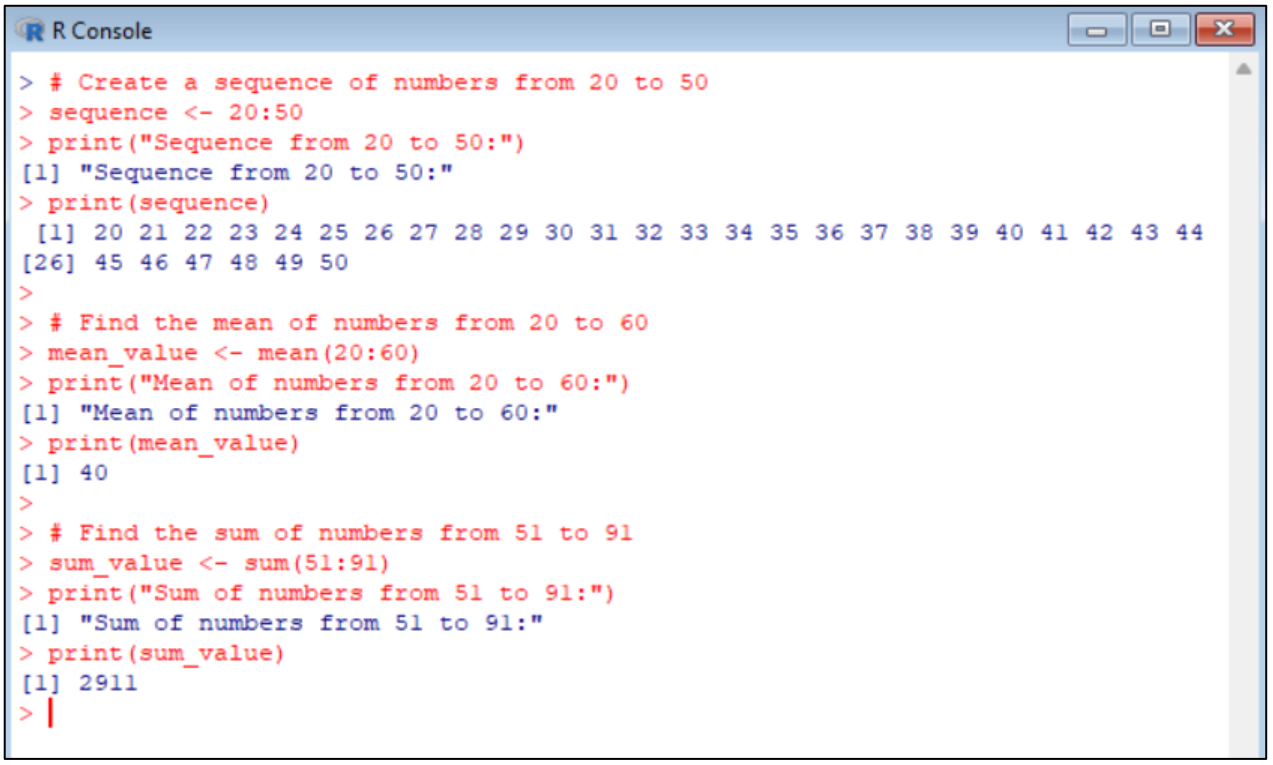
   CODE:

```
# Create a sequence of numbers from 20 to 50
sequence <- 20:50
print("Sequence from 20 to 50:")
print(sequence)

# Find the mean of numbers from 20 to 60
mean_value <- mean(20:60)
print("Mean of numbers from 20 to 60:")
print(mean_value)

# Find the sum of numbers from 51 to 91
sum_value <- sum(51:91)
print("Sum of numbers from 51 to 91:")
print(sum_value)
```

OUTPUT :



7. **Write a R program to create three vectors numeric data, character data and logical data.Display the content of the vectors and their type.**

CODE :

