Unit-3:
Introduction to R and working with Data

3.1 Overview of R and its applications in data analysis and statistics
3.2 Installing R and RStudio,
3.3 Basic R syntax, variables, and data types
3.4 Importing data into R from different file formats (CSV, Excel, etc.).
3.5 read, write and view data using data frames

• What Is R?

According to R-Project.org, "**R is a language and environment for statistical computing and graphics.**" It's an open-source programming language often used as a data analysis and statistical software tool.
R was developed in 1993 by Ross Ihaka and Robert Gentleman and includes linear regression, machine learning algorithms, statistical inference, time series, and more
R is a universal programming language compatible with the Windows, UNIX, and Linux platforms.
The environment features of R program is discussed below:

- A high-performance data storage and handling facility
- A vast, easily understandable, integrated assortment of intermediate tools dedicated to data analysis
- Graphical facilities for data analysis and display that work either for on-screen or hardcopy
- The well-developed, simple and effective programming language, featuring user-defined recursive functions, loops, conditionals, and input and output facilities.

The syntax of R consists of three items:

- Variables, which store data
- Comments, which are used to improve code readability
- Keywords, reserved words that have a special meaning for the compiler

Advantages of R programming

Here is a list of some of its major strong points:

- It’s open-source. No fees or licenses are needed, so it’s a low-risk venture if you’re developing a new program.
- It’s platform-independent. R runs on all operating systems, so developers only need to create one program that can work on competing systems. This independence is yet another reason why R is cost-effective!
- It’s great for statistics. Statistics are a big thing today, and R shines in this regard. As a result, programmers prefer it over other languages for statistical tool development.
- It’s well suited for Machine Learning. R is ideal for machine learning operations such as regression and classification. It even offers many features and packages for artificial neural network development.

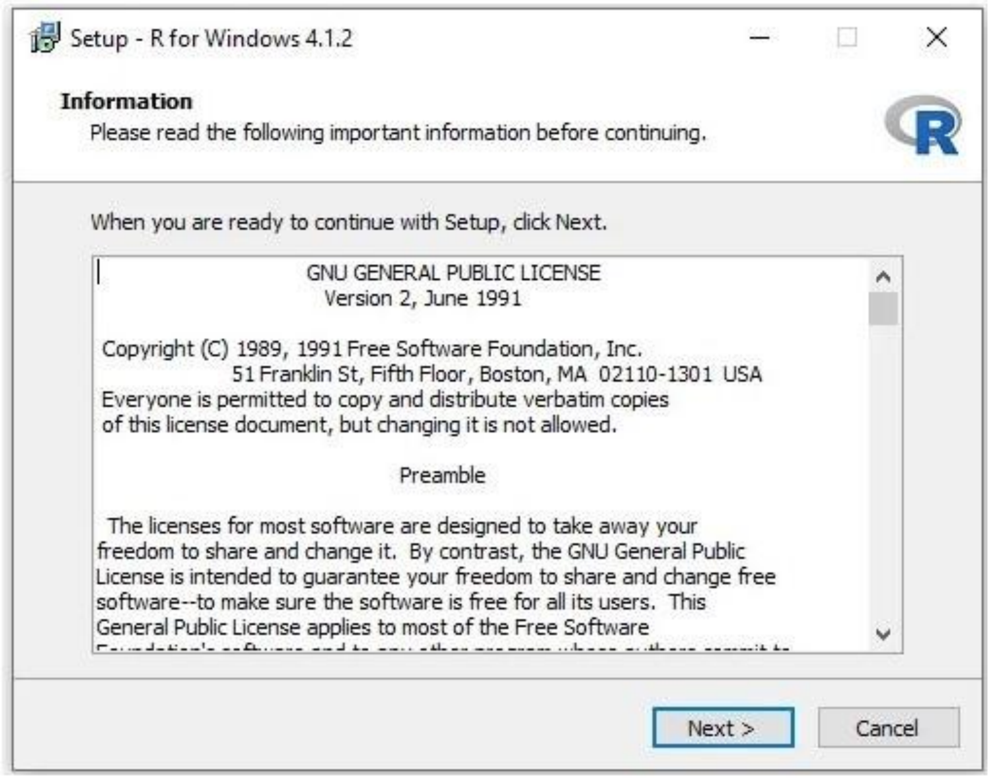
Its applications in data analysis and statistics
R is widely recognized as one of the premier programming languages and environments for statistical computing and data analysis. Its applications span a wide range of domains within data analysis and statistics, including:
1. Exploratory Data Analysis (EDA):
R provides powerful tools for summarizing data, visualizing distributions, detecting outliers, and exploring relationships between variables.
Packages like `ggplot2`, `plotly`, and `ggvis` are popular for creating detailed and customizable visualizations.

