UNIT 13 BASICS OF RPROGRAMMING

Structure **Page Nos.**

13.0 INTRODUCTION

This unit covers the fundamental concepts of R programming. The unit familiarises with the environment of R and covers the details of the global environment. It further discusses the various types of data that is associated with every variable to reserve some memory space and store values in R. The unit discusses the various types of the data objects known as factors and the types of operators used in R programming. The unit also explains the important elements of decision making, the general form of a typical decision-making structures and the loops and functions. R's basic data structures including vector, strings, lists, frames, matrices and arrays would also be discussed.

13.1 OBJECTIVES

After going through this Unit, you will be able to:

- explain about the environment of R, the global environment and their elements;
- explain and distinguish between the data types and assign them to variables;
- explain about the different types of operators and the factors;
- explain the basics of decision making, the structure and the types of loops;
- explain about the function- their components and the types;
- explain the data structures including vector, strings, lists, frames, matrices, and arrays.

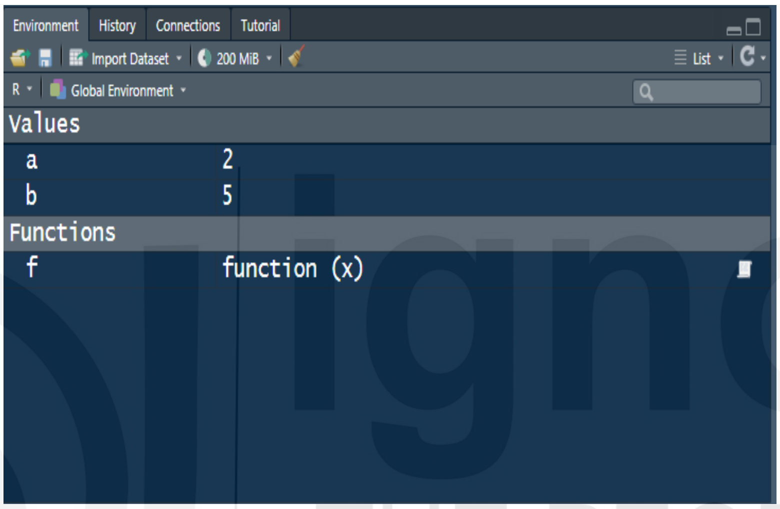
13.2 ENVIRONMENT OF R

R Programming language has been designed for statistical analysis of data. It also has a very good support for graphical representation of data. It has a vast set of commands. In this Block, we will cover some of the essential component of R programming, which would be useful for you for the purpose of data analysis. We will not be covering all aspects of this programming language; therefore, you may refer to the further readings for more details.

The discussion on R programming will be in the context of R-Studio, which is an open-source software. You may try various commands listed in this unit to facilitate your learning. The first important concept of R is its environment, which is discussed next.

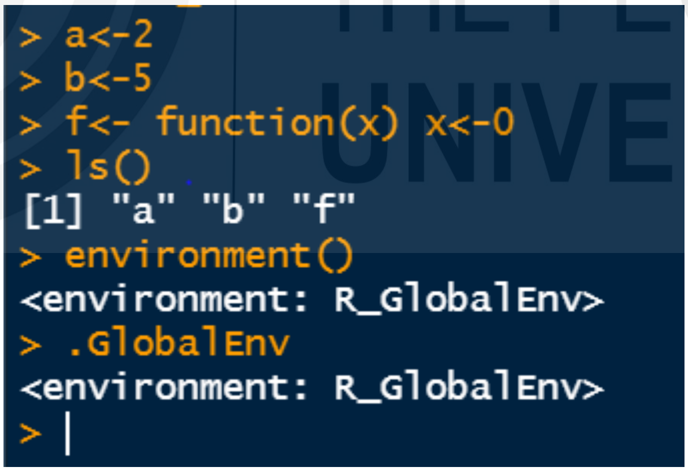
Environment can be thought of as a virtual space having collection of objects (variables, functions etc.) An environment is created when you first hit the R interpreter.

The top level environment present at R command prompt is the global environment known as R_GlobalEnv, it can also be referred as .GlobalEnv. You can use ls() command to know what variables/ functions are defined in the working environment. You can even check it in the Environment section of R Studio.



```
Values
 a        2
 b        5
Functions
 f        function (x)
```

Figure 5.1: Environment with a variable in RStudio



```
> a<-2
> b<-5
> f<- function(x) x<-0
> ls()
[1] "a" "b" "f"
> environment()
<environment: R_GlobalEnv>
> .GlobalEnv
<environment: R_GlobalEnv>
> |
```

Figure 5.2: Variables in Global Environment

In Figure 5.2, variables a, b and f are in R_GlobalEnv. Notice that x (as an argument to the function) is not in the global environment. When you define a function, a new environment is created. In Figure 5.1, a function f created a new environment inside the Global environment.

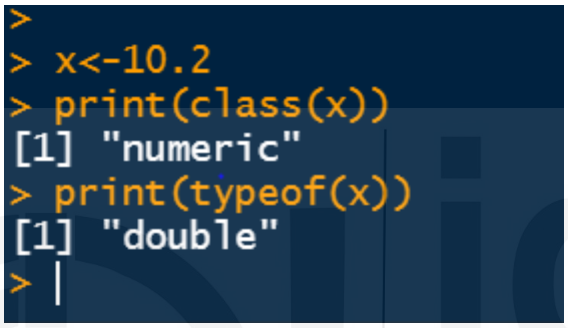
13.3 DATA TYPES, VARIABLES, OPERATORS, FACTORS

Every variable in R has an associated data type, which is known as the reserved memory. This reserved memory is needed for storing the values. Given below is a list of basic data types available in R programming:

DATA TYPE	Allowable Values
Integer	Values from the Set of Integers, Z
Numeric	Values from the Set of Real Numbers, R
Complex	Values from the Set of Complex numbers, C
Logical	Only allowable values are True ; False
Character	Possible values are -“x”, “@”, “1”, etc.

