

## UNIT 3 : Introduction to R and working with Data

R is a statistical computing and graphics system. This system is comprised of two parts: the R language itself (which is what most people mean when they talk about R) and a run-time environment.

R is an interpreted language, which means that users access its functions through a command-line interpreter.

Unlike languages such as Python and Java, R is not a general-purpose programming language. Instead, it's considered a domain-specific language (DSL), meaning its functions and use are designed for a specific area of use or domain.

In R's case, that's statistical computing and analysis. By extension, R is commonly used for all manner of data science tasks.

R is equipped with a large set of functions that enable data visualizations, so users can analyze data, model it as required, and then create graphics. In addition to the language's in-built graphical functions, there are numerous add-ons or modules that facilitate this.

### Why Use R Language?

The **R Language** is a powerful tool widely used for data analysis, statistical computing, and machine learning. Here are several reasons why professionals across various fields prefer R:

#### **1. Comprehensive Statistical Analysis:**

- R language is specifically designed for statistical analysis and provides a vast array of statistical techniques and tests, making it ideal for data-driven research.

#### **2. Extensive Packages and Libraries:**

- The R Language boasts a rich ecosystem of packages and libraries that extend its capabilities, allowing users to perform advanced data manipulation, visualization, and machine learning tasks with ease.

#### **3. Strong Data Visualization Capabilities:**

- R language excels in data visualization, offering powerful tools like ggplot2 and plotly, which enable the creation of detailed and aesthetically pleasing graphs and plots.

#### **4. Open Source and Free:**

- As an open-source language, R is free to use, which makes it accessible to everyone, from individual researchers to large organizations, without the need for costly licenses.

#### **5. Platform Independence:**

- The R Language is platform-independent, meaning it can run on various operating systems, including Windows, macOS, and Linux, providing flexibility in development environments.

#### **6. Integration with Other Languages:**

- R can easily integrate with other programming languages such as C, C++, Python, and Java, allowing for seamless interaction with different data sources and statistical packages.

## UNIT 3 : Introduction to R and working with Data

### **7. Growing Community and Support:**

- R language has a large and active community of users and developers who contribute to its continuous improvement and provide extensive support through forums, mailing lists, and online resources.

### **8. High Demand in Data Science:**

- R is one of the most requested programming languages in the Data Science job market, making it a valuable skill for professionals looking to advance their careers in this field.

## **Features of R Programming Language**

The **R Language** is renowned for its extensive features that make it a powerful tool for data analysis, statistical computing, and visualization. Here are some of the key features of R:

### **1. Comprehensive Statistical Analysis:**

- R language provides a wide array of statistical techniques, including linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, and clustering.

### **2. Advanced Data Visualization:**

- With packages like ggplot2, plotly, and lattice, R excels at creating complex and aesthetically pleasing data visualizations, including plots, graphs, and charts.

### **3. Extensive Packages and Libraries:**

- The Comprehensive R Archive Network (CRAN) hosts thousands of packages that extend R's capabilities in areas such as machine learning, data manipulation, bioinformatics, and more.

### **4. Open Source and Free:**

- R is free to download and use, making it accessible to everyone. Its open-source nature encourages community contributions and continuous improvement.

### **5. Platform Independence:**

- R is platform-independent, running on various operating systems, including Windows, macOS, and Linux, which ensures flexibility and ease of use across different environments.

### **6. Integration with Other Languages:**

- R language can integrate with other programming languages such as C, C++, Python, Java, and SQL, allowing for seamless interaction with various data sources and computational processes.

### **7. Powerful Data Handling and Storage:**

- R efficiently handles and stores data, supporting various data types and structures, including vectors, matrices, data frames, and lists.

### **8. Robust Community and Support:**

- R has a vibrant and active community that provides extensive support through forums, mailing lists, and online resources, contributing to its rich ecosystem of packages and documentation.

### **9. Interactive Development Environment (IDE):**

- RStudio, the most popular IDE for R, offers a user-friendly interface with features like syntax highlighting, code completion, and integrated tools for plotting, history, and debugging.

## UNIT 3 : Introduction to R and working with Data

### 10. Reproducible Research:

- R supports reproducible research practices with tools like R Markdown and Knitr, enabling users to create dynamic reports, presentations, and documents that combine code, text, and visualizations.

### Fields and industries where R is used

Academia

Data Science

Statistics

Social Media

R is especially effective when used for

- Data analysis
- Statistical inference
- Machine learning algorithms

R offers a wide variety of statistics-related libraries and provides a favorable environment for statistical computing and design. In addition, the R programming language gets used by many quantitative analysts as a programming tool since it's useful for data importing and cleaning.