2. Statistical Modeling:
R supports a vast array of statistical models, ranging from simple linear regression to complex multilevel models and survival analysis.
Packages like `stats`, `lme4`, `survival`, and `brms` are used for fitting various types of models to data.

• Installing R and RStudio,

To install R on Windows OS:

1. Go to the CRAN website.
2. Click on **"Download R for Windows"**.
3. Click on **"install R for the first time"** link to download the R executable (.exe) file.
4. Run the R executable file to start installation, and allow the app to make changes to your device.
5. Select the installation language.
6. Follow the installation instructions.



Setup - R for Windows 4.1.2

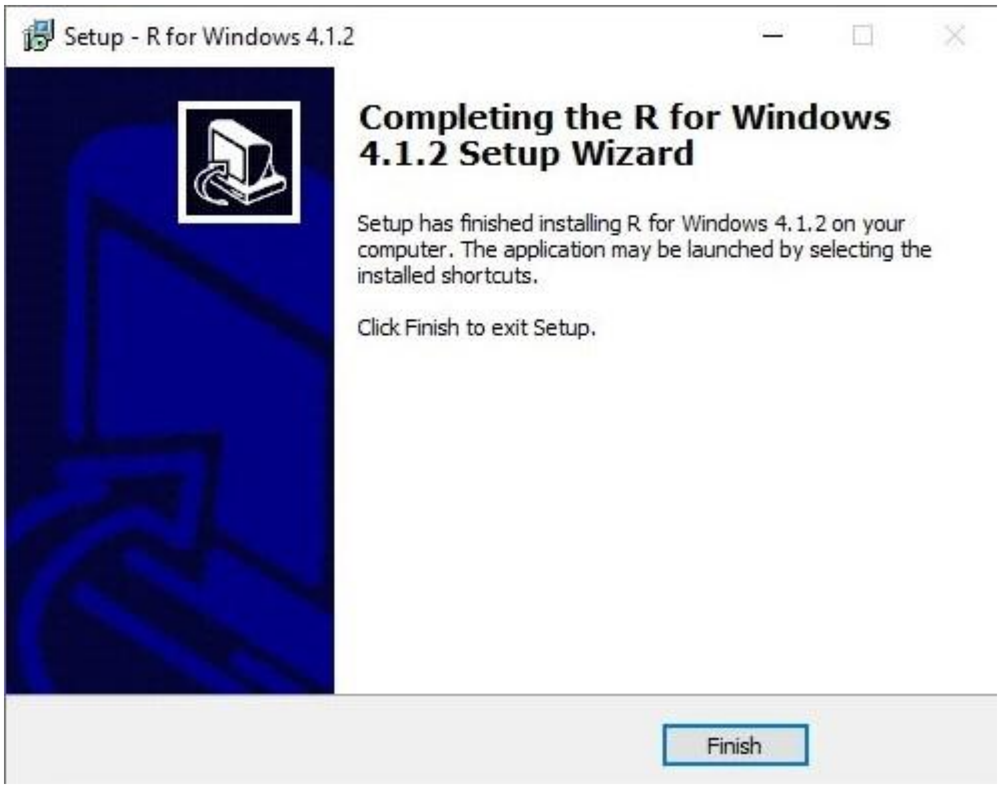
Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
```

7. Click on **"Finish"** to exit the installation setup.



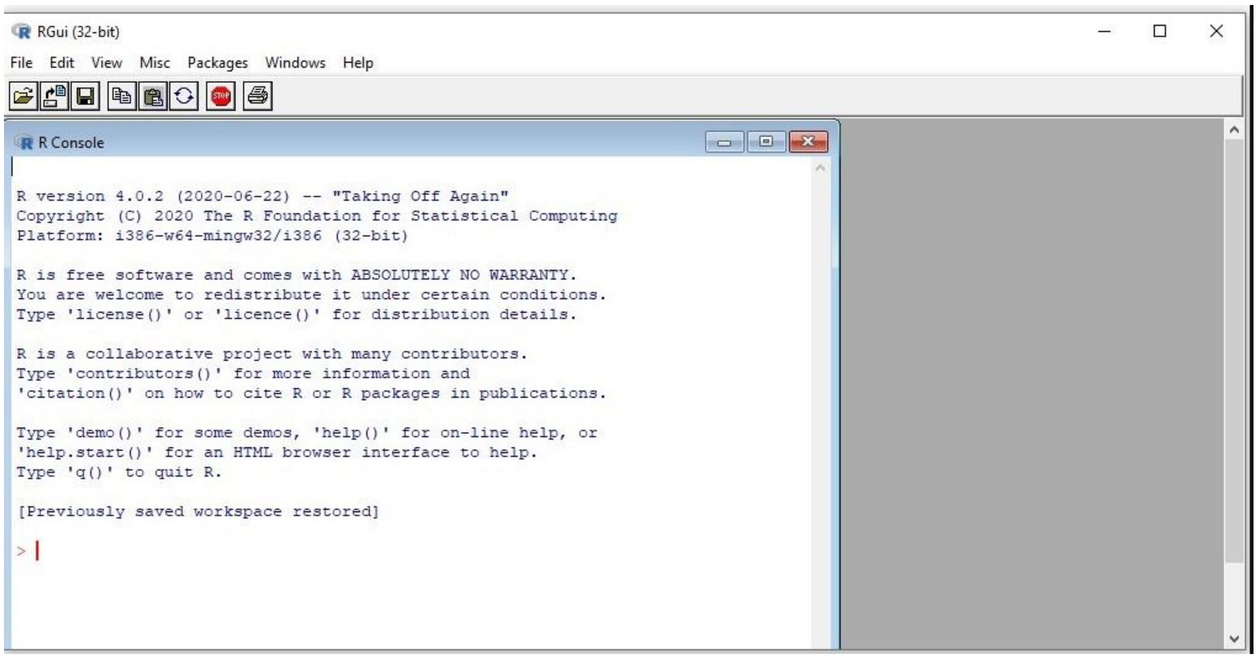
Setup - R for Windows 4.1.2