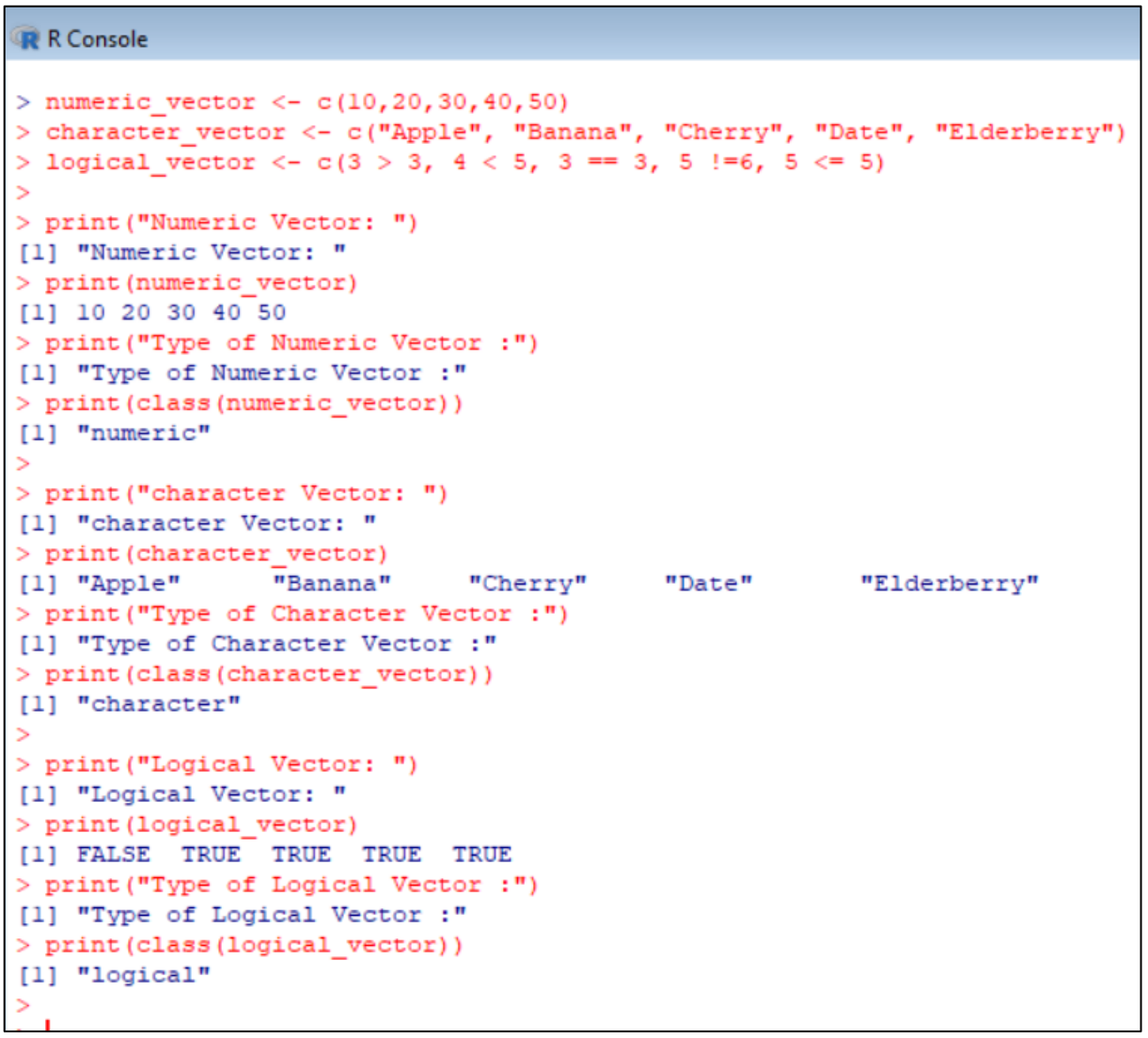
```
numeric_vector <- c(10,20,30,40,50)
character_vector <- c("Apple", "Banana", "Cherry", "Date", "Elderberry")
logical_vector <- c(3 > 3, 4 < 5, 3 == 3, 5 !=6, 5 <= 5)

print("Numeric Vector: ")
print(numeric_vector)
print("Type of Numeric Vector :")
print(class(numeric_vector))

print("character Vector: ")
print(character_vector)
print("Type of Character Vector :")
print(class(character_vector))
```

```
print("Logical Vector: ")
print(logical_vector)
print("Type of Logical Vector :")
print(class(logical_vector))
```