### How to Install 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:

## UNIT 3 : Introduction to R and working with Data

```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-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.

Natural language support but running in an English locale

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.

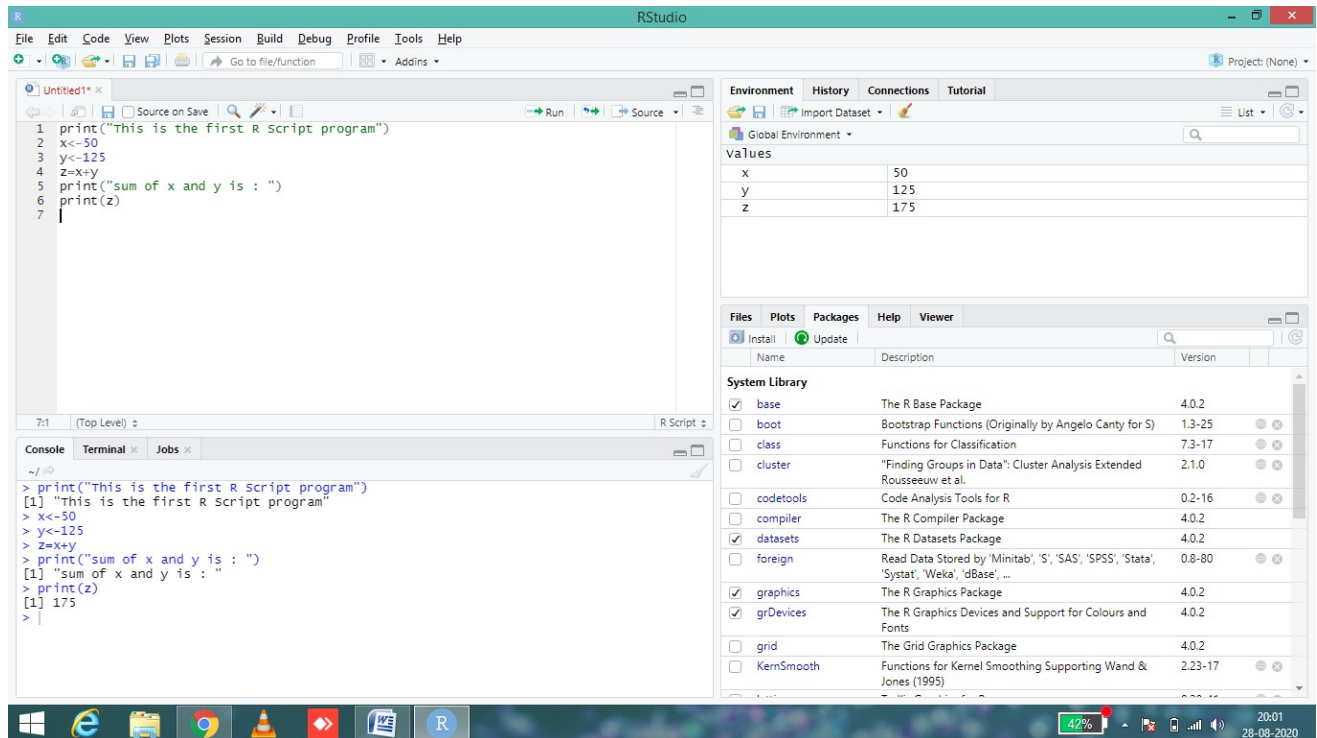
> 5 + 5
[1] 10
> _
```

RStudio Desktop is an IDE for standalone machines.

### **To install RStudio Desktop:**

1. Install R.
2. Go to the RStudio website, navigate to the RStudio Desktop Download page, and download RStudio Desktop.
3. Run the installer and follow the prompts.
4. Click the desktop icon to initialize RStudio.

# UNIT 3 : Introduction to R and working with Data



After installing R :

you will get a prompt > where you can start typing your program as follows –

```
> myString <- "Hello, World!"
> print ( myString)
[1] "Hello, World!"
```

## R Script File

you will do your programming by writing your programs in script files and then you execute those scripts at your command prompt with the help of R interpreter called **Rscript**.

```
myString <- "Hello, World!"

print ( myString)
```

Save the above code in a file with extension .R. When we run the above program, it produces the following result.

```
[1] "Hello, World!"
```

## UNIT 3 : Introduction to R and working with Data

### Comments

Comments are like helping text in your R program and they are ignored by the interpreter while executing your actual program. Single comment is written using # in the beginning of the statement as follows –

```
# My first program in R Programming
```

### Print

Unlike many other programming languages, you can output code in R without using a print function:

```
print("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:

```
name <- "Sachin"
age <- 50

name # output "Sachin"
age  # output 50
```

it is common to use = as an assignment operator. In R, we can use both = and <- as assignment operators. Compared to many other programming languages, you do not have to use a function to print/output variables in R. You can just type the name of the variable:

### 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 (,):

## UNIT 3 : Introduction to R and working with Data

```
text <- "awesome"

paste("R is", text)

output : R is awesome.
```

You can also use `,` to add a variable to another variable. If try to combine a string (text) and a number, R will give you an error

### Multiple Variables

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

```
# Assign the same value to multiple variables in one line
var1 <- var2 <- var3 <- "Sachin"

# 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..)

```
# Legal variable names:
```

```
myvar <- "John"
my_var <- "John"
myVar <- "John"
MYVAR <- "John"
myvar2 <- "John"
.myvar <- "John"
```

```
# Illegal variable names:
```

```
2myvar <- "John"
my-var <- "John"
```