Completing the R for Windows 4.1.2 Setup Wizard

Setup has finished installing R for Windows 4.1.2 on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

R has now been sucessfully installed on your Windows OS. Open the R GUI to start writing R codes.

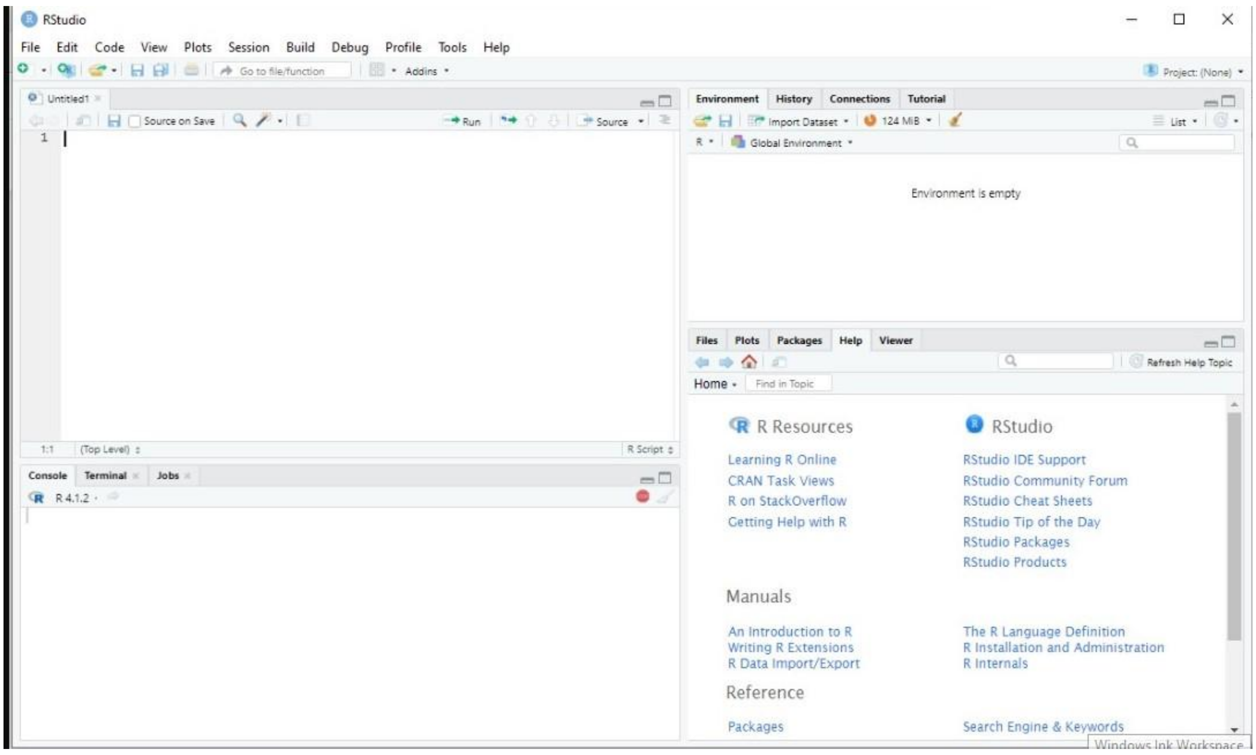


```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

• Installing RStudio Desktop

To install RStudio Desktop on your computer, do the following:

1. Go to the RStudio website.
2. Click on **"DOWNLOAD"** in the top-right corner.
3. Click on **"DOWNLOAD"** under the **"RStudio Open Source License"**.
4. Download RStudio Desktop recommended for your computer.
5. Run the RStudio Executable file (.exe) for Windows OS.
6. Follow the installation instructions to complete RStudio Desktop installation.
7. RStudio is now successfully installed on your computer. The RStudio Desktop IDE interface is shown in the figure below:



• Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

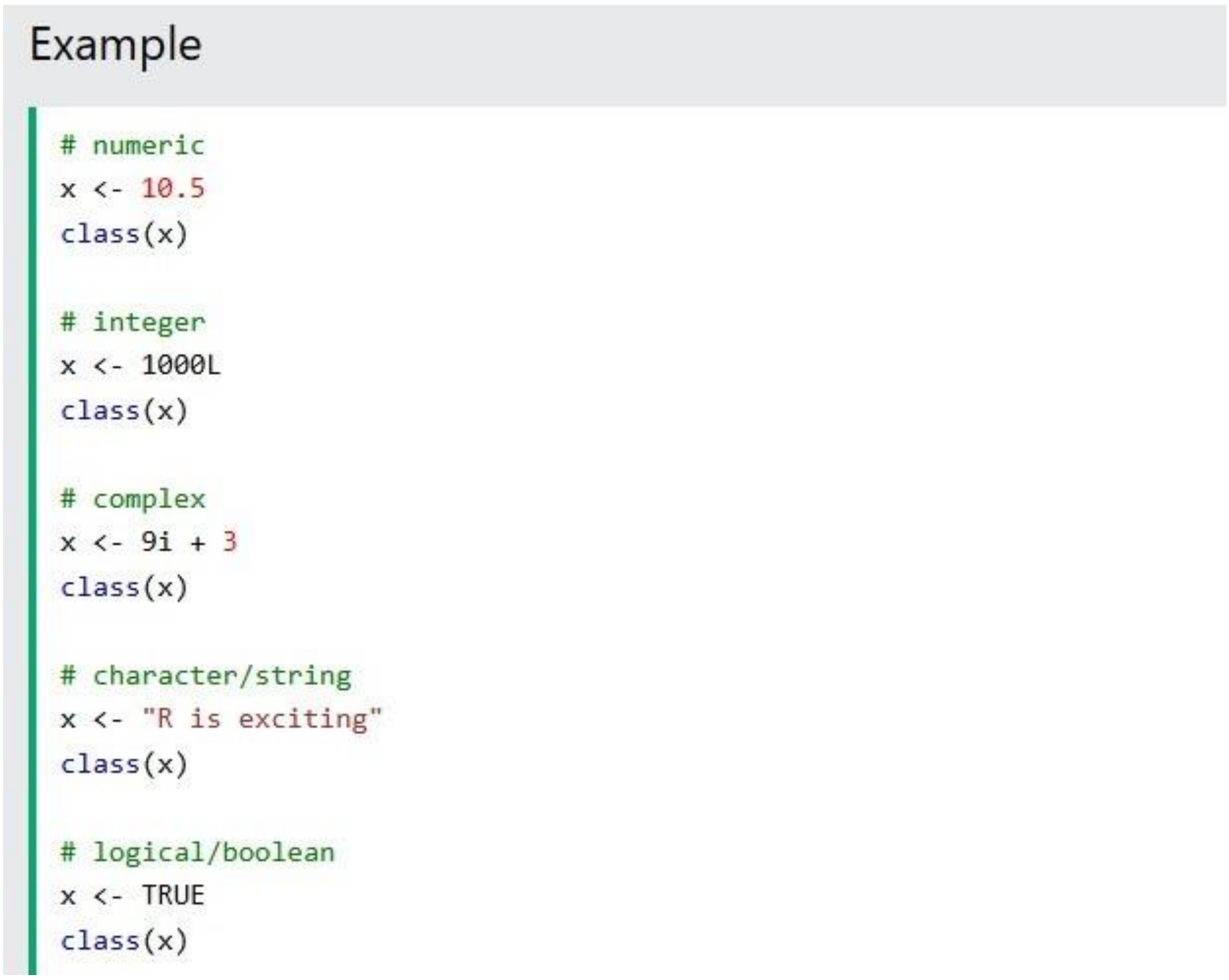
In R, variables do not need to be declared with any particular type, and can even change type after they have been set

• 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:

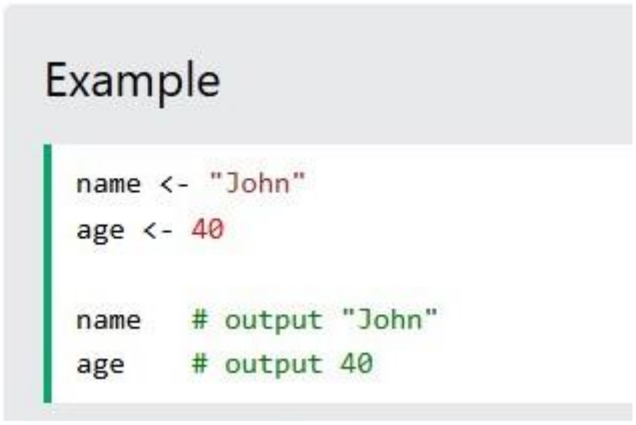

Example

```
# numeric
x <- 10.5
class(x)

# integer
x <- 1000L
class(x)

# complex
x <- 9i + 3
class(x)

# character/string
x <- "R is exciting"
class(x)

# logical/boolean
x <- TRUE
class(x)
```

- **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
```

In R we must use . and _(under score) in variable name other symbole is not allowed in variable name like *,-,&,%,# etc.

In starting of variable we may use . and any character only we don’t use any digit or any symbole in starting of variable name.

We can also assign variable like:

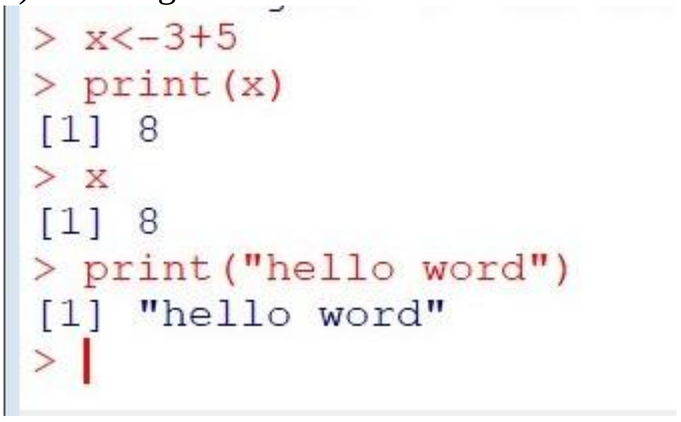
Var1<-10

Var2=10

10->var3

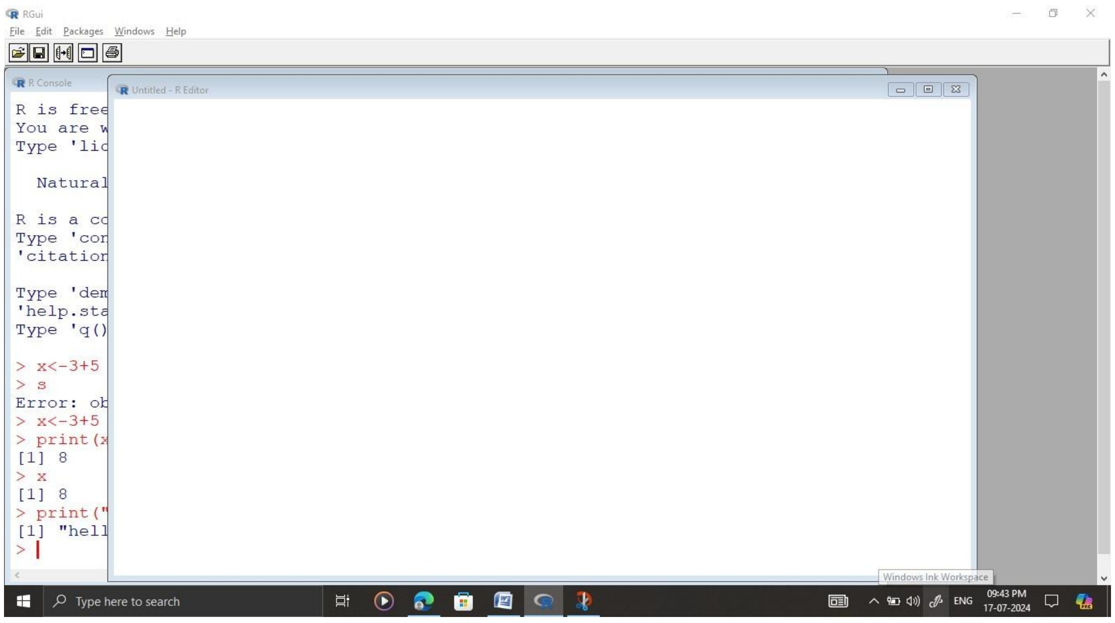
- ## Some important features in R

1) For Increase in size : edit > gui preferences > increase size.
2) For print a data: print(“hello word”)
3) For assign variable: <-



```
> x<-3+5
> print(x)
[1] 8
> x
[1] 8
> print("hello word")
[1] "hello word"
>
```

4) for open R editor: File > New script



```
R is free
You are w
Type 'lic

  Natural

R is a co
Type 'cor
'citation

Type 'dem
'help.sta
Type 'q()