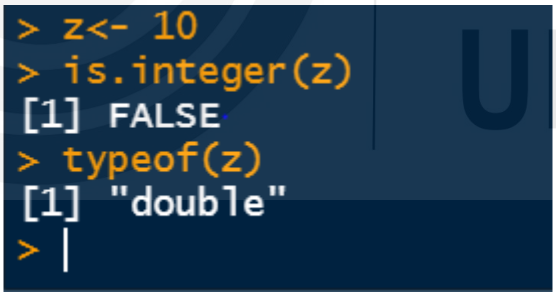
Table 1: Basic Data Types

Numeric Datatype:
Decimal values are known to be numeric in R and is default datatype for any number in R.



```
>
> x<-10.2
> print(class(x))
[1] "numeric"
> print(typeof(x))
[1] "double"
> |
```

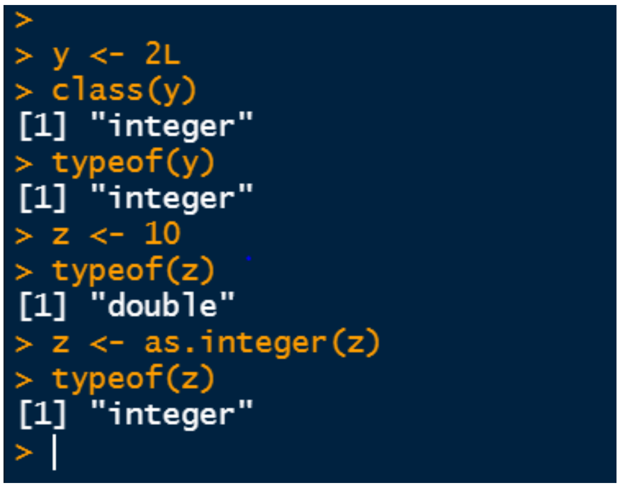
Whenever a number is stored in R, it gets converted into decimal type with at least 2 decimal points or the “double” value. So, if you enter a normal integer value also, for example 10, then R interpreter will convert it into double i.e. 10.00. You can even confirm this by checking the type of the variable, as given below:



```
> z<- 10
> is.integer(z)
[1] FALSE
> typeof(z)
[1] "double"
> |
```

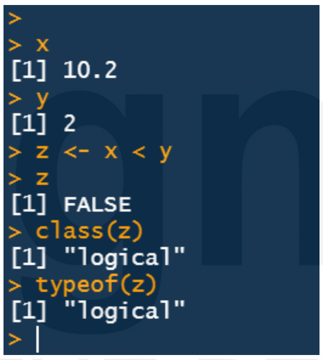
is.integer() function returning FALSE confirms that the variable z is converted into double or the decimal type.

Integer Datatype:
R supports integer data type, you can create an integer by suffixing “L” to denote that particular variable as integer as well as convert a value to an integer by passing the variable to as.integer() function.



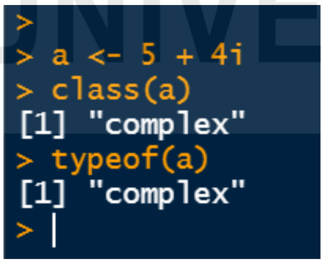
```
>
> y <- 2L
> class(y)
[1] "integer"
> typeof(y)
[1] "integer"
> z <- 10
> typeof(z)
[1] "double"
> z <- as.integer(z)
> typeof(z)
[1] "integer"
> |
```

Logical Datatype:
R has a logical datatype which returns value as either TRUE or FALSE. It is usually used while comparing two variables in a condition.



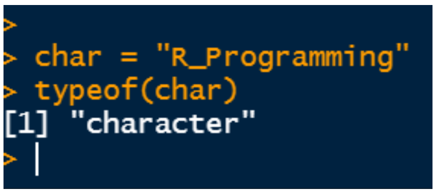
```
>
> x
[1] 10.2
> y
[1] 2
> z <- x < y
> z
[1] FALSE
> class(z)
[1] "logical"
> typeof(z)
[1] "logical"
> |
```

Complex Datatype:
Complex data types are also supported in R. These datatype includes the set of all complex numbers.



```
>
> a <- 5 + 4i
> class(a)
[1] "complex"
> typeof(a)
[1] "complex"
> |
```

Character Datatype:
R supports character datatype which includes alphabets and special characters. We need to include the value of the character type inside single or double inverted commas.



```
>
> char = "R_Programming"
> typeof(char)
[1] "character"
> |
```

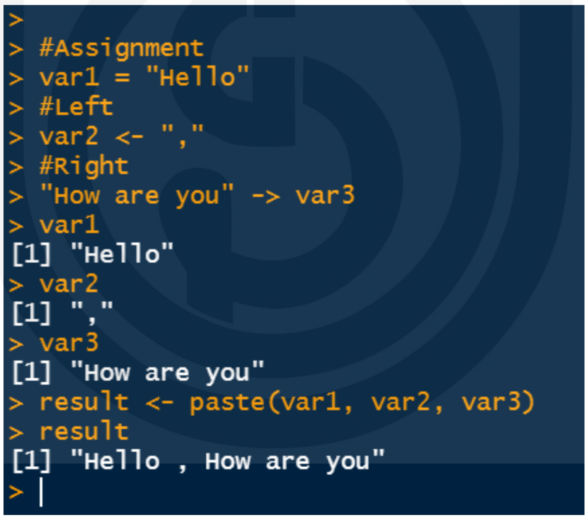
VARIABLES:

A variable as discussed in the previous section, allocates a memory space and stores the values, which can be manipulated. A valid variable name consists of letters, numbers and dot or underline characters

Variable Name	Valid	Reason
var_name1.	Valid	Contains letters, number, dot and underscore
1var_name	Invalid	Starting with a number
Var_name@	Invalid	Has special character (@). Only dot and underscore is allowed.
.var_name, var.name	Valid	Can start with a dot, which is followed by an alphabet.
_var_name	Invalid	Should not start with underscore.
.2var_name	Invalid	Dot is followed by a number and hence invalid.