## UNIT 3 : Introduction to R and working with Data

```
my var <- "John"  
_my_var <- "John"  
my_v@ar <- "John"  
TRUE <- "John"
```

variable names are case-sensitive

### 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** (string) - ("k", "R is a language", "FALSE", "11.5")
- **logical** (boolean) - (TRUE or FALSE)

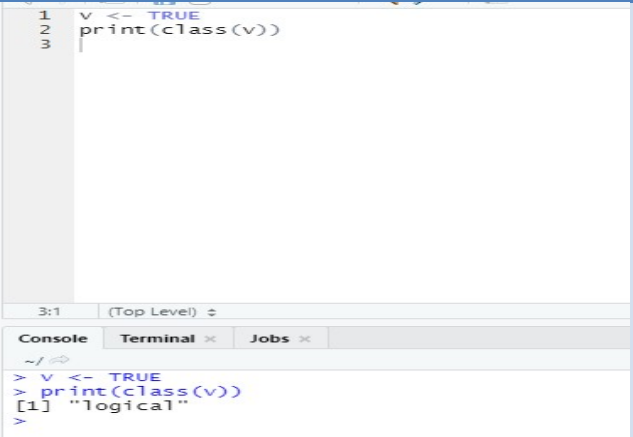
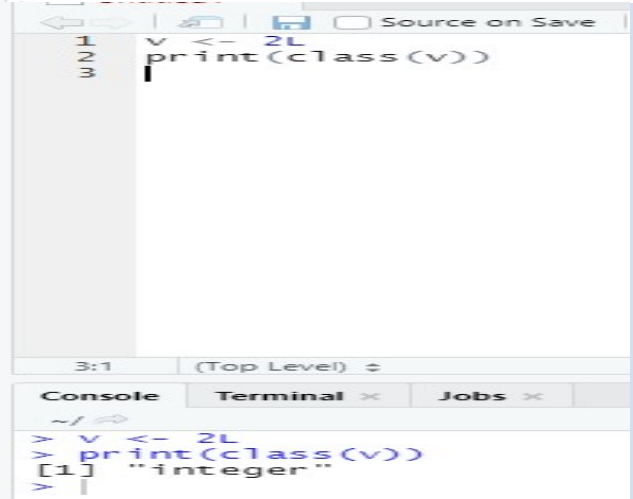
there are many types of R-objects. The frequently used ones are –

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

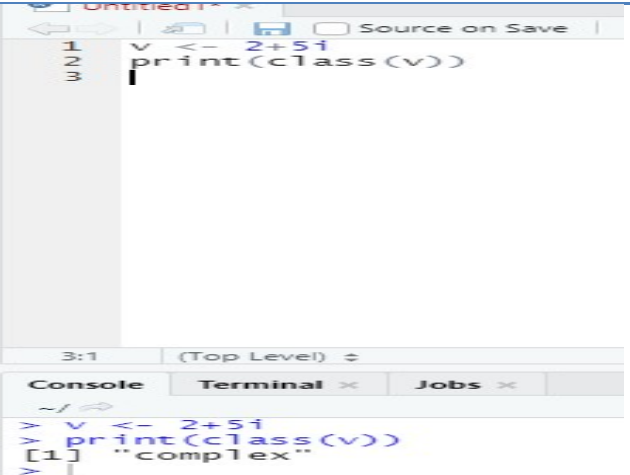
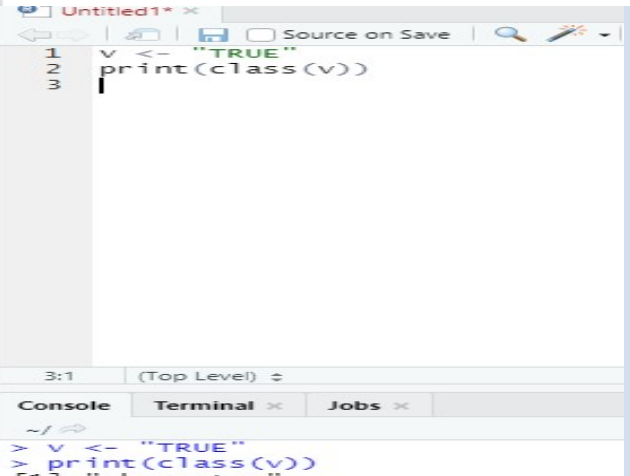
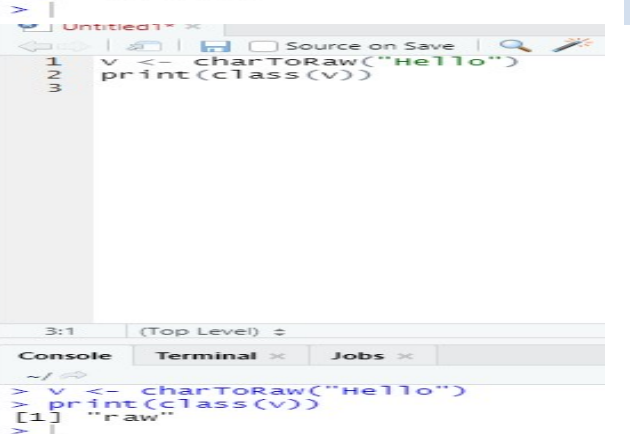
Class() is used to identify the datatype of a variable.