> x<-3+5
> s
Error: ob
> x<-3+5
> print(x
[1] 8
> x
[1] 8
> print("
[1] "hell
> 
```

In R editor we can write a R code and execute it in R consol by using ctrl+r by line by line.

And if we have to run all the program then we have to use : edit > run all option.

5) R is case sensitive language.

6) We print the variable by the use of print function .

Ex. Print("hello")

7) we can use the function cat for print multiple variables.

Ex. cat(x," ",x1)

8) We can define comment by the use of #.

Ex. # Wel come to the first session of R programming.

9) We can quite the R console by the useing of q().

10) If we have to convert other the data type into numeric then, we use the function as.numeric()

Ex. W<-as.numeric(25L)
W

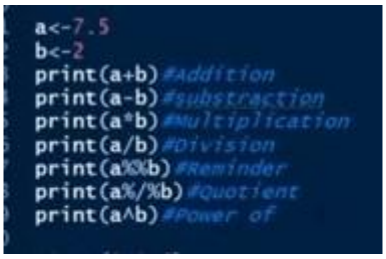
25

10) If we have to convert other the data type into integer then, we use the function as.integer()

Ex. W<-as.numeric(25.75)
W
25

- **<u>operation in R programming.</u>**

Arithmetic operation (+, -, *, /, %%, %/%, ^)
Ex.



```
a<-7.5
b<-2
print(a+b) #Addition
print(a-b) #substraction
print(a*b) #Multiplication
print(a/b) #Division
print(a%%b) #Reminder
print(a%/%b) #Quotient
print(a^b) #Power of
```

Relational operation

<	Less than
>	Greater than
==	Equal to
<=	Less than equal to
>=	Greater than equal to
!=	Not equal to

Logical operator

&	And
/	Or
!	Not

Assignment operator

<- , = , -> , <<- , ->>

- **<u>Conditional statement in R</u>**

1) if statement
# Example 1: Basic if statement

```
x <- 10

if (x > 5) {
  print("x is greater than 5")
}
```

2) if else statement

```
# Example 2: if-else statement
x <- 3

if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is not greater than 5")
}
```

3) if else-if else statement

```
# Example 3: if else-if  else statement
x <- 7

if (x > 10) {
  print("x is greater than 10")
} else if (x > 5) {
  print("x is greater than 5 but not greater than 10")
} else {
  print("x is 5 or less")
}
```

4) nested if statement

```
# Example 4: Nested if statements
x <- 12

if (x > 5) {
  if (x < 10) {
    print("x is between 5 and 10")
  } else {
    print("x is greater than or equal to 10")
  }
} else {
  print("x is 5 or less")
}
```

- <u>**looping statement**</u>

1) for Loop

A for loop is used when you know exactly how many times you want to execute a block of code. It iterates over a sequence of values, such as a sequence of numbers or elements in a vector.

Statement

```
for (variable in sequence) {
  # Code block to be executed
    Print(“”)
}
# Example 1: Looping over a sequence of numbers
for (i in 1:5) {
  print(i)
}
Output
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

2) while Loop

A while loop is used when you want to execute a block of code repeatedly as long as a condition is TRUE.

```
while (condition) {
  # Code block to be executed
}
```

condition: A logical expression that is evaluated before each iteration. If TRUE, the loop continues; if FALSE, the loop terminates.

Example

```
# Example: while looP
count <- 1
```

```
while (count <= 5) {
  print(count)
  count <- count + 1
}
```

Output

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

3) Control Statements (break and next)

In both for and while loops, you can use break to exit the loop prematurely and next to skip the current iteration and proceed to the next one.

Example:

```
# Example: using break and next
for (i in 1:10) {
  if (i == 3) {
    next  # Skip iteration when i equals 3
  }
  if (i == 8) {
    break  # Exit loop when i equals 8
  }
  print(i)
}
```

Output

```
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
[1] 7
```

4) Nested Loops

You can nest loops inside one another to perform more complex tasks, such as iterating over multiple dimensions or processing nested data structures.

Example: nested loops

```
for (i in 1:3) {
  for (j in 1:2) {
   print(paste("i:", i, "j:", j))
  }}
```

Output

```
[1] "i: 1 j: 1"
[1] "i: 1 j: 2"
[1] "i: 2 j: 1"
[1] "i: 2 j: 2"
[1] "i: 3 j: 1"
[1] "i: 3 j: 2"
```

R functions

In R, a function is a block of code that performs a specific task and can be reused throughout your script or session. Functions in R are defined using the `function()` keyword. Here's a basic overview of how functions work in R:

Syntax:

```
function_name <- function(arg1, arg2, ...) {
  # Function body: code to execute
  statements
  return(value)  # Optional: specify the return value
}
```

1. Function Name:

This is the name you give to your function, which you use to call it later in your script.

2. Arguments (Parameters):

These are placeholders for values that you pass into the function. They are enclosed in parentheses `()` after the function name.