Variables Assignment: Variables can be assigned in multiple ways –

- Assignment (=): var1 = "Hello"
- Left (←): var2 ← ", "
- Right (→): "How are you" → var3



```
>
> #Assignment
> var1 = "Hello"
> #Left
> var2 <- ","
> #Right
> "How are you" -> var3
> var1
[1] "Hello"
> var2
[1] ","
> var3
[1] "How are you"
> result <- paste(var1, var2, var3)
> result
[1] "Hello , How are you"
> |
```

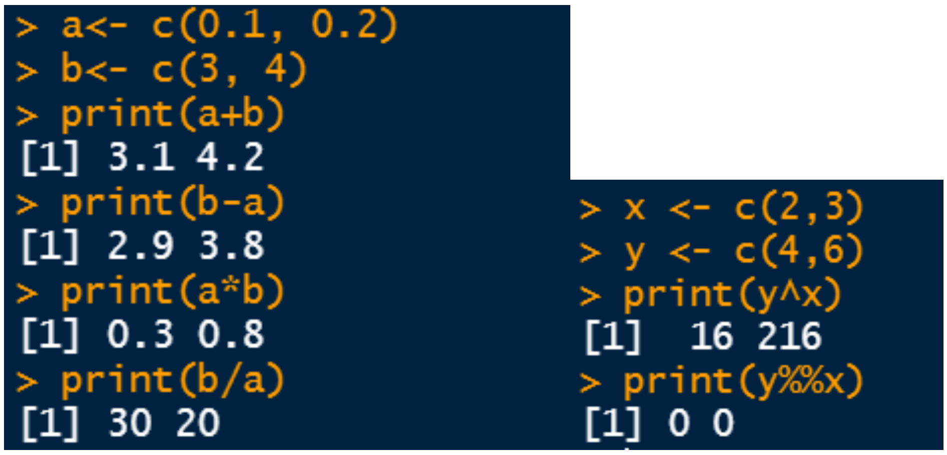
OPERATORS:

As the case with other programming languages, R also supports assignment, arithmetic, relational and logical operators. The logical operators of R include element by element operations. In addition, several other operators are supported by R, as explained in this section.

Arithmetic Operators:

- Addition (+): The value at the corresponding positions in the vectors are added. Please note the difference with C programming, as you are adding a complete vector using a single operator.
- Subtraction (-): The value at the corresponding positions are subtracted. Once again please note that single operator performs the task of subtracting elements of two vectors.
- Multiplication (*): The value at the corresponding positions are multiplied.

- Division (/): The value at the corresponding positions are divided.
- Power (^): The first vector is raised to the exponent (power) of the second.
- Modulo (%%): The remainder after dividing the two will be returned.

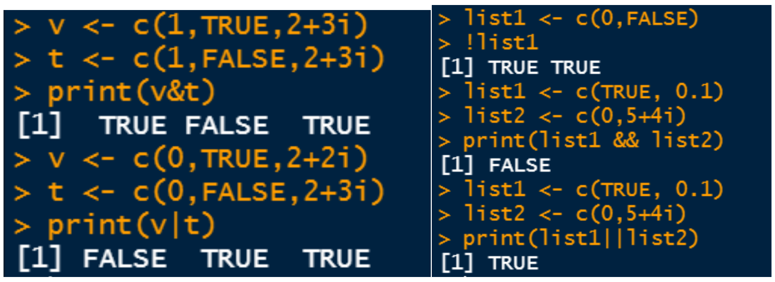


```
> a<- c(0.1, 0.2)
> b<- c(3, 4)
> print(a+b)
[1] 3.1 4.2
> print(b-a)
[1] 2.9 3.8
> print(a*b)
[1] 0.3 0.8
> print(b/a)
[1] 30 20
```

```
> x <- c(2,3)
> y <- c(4,6)
> print(y^x)
[1]  16 216
> print(y%%x)
[1] 0 0
```

Logical Operators:

- Element-wise Logical AND Operator (&): If both the corresponding operands are true, then this operator returns the Boolean value TRUE for that element. Please note the difference with C programming, in which it is a bitwise AND operator, whereas in R it is an element wise AND operator.
- Element-wise Logical OR Operator (|): If either of the corresponding operands are TRUE, then this operator returns the Boolean value TRUE for that element.
- Not Operator (!): This is a unary operator that is used to negate the operand.
- Logical AND Operator (&&): If the first element of both the operand are TRUE, then this operator returns the Boolean value TRUE.
  Logical OR Operator (||): If either of the first elements of the operands are true, then this operator returns Boolean value TRUE.



```
> v <- c(1,TRUE,2+3i)
> t <- c(1,FALSE,2+3i)
> print(v&t)
[1]  TRUE FALSE  TRUE
> v <- c(0,TRUE,2+2i)
> t <- c(0,FALSE,2+3i)
> print(v|t)
[1] FALSE  TRUE  TRUE
```

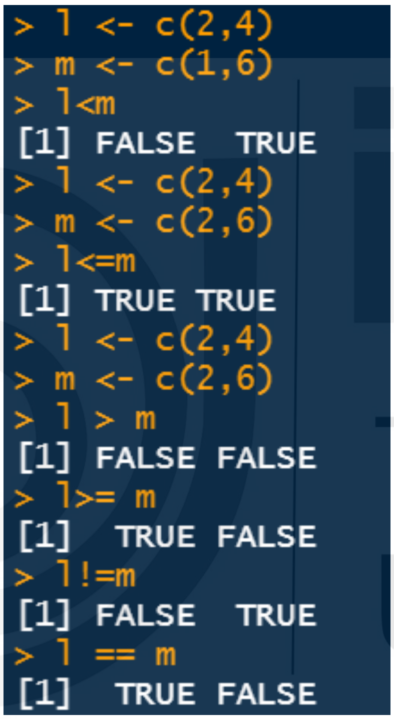
```
> list1 <- c(0,FALSE)
> !list1
[1] TRUE TRUE
> list1 <- c(TRUE, 0.1)
> list2 <- c(0,5+4i)
> print(list1 && list2)
[1] FALSE
> list1 <- c(TRUE, 0.1)
> list2 <- c(0,5+4i)
> print(list1||list2)
[1] TRUE
```

Relational Operators:

The relational operators can take scalar or vector operands. In case of vector operands comparison is done element by element and a vector of TRUE/FALSE values is returned.

- Less than (<): If an element of the first operand (scalar or vector) is less than that the corresponding element of the second operand, then this operator returns Boolean value TRUE.