## UNIT 3 : Introduction to R and working with Data

Data Type	Example	Outputs
Logical	TRUE, FALSE	 <pre>1 v &lt;- TRUE 2 print(class(v)) 3</pre> <p>3:1 (Top Level) ↕</p> <p>Console Terminal × Jobs ×</p> <p>~/</p> <pre>&gt; v &lt;- TRUE &gt; print(class(v)) [1] "logical" &gt;</pre>
Numeric	253, 25.2, 899	 <pre>1 v &lt;- 23.5 2 print(class(v)) 3 4</pre> <p>3:1 (Top Level) ↕</p> <p>Console Terminal × Jobs ×</p> <p>~/</p> <pre>&gt; v &lt;- 23.5 &gt; print(class(v)) [1] "numeric" &gt;</pre>
Integer	25L, 2L	 <pre>1 v &lt;- 2L 2 print(class(v)) 3</pre> <p>3:1 (Top Level) ↕</p> <p>Console Terminal × Jobs ×</p> <p>~/</p> <pre>&gt; v &lt;- 2L &gt; print(class(v)) [1] "integer" &gt;</pre>

## UNIT 3 : Introduction to R and working with Data

<b>Complex</b>	$25+8i$	 <pre>1 v &lt;- 2+5i 2 print(class(v)) 3</pre> <p>3:1 (Top Level)</p> <p>Console Terminal Jobs</p> <pre>&gt; v &lt;- 2+5i &gt; print(class(v)) [1] "complex" &gt;</pre>
<b>Character</b>	"Hello" "123" "True" "a"	 <pre>1 v &lt;- "TRUE" 2 print(class(v)) 3</pre> <p>3:1 (Top Level)</p> <p>Console Terminal Jobs</p> <pre>&gt; v &lt;- "TRUE" &gt; print(class(v)) [1] "character" &gt;</pre>
<b>Raw</b>	"Hello" is stored as 48 65 6c 6c 6f	 <pre>1 v &lt;- charToRaw("Hello") 2 print(class(v)) 3</pre> <p>3:1 (Top Level)</p> <p>Console Terminal Jobs</p> <pre>&gt; v &lt;- charToRaw("Hello") &gt; print(class(v)) [1] "raw" &gt;</pre>

The variables can be print using functions like: `print()` , `cat()` . The `cat()` function combines multiple items in continuous print option.

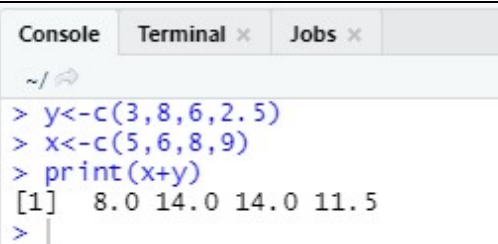
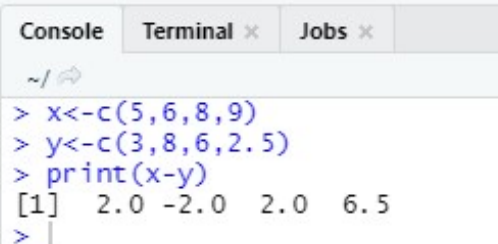
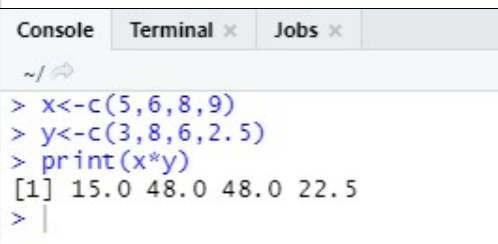
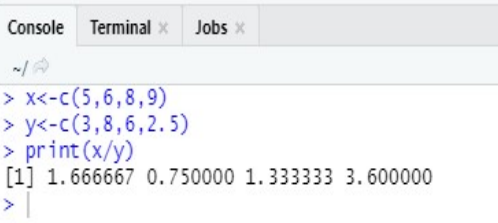
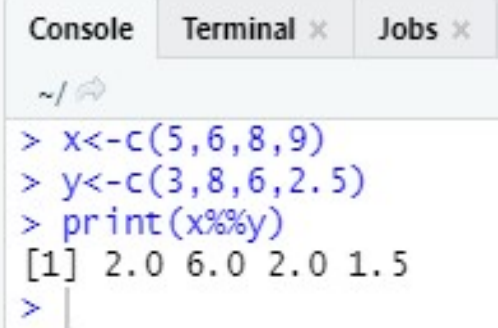
Deleting variables

- `rm()` function is used to delete the variable.
- All the variables can be delete by `rm()` and `ls()` function together.