OUTPUT :



8. **Write a R program to create a data frame from four given vectors.**
   name = c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura','Kevin', 'Jonas')
   score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)
   attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)
   qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes')

   CODE :

```
name <- c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew',
'Laura','Kevin', 'Jonas')
score <- c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)
attempts <- c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)
qualify <- c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes')

df <- data.frame(Name = name, Score = score, Attempts = attempts, Qualify =
qualify)

print(df)
```

   OUTPUT :

```
> name <- c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew'$
+           'Kevin', 'Jonas')
> score <- c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)
> attempts <- c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)
> qualify <- c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes')
>
> df <- data.frame(Name = name, Score = score, Attempts = attempts, Qualify = qualif$
>
> print(df)
       Name Score Attempts Qualify
1  Anastasia  12.5        1     yes
2       Dima   9.0        3      no
3  Katherine  16.5        2     yes
4      James  12.0        3      no
5      Emily   9.0        2      no
6    Michael  20.0        3     yes
7    Matthew  14.5        1     yes
8      Laura  13.5        1      no
9      Kevin   8.0        2      no
10     Jonas  19.0        1     yes
>
```

9. **Write a R program to extract specific column from a data frame using column name.**

CODE :

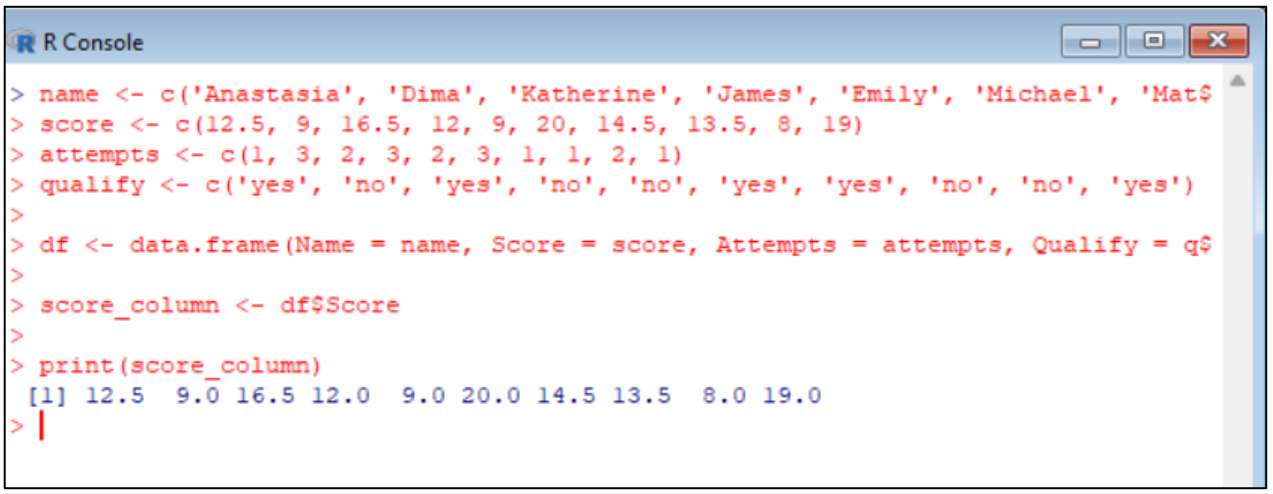
```
name <- c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew',
'Laura','Kevin', 'Jonas')
score <- c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)
attempts <- c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)
qualify <- c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes')

df <- data.frame(Name = name, Score = score, Attempts = attempts, Qualify =
qualify)

score_column <- df$Score

print(score_column)
```

OUTPUT :



10. Write a R program to create an ordered factor from data consisting of the names of months.

CODE :

```
months <- c('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
            'September', 'October', 'November', 'December')

ordered_months <- factor(months, levels = months, ordered = TRUE)

print(ordered_months)
```

OUTPUT :

```
> months <- c('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
+ 'September', 'October', 'November', 'December')
>
> ordered_months <- factor(months, levels = months, ordered = TRUE)
>
> print(ordered_months)
 [1] January   February  March     April     May       June      July      August    September October   November  December
Levels: January < February < March < April < May < June < July < August < September < October < November < December
>
```