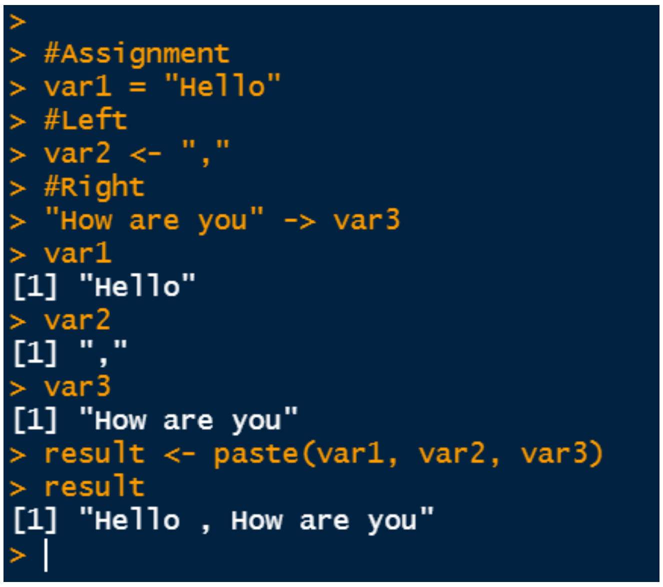
- Less than Equal to (<=): If every element in the first operand or vector is less than or equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Greater than (>): If every element in the first operand or vector is greater than that the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Greater than (>=): If every element in the first operand or vector is greater than or equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Not equal to (!=): If every element in the first operand or vector is not equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Equal to (==): If every element in the first operand or vector is equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.



```
> l <- c(2,4)
> m <- c(1,6)
> l<m
[1] FALSE  TRUE
> l <- c(2,4)
> m <- c(2,6)
> l<=m
[1] TRUE TRUE
> l <- c(2,4)
> m <- c(2,6)
> l > m
[1] FALSE FALSE
> l>= m
[1]  TRUE FALSE
> l!=m
[1] FALSE  TRUE
> l == m
[1]  TRUE FALSE
```

Assignment Operators:

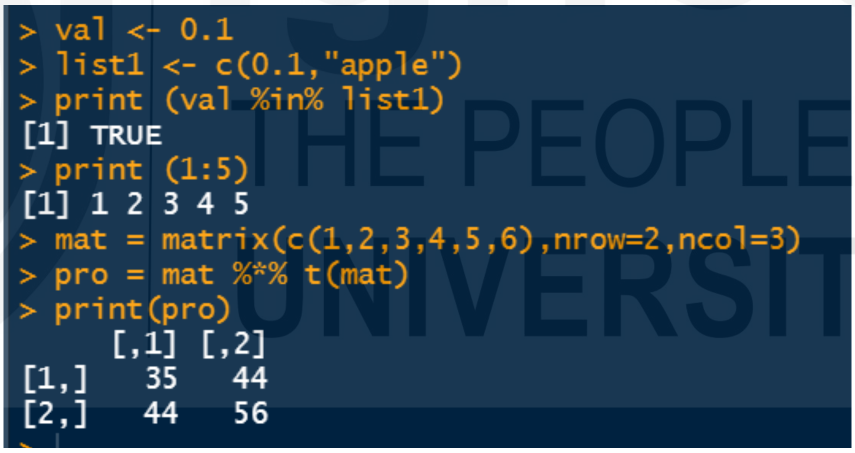
- Left Assignment (← or <<-or =): Used for assigning value to a vector.
- Right Assignment (-> or ->>): Used for assigning value to a vector.



```
>
> #Assignment
> var1 = "Hello"
> #Left
> var2 <- ","
> #Right
> "How are you" -> var3
> var1
[1] "Hello"
> var2
[1] ","
> var3
[1] "How are you"
> result <- paste(var1, var2, var3)
> result
[1] "Hello , How are you"
> |
```

Miscellaneous Operators:

- %in% operator: It determines whether a data element is contained in a list and returns a Boolean value TRUE if the element is found to exist.
- Colon(:) Operator: It prints a list of elements from before the colon to after the colon.
- %*% Operator: It helps in multiplying a matrix with its transpose.



```
> val <- 0.1
> list1 <- c(0.1,"apple")
> print (val %in% list1)
[1] TRUE
> print (1:5)
[1] 1 2 3 4 5
> mat = matrix(c(1,2,3,4,5,6),nrow=2,ncol=3)
> pro = mat %*% t(mat)
> print(pro)
     [,1] [,2]
[1,]   35   44
[2,]   44   56
```

FACTORS:

Factors are the data objects are used for categorizing and further storing the data as levels. They store both, strings and integer values. Factors are useful in the columns that have a limited number of unique values also known to be categorical variable. They are useful in data analysis for statistical modelling. For example, a categorical variable employment types – (Unemployed, Self-Employed, Salaried, Others) can be represented using factors. More details on factors can be obtained from the further readings.

Check Your Progress 1

1. What are various Operators in R?

……………………………………………………………………………

……………………………………………………………………………

2. What does %*% operator do?

………………………………………………………………………………….