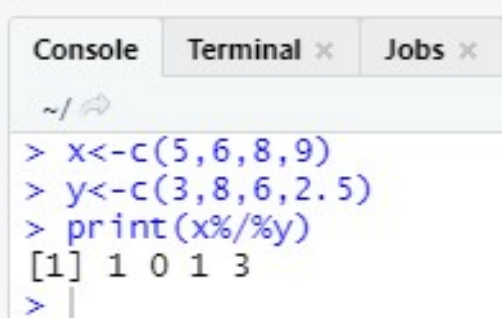
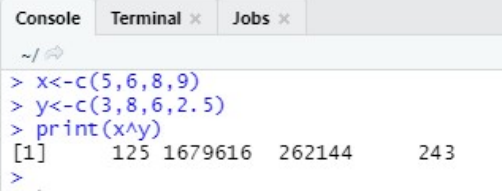
## UNIT 3 : Introduction to R and working with Data

### Operators

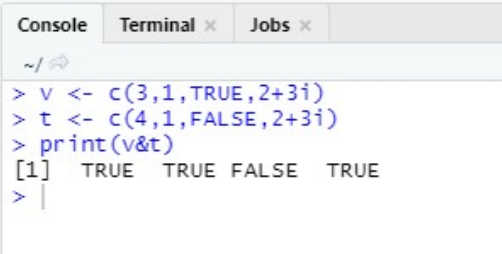
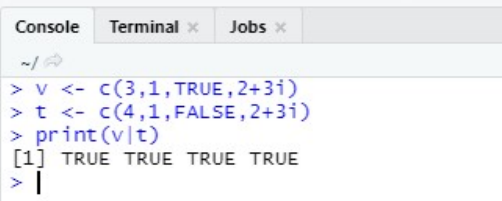
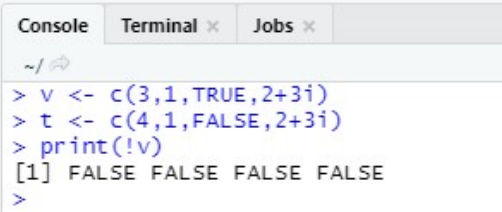
#### Arithmetic Operator

No	Operator	Use	Example
1.	+	Adds two vectors	 <pre> &gt; y&lt;-c(3,8,6,2.5) &gt; x&lt;-c(5,6,8,9) &gt; print(x+y) [1] 8.0 14.0 14.0 11.5 &gt; </pre>
2.	-	Subtracts second vector from the first	 <pre> &gt; x&lt;-c(5,6,8,9) &gt; y&lt;-c(3,8,6,2.5) &gt; print(x-y) [1] 2.0 -2.0 2.0 6.5 &gt; </pre>
3.	*	Multiplies both vectors	 <pre> &gt; x&lt;-c(5,6,8,9) &gt; y&lt;-c(3,8,6,2.5) &gt; print(x*y) [1] 15.0 48.0 48.0 22.5 &gt; </pre>
4.	/	Divide the first vector with the second	 <pre> &gt; x&lt;-c(5,6,8,9) &gt; y&lt;-c(3,8,6,2.5) &gt; print(x/y) [1] 1.666667 0.750000 1.333333 3.600000 &gt; </pre>
5.	%%	Give the remainder of the first vector with the second	 <pre> &gt; x&lt;-c(5,6,8,9) &gt; y&lt;-c(3,8,6,2.5) &gt; print(x%%y) [1] 2.0 6.0 2.0 1.5 &gt; </pre>

## UNIT 3 : Introduction to R and working with Data

6.	%/%	The result of division of first vector with second (quotient)	 <pre> &gt; x&lt;-c(5,6,8,9) &gt; y&lt;-c(3,8,6,2.5) &gt; print(x%/%y) [1] 1 0 1 3 &gt; </pre>
7.	^	The first vector raised to the exponent of second vector	 <pre> &gt; x&lt;-c(5,6,8,9) &gt; y&lt;-c(3,8,6,2.5) &gt; print(x^y) [1] 125 1679616 262144 243 &gt; </pre>

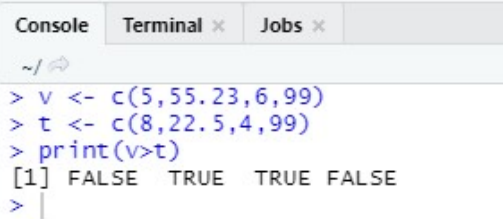
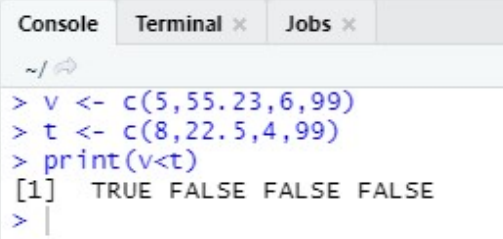
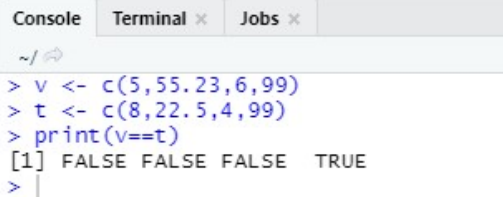
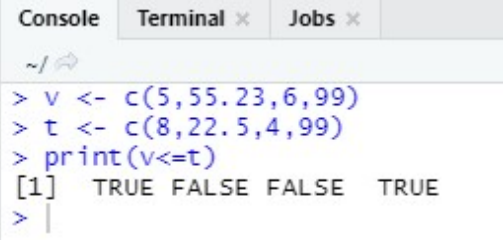
### Logical Operator