Arguments are optional. If your function doesn't require any inputs, you can leave them empty.

3. Function Body:

This is where you write the code that you want the function to execute. It can contain any valid R expressions, assignments, control structures (like `if`, `else`, `for`, `while`), and other function calls.

4. Return Value:

The `return()` statement specifies the value that the function should return to the caller. It's optional; if omitted, the function returns the value of the last expression evaluated.

Examples:

Example 1: Simple Function

```
# Define a function that adds two numbers
add_numbers <- function(x, y) {
 result <- x + y
 return(result)
}
# Call the function with arguments

sum_result <- add_numbers(3, 5)
print(sum_result)

# Output: 8
```

Example 2: Function without Arguments

```
# Define a function that prints a greeting
greet <- function() {
 print("Hello, world!")
}
# Call the function
greet()
# Output: Hello, world!
```

Example 3: Function with Default Argument

```
# Define a function with a default argument
power <- function(x, exp = 2) {
  result <- x ^ exp
  return(result)
}

# Call the function with different arguments
```

```
power(3)     # Output: 9 (default exponent 2)
power(2, 3)  # Output: 8 (2^3)
```

Notes:

- Functions in R are first-class objects, meaning they can be assigned to variables, passed as arguments to other functions, and returned as values from other functions.

- Arguments in R functions can have default values. If a default value is provided, the argument becomes optional when calling the function.

Using functions allows you to modularize your code, improve readability, and facilitate code reuse, which are fundamental principles in writing efficient and maintainable R scripts.

• Take input from user

In R, the readline() function is used to interactively prompt the user for input within a script or function. It reads a line of text from the console and returns it as a character string. Here’s how you can use it:

```
# Example 1: Basic usage
user_input <- readline(prompt = "Enter your name: ")
cat("Hello, ", user_input, "! Nice to meet you.\n")

# Example 2: Using readline in a function
greet_user <- function() {
name <- readline(prompt = "Enter your name: ")
cat("Hello, ", name, "! Welcome.\n")
}
    greet_user()  # Call the function to prompt for input
```

 Basic Usage:

- readline(prompt = "Enter your name: "): This line prompts the user to enter their name with the specified message ("Enter your name: ").
- The user's input is captured in the variable user_input.
- cat("Hello, ", user_input, "! Nice to meet you.\n"): This line prints a greeting message using the value entered by the user.

 Function Example:

- greet_user() is a function that uses readline() to prompt for the user's name.
- Inside the function, name <- readline(prompt = "Enter your name: ") captures the user's input.
- cat("Hello, ", name, "! Welcome.\n") then prints a personalized greeting.

Built in function

R has a wealth of built-in functions that cover a wide range of tasks, from basic arithmetic to complex statistical analyses. Here are some common categories of built-in functions in R:

1. Mathematical Functions

- sqrt(x): Square root of x.
- log(x, base): Natural logarithm of x or logarithm to a specified base.
- exp(x): Exponential function.

2. Statistical Functions

- mean(x): Mean of x.
- median(x): Median of x.
- sd(x): Standard deviation of x.
- var(x): Variance of x.
- summary(x): Summary statistics of x.

3. Vector and Matrix Operations

- length(x): Length of a vector or list.
- dim(x): Dimensions of an object (e.g., matrix).
- t(x): Transpose of a matrix.

4. Character and String Functions

- nchar(x): Number of characters in each element of x.

- tolower(x): Convert to lowercase.
- toupper(x): Convert to uppercase.
- substr(x, start, stop): Extract or replace substrings.

5. Control Flow

- ifelse(test, yes, no): Vectorized conditional statement.
- for(i in 1:n) { }: Looping construct.
- while(condition) { }: Looping construct.

6. Graphics and Plotting

- plot(x, y): Basic scatterplot.
- hist(x): Histogram of x.
- boxplot(x): Boxplot of x.

7. Input and Output

- read.csv(file): Read a CSV file into a data frame.
- write.csv(x, file): Write a data frame to a CSV file.
- scan(): Read data from the console or a file.

8. Date and Time Functions

- Sys.Date(): Current date.
- Sys.time(): Current date and time.
- as.Date(x): Convert an object to a date.

These are just a few examples of the extensive set of built-in functions in R. For a comprehensive list and detailed documentation, you can use R’s help system with ?function_name or help(function_name).

Data structure in R

Data structure is a way to store a data in a memory.

(vector , matrix , array , list , data frames)

1. Vectors

- **Definition**: A vector is a one-dimensional array that holds elements of the same type (e.g., numeric, character, logical).
- elements of a vector are known as component
- **Creation**: Use the c() function.

- if we have to find the length of a vector then use length() function.
- For indexing we can use[] (in R indexing starts from 1).
- We use names function for give the names() of vector elements.