…………………………………………………………………………………

3. Is .5Var a valid variable name? Give reason in support of your answer.

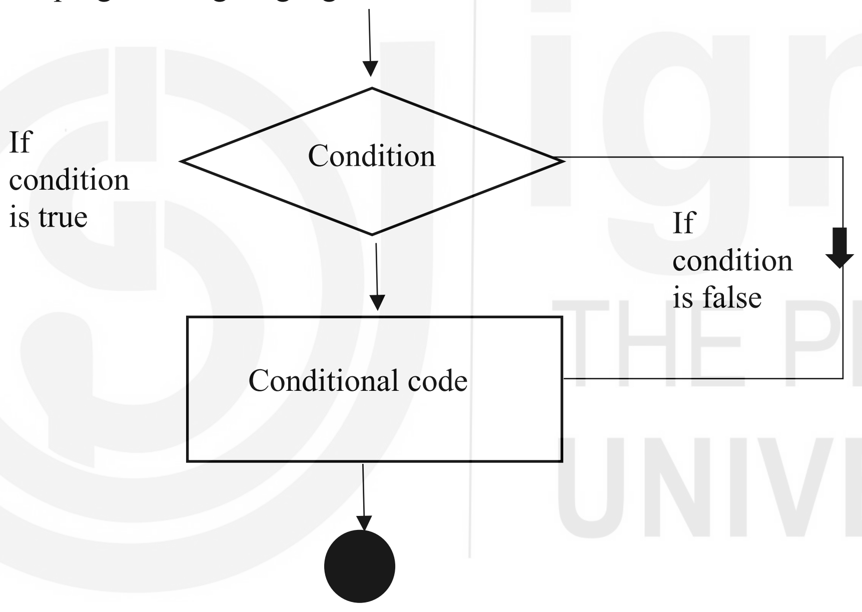
…………………………………………………………………………………

…………………………………………………………………………………

13.4 DECISION MAKING, LOOPS, FUNCTIONS

Decision making requires the programmer to specify one or more conditions which will be evaluated or tested by the program, along with the statements to be executed if the condition is determined to be true, and optional statements to be executed if the condition is determined to be false.

Given below is the general form of a typical decision making structure found in most of the programming languages–



The format of if statement in R is as follows:

***if* (*conditional statement*, may include relational and logical operator) {**

R statements to be executed, if the *conditional statement* is true

}

***else* {**

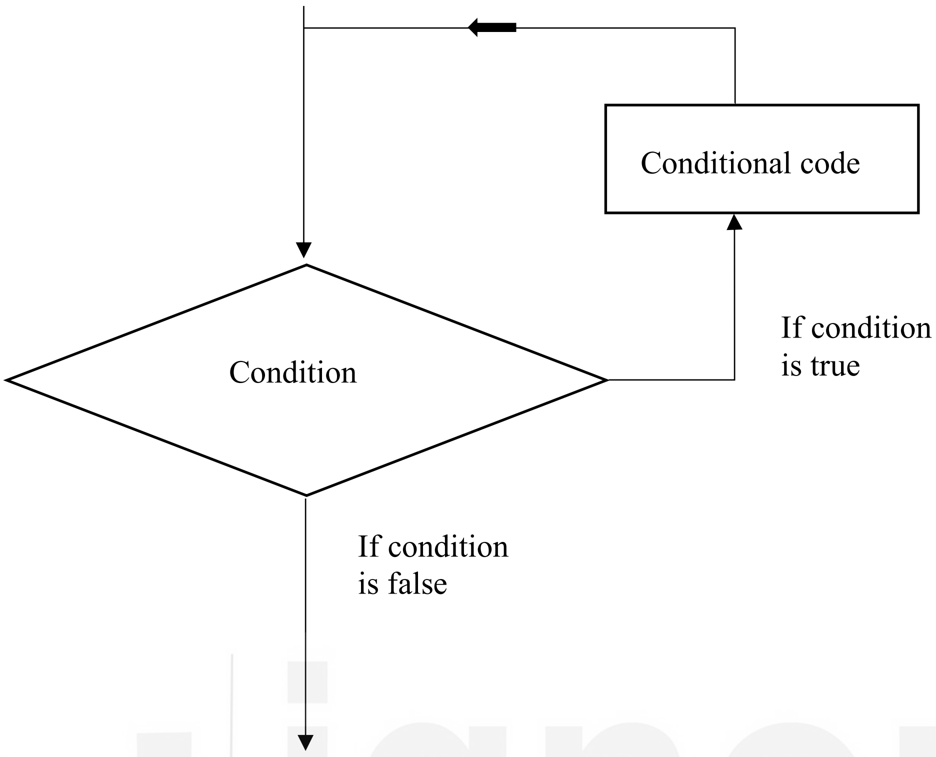
R statements to be executed, if the *conditional statement* is FALSE

}

You may use *else if* instead of *else*

LOOPS:

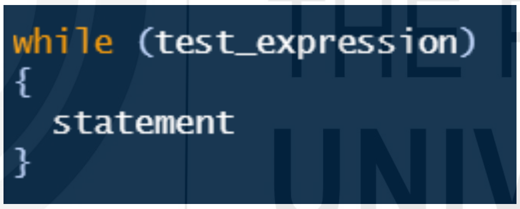
A loop is defined as a situation where we need to execute a block of code several number of times. In the case of loops, the statements are executed sequentially.



Loop Type and Description:

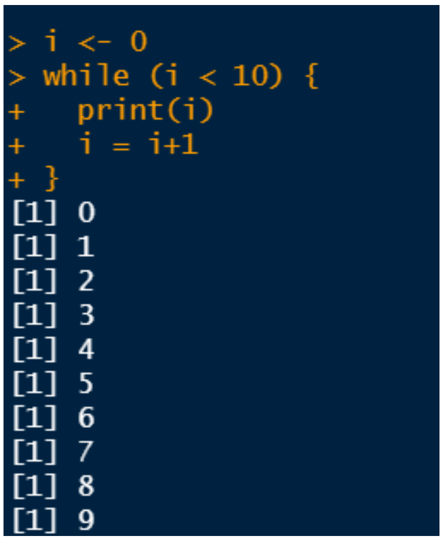
- Repeat loop: Executes sequence of statements multiple times.
- While loop: Repeat a given statement while the given condition is true, executes before executing the loop body.