No	Operator	Use	Example
1.	&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.	 <pre> &gt; v &lt;- c(3,1,TRUE,2+3i) &gt; t &lt;- c(4,1,FALSE,2+3i) &gt; print(v&amp;t) [1] TRUE TRUE FALSE TRUE &gt; </pre>
2.		It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	 <pre> &gt; v &lt;- c(3,1,TRUE,2+3i) &gt; t &lt;- c(4,1,FALSE,2+3i) &gt; print(v t) [1] TRUE TRUE TRUE TRUE &gt; </pre>
3.	!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	 <pre> &gt; v &lt;- c(3,1,TRUE,2+3i) &gt; t &lt;- c(4,1,FALSE,2+3i) &gt; print(!v) [1] FALSE FALSE FALSE FALSE &gt; </pre>
4.	&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if	<pre> x &lt;- 5 if (x &gt; 0 &amp;&amp; x &lt; 10) {   print("x is positive and less than 10") } else { </pre>

## UNIT 3 : Introduction to R and working with Data

		both are TRUE.	<pre>print("x is either negative or greater than 10") }</pre>
5.		Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	<pre>x &lt;- 5 if (x &gt; 0    x &lt; 10) {   print("x is positive and less than 10") } else {   print("x is either negative or greater than 10") }</pre>

### Relational Operator

No	Operator	Use	Example
1.	>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	 <pre>Console Terminal x Jobs x ~/ &gt; v &lt;- c(5,55.23,6,99) &gt; t &lt;- c(8,22.5,4,99) &gt; print(v&gt;t) [1] FALSE TRUE TRUE FALSE &gt;  </pre>
2.	<	Checks if each element of the first vector is less than the corresponding element of the second vector..	 <pre>Console Terminal x Jobs x ~/ &gt; v &lt;- c(5,55.23,6,99) &gt; t &lt;- c(8,22.5,4,99) &gt; print(v&lt;t) [1] TRUE FALSE FALSE FALSE &gt;  </pre>
3.	==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	 <pre>Console Terminal x Jobs x ~/ &gt; v &lt;- c(5,55.23,6,99) &gt; t &lt;- c(8,22.5,4,99) &gt; print(v==t) [1] FALSE FALSE FALSE TRUE &gt;  </pre>
4.	<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	 <pre>Console Terminal x Jobs x ~/ &gt; v &lt;- c(5,55.23,6,99) &gt; t &lt;- c(8,22.5,4,99) &gt; print(v&lt;=t) [1] TRUE FALSE FALSE TRUE &gt;  </pre>

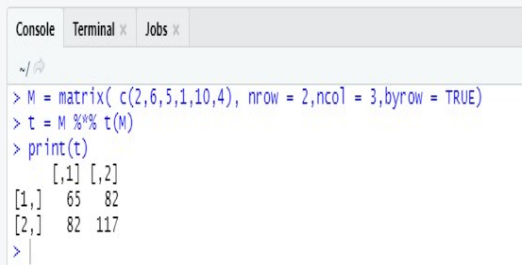
## UNIT 3 : Introduction to R and working with Data

5.	>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	<div> <div>ConsoleTerminal ×Jobs ×</div> <div>~/ ↗</div> <pre> &gt; v &lt;- c(5,55.23,6,99) &gt; t &lt;- c(8,22.5,4,99) &gt; print(v&gt;=t) [1] FALSE TRUE TRUE TRUE &gt;   </pre> </div>
6.	!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	<div> <div>ConsoleTerminal ×Jobs ×</div> <div>~/ ↗</div> <pre> &gt; v &lt;- c(5,55.23,6,99) &gt; t &lt;- c(8,22.5,4,99) &gt; print(v!=t) [1] TRUE TRUE TRUE FALSE &gt;   </pre> </div>

### Miscellaneous Operator

No	Operator	Use	Example
1.	:	Colon operator. It creates the series of numbers in sequence for a vector.	<div> <div>ConsoleTerminal ×Jobs ×</div> <div>~/ ↗</div> <pre> &gt; var&lt;-matrix(c(101:106),nrow=3,ncol=2) &gt; print(var)   [,1] [,2] [1,] 101 104 [2,] 102 105 [3,] 103 106 &gt;   </pre> </div>
2.	%in%	This operator is used to identify if an element belongs to a vector.	<div> <div>ConsoleTerminal ×Jobs ×</div> <div>~/ ↗</div> <pre> &gt; x&lt;-45 &gt; y&lt;-266 &gt; z&lt;-1:50 &gt; print(x %in% z) [1] TRUE &gt; print(y %in% z) [1] FALSE &gt;   </pre> </div>

## UNIT 3 : Introduction to R and working with Data

3.	%*%	This operator is used to multiply a matrix with its transpose.	 <pre>&gt; M = matrix( c(2,6,5,1,10,4), nrow = 2, ncol = 3, byrow = TRUE) &gt; t = M %*% t(M) &gt; print(t)       [,1] [,2] [1,]   65   82 [2,]   82  117 &gt;  </pre>
----	-----	--	--

### Decision Making

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

- If statement : An if statement consists of a Boolean expression followed by one or more statements.
- If...else statement : An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

switch statement : A switch statement allows a variable to be tested for equality against a list of values.