Example:1

```
numeric_vector <- c(1, 2, 3, 4)
char_vector <- c("apple", "banana", "cherry")

x<-1:10
```

by the use of seq() function

```
sq<-seq(1,3,length.out=5)
    seq(from=3.5,to=1.5,by=.5)
    seq(from=-2.7,to=1.5,length.out=.5)
```

2. Lists

- **Definition**: A list is a one-dimensional array that can hold elements of different types, including other lists.
- **Creation**: Use the list() function.

  ```
  my_list <- list(name="Alice", age=25, scores=c(90, 85, 88))
  ```

  example

  ```
  my_list <- list(c("rer","ter","ttt"), c(55,65,70), c("b.com","b.b.a.","b.ca."))
  my_list
  #here we give name of each list in my_list.
  names(my_list)=c("name","roll no.","department")
  my_list
  ```

- For indexing the element of list we can use print(my_list[1]) or print(my_list$`roll no.`)

- V<-unlist(my_list) function convert list into vector.

3. Matrices

- **Definition**: A matrix is a two-dimensional array where all elements must be of the same type.
- **Creation**: Use the matrix() function.
- Matrix(data,nrow,ncolumn,byrow,dim_name)

```
my_matrix <- matrix(1:6, nrow=2, ncol=3)


my_matrix <- matrix(1:9, nrow=3, ncol=3,byrow=FALSE)
my_matrix
```

4. Arrays

- **Definition**: An array is a multi-dimensional extension of a matrix. It can have more than two dimensions.
- **Creation**: Use the array() function.

```
my_array <- array(1:24, dim=c(2, 3, 4))



v1<-c(12:17)
v2<-c(15,17,19,20)
row_name<-c("r1","r2","r3")
col_name<-c("c1","c2","c3")
mat_name<-c("mat1","mat2")
my<-
+array(c(v1,v2),dim=c(3,3,2),dimnames=list(row_name,col_name,mat_n
ame))#give name to row column and matrix
my
c<-print(my[3,2,2])#indexing the element of array
```

- We also use add or substract two array like v1+v2.

5. Data Frames

- **Definition**: A data frame is a two-dimensional table-like structure where each column can be of a different type. It is similar to a spreadsheet or SQL table.
- **Creation**: Use the data.frame() function.

```
my_df <- data.frame(
Name = c("ram", "shyam", "raj"),
Age = c(25, 30, 35),
Score = c(90, 85, 88)
)
```

- We can convert data frame into string by the use of str() function.
- We can indexing in data frame with the help of [] bracket and $ sign.

- Ex:
  print(my_df[1])
  print(my_df[1,2])
  print(my_df[1,])
  print(my_df([,3])
  my_df$name
  my_df$Name<-c("aa","bb","cc")
  my_df

- We can add the new column and row by the use of cbind() and rbind() function in data frame.
  Ex:
    cbind(my_df,assign=c(78,76,78))
    rbind(my_df,c("AA",40,88,77))
- Also we can combine two data frame with the help of cbind() and rbind() function.
  Ex: v<-rbind(my_df1,my_df2)
- We can check the dimention of data frame with the use of dim() function.
  Ex: dim(my_df) #give the number of row and column in data frame.

- we can get the sub data frame by the help of subset() function.

Ex: subset(my_df,Name!="aa")

- <u>Importing data into R from different file formats (CSV, Excel).</u>

1. CSV Files

CSV (Comma Separated Values) files are one of the most straightforward formats to import into R.

# Assuming your CSV file is named 'data.csv' and is located in your working directory
data <- read.csv("data.csv")

- Some function for csv file

getwd()=give current working directort

```
setwd("f:material/r") =set current working directort

aa<-read.csv("people.csv")
View(aa)
fix(aa)=fixong the csv file
str(aa)=show the structure of the data frame
summary(aa)=give statistical summary of the csv file
names(aa)=provide all the variable name.
nrow(aa)=provide number of row.
ncol(aa)=provide number of colume.
length(aa)=give length of file.
dim(aa)=show the dimention of the data frame.
colnames(aa)=return column names.
head(aa)=show first six row of file.
tail(aa)=show last six row of file.
bb<-aa[c(1:2,3,7)]=give 1,2.3.7 column of file
View(bb)
cc<-aa[c(1:3),c(1:3)]=provide first three column and three row of data
View(cc)
names(aa)
vv<-aa$User.Id[]=for indexing the data value.
Vv
ff<-subset(aa,Sex=="Male")
View(ff)
```

2. Excel Files

For Excel files, you typically need to use the readxl package, which provides functions to read Excel files into R.
First, make sure to install the readxl package if you haven't already:

```
install.packages("readxl")
```

Then, you can use the read_excel() function to import data from Excel:

```
library(readxl)

# Assuming your Excel file is named 'data.xlsx' and is located in your
working directory
data <- read_excel("data.xlsx")
```