  Syntax:



```
while (test_expression)
{
  statement
}
```

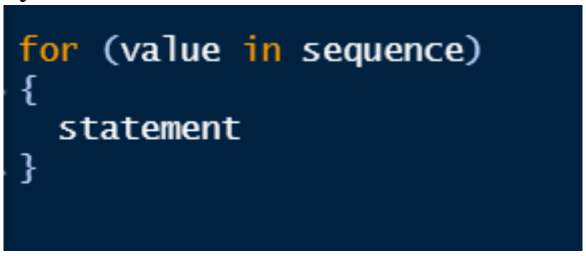
Example:



```
> i <- 0
> while (i < 10) {
+   print(i)
+   i = i+1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

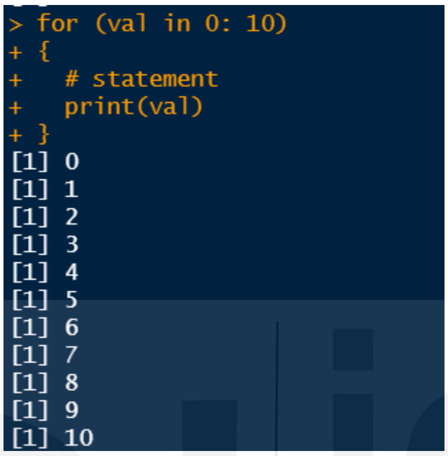
- For loop: Like while statement, executes the test condition at the end of the loop body.

Syntax:



```
for (value in sequence)
{
  statement
}
```

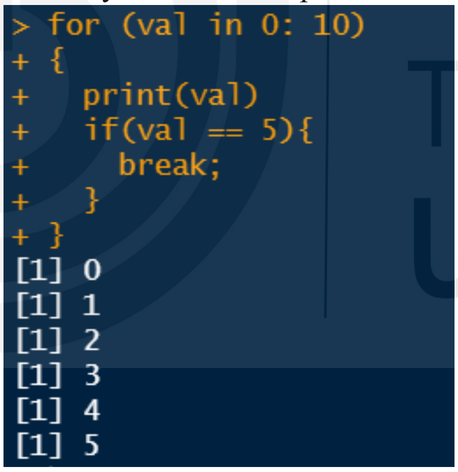
Example:



```
> for (val in 0: 10)
+ {
+   # statement
+   print(val)
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Loop Control Statements:

- Break Statements: Terminates the loop statement and execute the statements immediately below the loop.



```
> for (val in 0: 10)
+ {
+   print(val)
+   if(val == 5){
+     break;
+   }
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

FUNCTIONS:

A function refers to a set of instructions that is required to execute a command to achieve a task in R. There are several built-in functions available in R. Further, users may create a function basis their requirements.

Definition:

A function can be defined as:

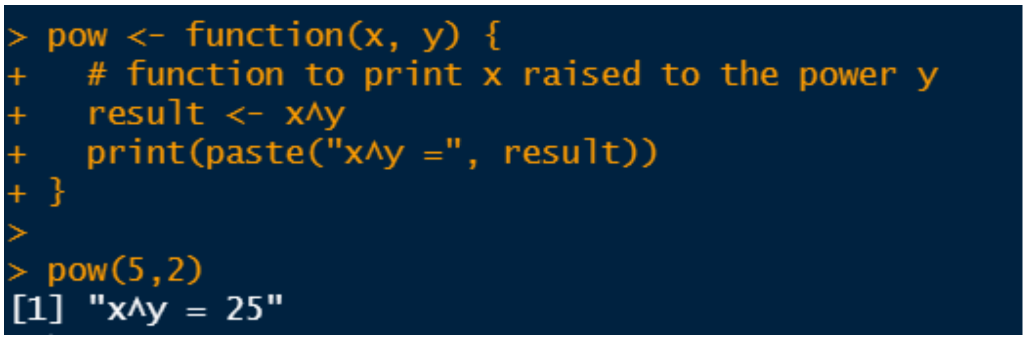
```
function_name<- function(arg_1, arg_2, ...) {
  Function body
}
```

Function Components

- Function Name: Actual name of the function.

- Arguments: Passed when the function is invoked. They are optional.
- Function Body: statements that define the logic of the function.
- Return value: last expression of the function to be executed.

Built-in function: Built in functions are the functions already written and is accessible just by calling the function name. Some examples are seq(), mean(), min(), max(), sqrt(), paste() and many more.



```
> pow <- function(x, y) {
+   # function to print x raised to the power y
+   result <- x^y
+   print(paste("x^y =", result))
+ }
>
> pow(5,2)
[1] "x^y = 25"
```

13.5 Data Structures in R

R's basic data structures include Vector, Strings, Lists, Frames, Matrices and Arrays.

13.5.1 Strings and Vectors

Vectors:

A vector is a one-dimensional array of data elements that have same data type. The most basic data structure are the Vectors, which supports logical, integer, double, complex, character datatypes.

Strings:
Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.

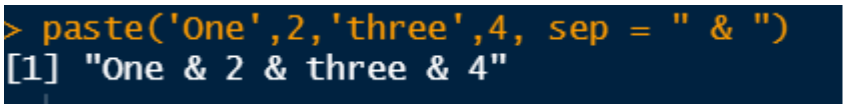
Rules Applied in String Construction

- The quotes at the beginning and end of a string should be either both double quotes or both single quote. They cannot be mixed.
- Double quotes can be inserted into a string starting and ending with single quote.
- Single quote can be inserted into a string starting and ending with double quotes.
- Double quotes cannot be inserted into a string starting and ending with double quotes.
- Single quote cannot be inserted into a string starting and ending with single quote.

Length of String: The length of strings tells the number of characters in a string. The inbuilt function nchar() or function str_length() of the stringr package can be used to get the length of the string.

String Manipulations:

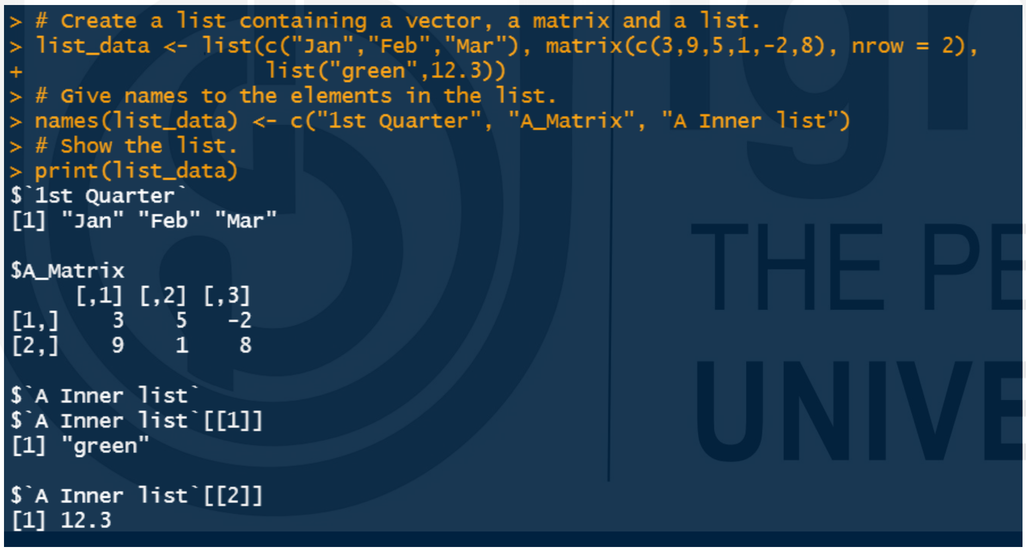
- Substring: Accessing the different portions of the strings. The 2 inbuilt functions present for this is substr() or substring() to extract the sub-strings.
- Case Conversion: The characters of the string can be converted to upper or the lower case by using toupper() or tolower().
- Concatenation: The strings in R can be combined by using the paste() function. It can concatenate any number of strings together. For example, paste(..., sep = " ", collapse = NULL)where x is vector having values, sep: is a separator symbol that is used to separate elements& collapse gives value to collapse.



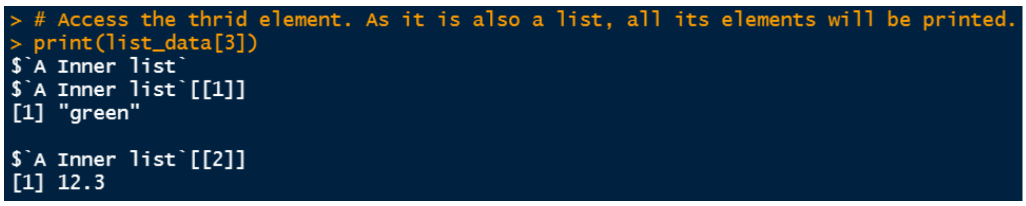
```
> paste('One',2,'three',4, sep = " & ")
[1] "One & 2 & three & 4"
```

13.5.2 Lists

Lists are the objects in R that contains different types of objects within itself like number, string, vectors or even another list, matrix or any function as its element It is created by calling list() function.