### Loops

#### 1. Repeat

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

<pre>repeat {   Statements to be executed   If(expression)   {     break   } }</pre>
--

## UNIT 3 : Introduction to R and working with Data

```
Console Terminal x Jobs x
~/
> x<-1
> repeat
+ {
+   if(x>10)
+     break
+   print(x)
+   x<-x+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
> |
```

### 2. While Loop

The While loop executes the same code again and again until a stop condition is met.

```
initialization
While (Condition)
{
Statements
}
```

```
Console Terminal x Jobs x
~/
> x<-1
> while (x<=10) {
+   print(x)
+   x<-x+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
> |
```



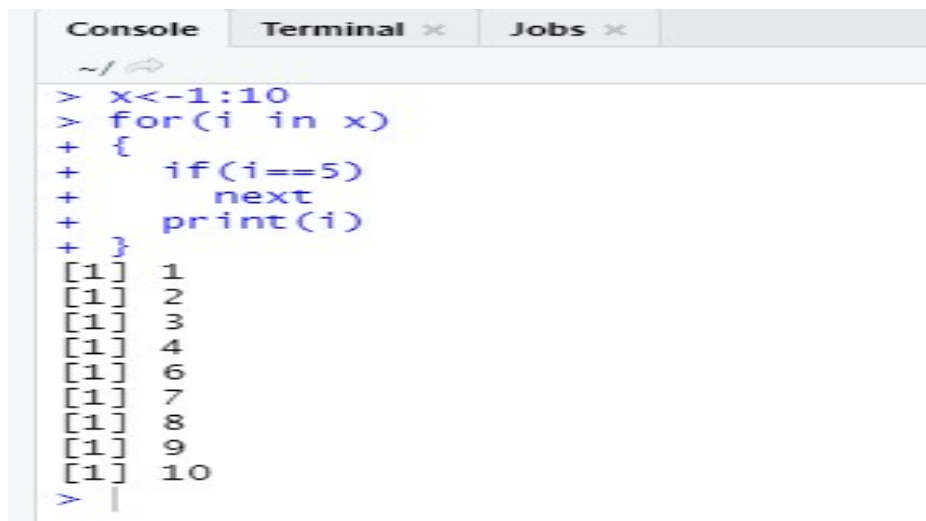
## UNIT 3 : Introduction to R and working with Data

### 3. For Loop

A For loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times

Syntax:

```
for (value in vector)
{
  Statement(s)
}
```

A screenshot of an R console window with tabs for 'Console', 'Terminal', and 'Jobs'. The console shows the following R code and its output:

```
> x<-1:10
> for(i in x)
+ {
+   if(i==5)
+     next
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
> |
```

### Functions:

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

Syntax:

```
function_name <- function(arg_1, arg_2, ...)
{
  Function body
  ....
  return ()
}
```

## UNIT 3 : Introduction to R and working with Data

```
~/
> SumOfNum<-function(num)
+ {
+   sum<-0
+   while(num!=0)
+   {
+     sum<-sum+num%%10
+     num<-num/10
+   }
+   return(sum)
+ }
> num = as.integer(readline(prompt = "Enter a number: "))
Enter a number: 2564
> result<-SumOfNum(num)
> cat("Sum of Number ",num,"is : ",result)
Sum of Number  2564 is :  18.88889
>
```

## R Data Structures

### Vectors, Matrix, Array, Data Frame

#### 1. Vectors

Vectors are a sequence of data elements of the same basic datatype. Atomic vectors also termed as a five classes of vectors.

To combine the list of items to a vector, use the `c()` function and separate the items by a comma.

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

# Print fruits
fruits
```

```
# Vector of numerical values
numbers <- c(1, 2, 3)

# Print numbers
numbers
```

## UNIT 3 : Introduction to R and working with Data

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

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

numbers
```

```
# Vector of logical values
values <- c(TRUE, FALSE, TRUE, FALSE)

values
```

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

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

length(fruits)
```

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

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

sort(fruits) # Sort a string
sort(numbers) # Sort numbers
```

The vector items can be accessed by referring to its index number inside brackets `[]`. The first item has index 1, the second item has index 2, and so on:

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

# Access the first item (banana)
fruits[1]
```

access multiple elements by referring to different index positions with the `c()` function:

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

# Access the first and third item (banana and orange)
fruits[c(1, 3)]
```

## UNIT 3 : Introduction to R and working with Data

use negative index numbers to access all items except the ones specified:

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")  
  
# Access all items except for the first item  
fruits[c(-1)]
```

### 2. List

A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable.

To create a list, use the `list()` function:

```
# List of strings  
thislist <- list("apple", "banana", "cherry")  
  
# Print the list  
thislist
```

List can be accessed like vectors.

Find out if a specified item is present in a list, use the `%in%` operator:

```
thislist <- list("apple", "banana", "cherry")  
  
"apple" %in% thislist
```

To add an item to the end of the list, use the `append()` function:

```
thislist <- list("apple", "banana", "cherry")  
  
append(thislist, "orange")
```

You can specify a range of indexes by specifying where to start and where to end the range, by using the `:` operator.