```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+                  list("green",12.3))
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
> # Show the list.
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3
```

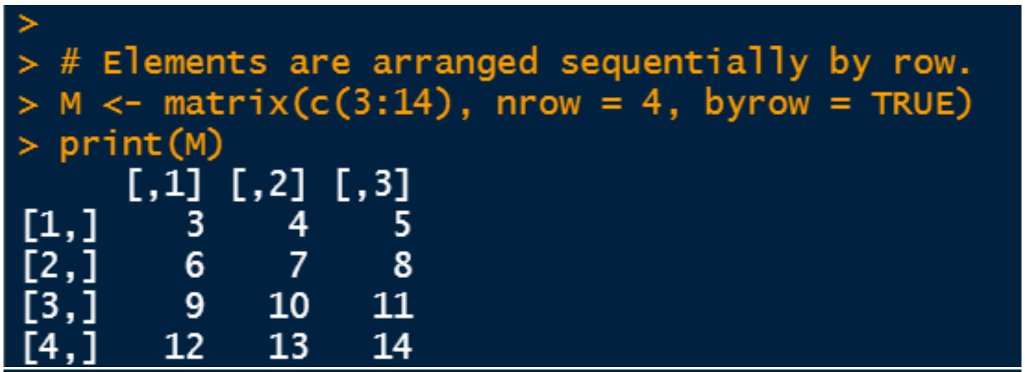


```
> # Access the thrid element. As it is also a list, all its elements will be printed.
> print(list_data[3])
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3
```

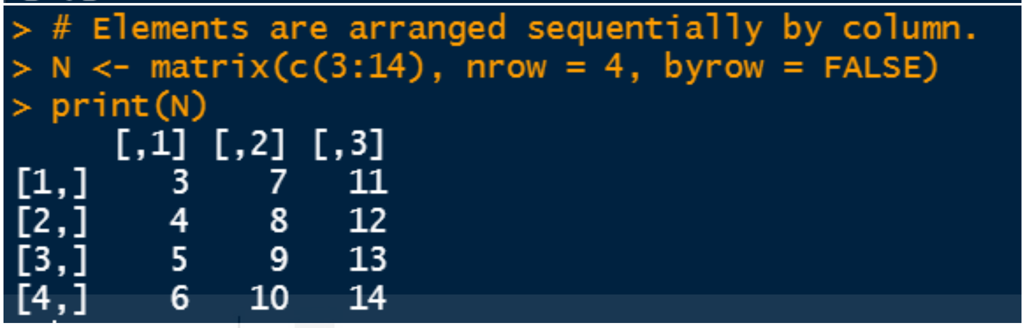
13.5.3 Matrices, Arrays and Frames

Matrices are R objects which are arranged in 2-D layout. They contain element of same type. The basic syntax of creating a matrix in R is:

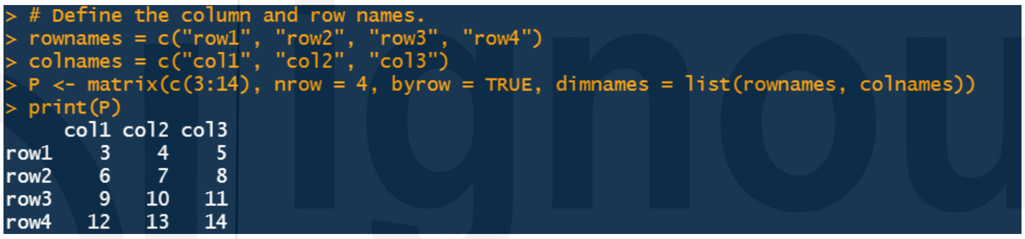
matrix(data, nrow, ncol, byrow, dimnames), where *data* is the name of input vector, *nrow* is no of rows, *ncol* is no of columns, *byrow* is to specify either row matrix or column matrix and *dimname*is the name assigned to rows and columns.