```
thislist <- list("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
  
(thislist)[2:5]
```

## UNIT 3 : Introduction to R and working with Data

Loop through the list items by using a for loop:

```
thislist <- list("apple", "banana", "cherry")

for (x in thislist) {
  print(x)
}
```

The c() function, which combines two elements together:

```
list1 <- list("a", "b", "c")
list2 <- list(1,2,3)
list3 <- c(list1,list2)

list3
```

### 3. Matrix

A matrix is a two dimensional data set with columns and rows.

A column is a vertical representation of data, while a row is a horizontal representation of data.

A matrix can be created with the matrix() function. Specify the nrow and ncol parameters to get the amount of rows and columns:

```
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix
thismatrix
```

Access the items by using [ ] brackets. The first number "1" in the bracket specifies the row-position, while the second number "2" specifies the column-position:

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

thismatrix[1, 2]
```

## UNIT 3 : Introduction to R and working with Data

The `dim()` function to find the number of rows and columns in a Matrix.

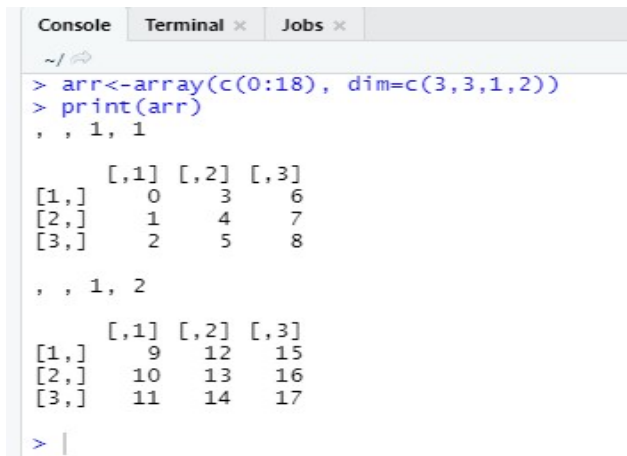
### 4. Arrays

Compared to matrices, arrays can have more than two dimensions.

We can use the `array()` function to create an array, and the `dim` parameter to specify the dimensions. **`array(data, dim, dimnames)`**

```
# An array with one dimension with values ranging from 1 to 24
thisarray <- c(1:24)
thisarray

# An array with more than one dimension
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray
```



The screenshot shows an R console window with the following code and output:

```
> arr<-array(c(0:18), dim=c(3,3,1,2))
> print(arr)
, , 1, 1
      [,1] [,2] [,3]
[1,]    0    3    6
[2,]    1    4    7
[3,]    2    5    8

, , 1, 2
      [,1] [,2] [,3]
[1,]    9   12   15
[2,]   10   13   16
[3,]   11   14   17
> |
```

### 5. DataFrames

Data Frames are data displayed in a format as a table.

Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.

Use the `data.frame()` function to create a data frame.

## UNIT 3 : Introduction to R and working with Data

```
# 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
```

	Training	Pulse	Duration
1	Strength	100	60
2	Stamina	150	30
3	Other	120	45

use single brackets [ ], double brackets [[ ]] or \$ to access columns from a data frame:

```
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
```

dim(), nrow() and ncol() functions are used to find the dimension, number of rows and number of columns of data frame respectively.

### **Importing data into R from different file formats (CSV, Excel, etc.)**

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

### **To read data from a csv file and then write data into a csv**

Getting and Setting the Working Directory

## UNIT 3 : Introduction to R and working with Data

```
# Get and print current working directory.
print(getwd())

# Set current working directory.
setwd("/web/com")

# Get and print current working directory.
print(getwd())
```

now create or input a csv file.

Reading a CSV File

```
data <- read.csv("csv file name")
print(data)
```

### Analyzing the CSV File

By default the read.csv() function gives the output as a data frame.

```
data <- read.csv("csv file name")

print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

Once we read data in a data frame, we can apply all the functions applicable to data frame.

### Writing into a CSV File

R can create csv file from existing data frame. The write.csv() function is used to create the csv file.

```
# Create a data frame.
data <- read.csv("input.csv")

# Write data into a new file.
write.csv(data, "output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```



## UNIT 3 : Introduction to R and working with Data

### Reading the Excel File

The xlsx is read by using the **read.xlsx()** function. The result is stored as a data frame in the R environment.

```
# Read the first worksheet in the file xlsx.  
data <- read.xlsx("xlsx filename", sheetIndex = 1)  
print(data)
```