```
>
> # Elements are arranged sequentially by row.
> M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
> print(M)
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
```

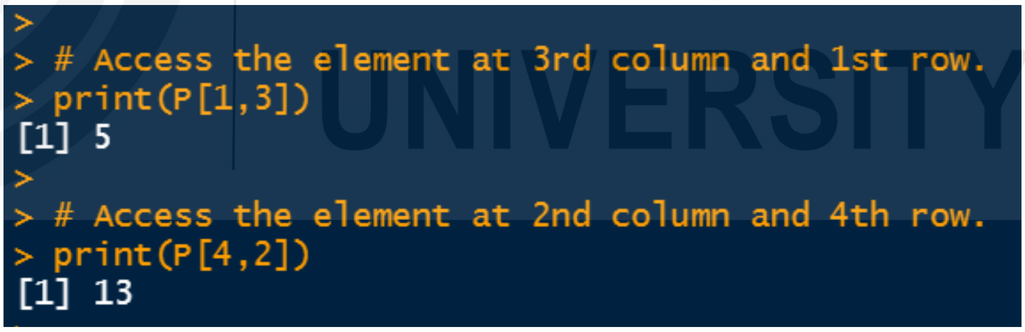


```
> # Elements are arranged sequentially by column.
> N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
> print(N)
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
```



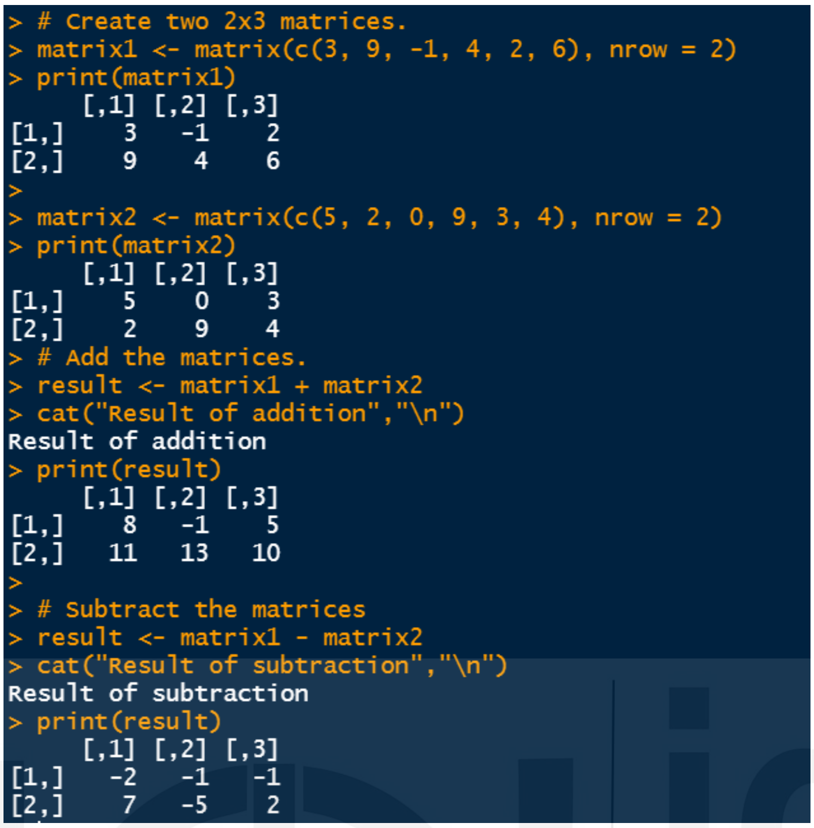
```
> # Define the column and row names.
> rownames = c("row1", "row2", "row3", "row4")
> colnames = c("col1", "col2", "col3")
> P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
> print(P)
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

Accessing the elements of the matrix: Elements of a matrix can be accessed by specifying the row and column number.

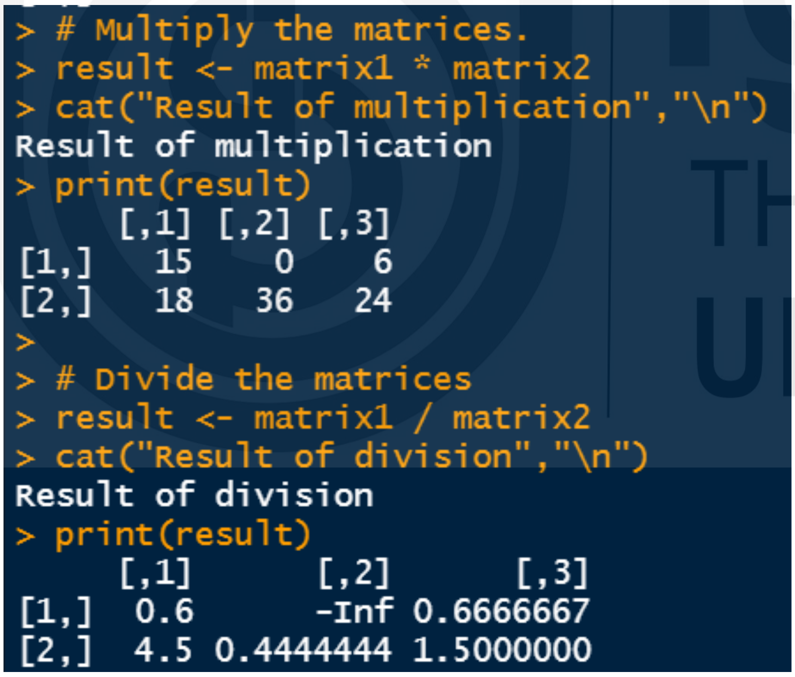


```
>
> # Access the element at 3rd column and 1st row.
> print(P[1,3])
[1] 5
>
> # Access the element at 2nd column and 4th row.
> print(P[4,2])
[1] 13
```

Matrix Manipulations:
Mathematical operations can be performed on the matrix like addition, subtraction, multiplication and division. You may please note that matrix division is not defined mathematically, but in R each element of a matrix is divided by the corresponding element of other matrix.



```
> # Create two 2x3 matrices.
> matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
> print(matrix1)
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
>
> matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
> print(matrix2)
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
> # Add the matrices.
> result <- matrix1 + matrix2
> cat("Result of addition","\n")
Result of addition
> print(result)
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
>
> # Subtract the matrices
> result <- matrix1 - matrix2
> cat("Result of subtraction","\n")
Result of subtraction
> print(result)
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
```

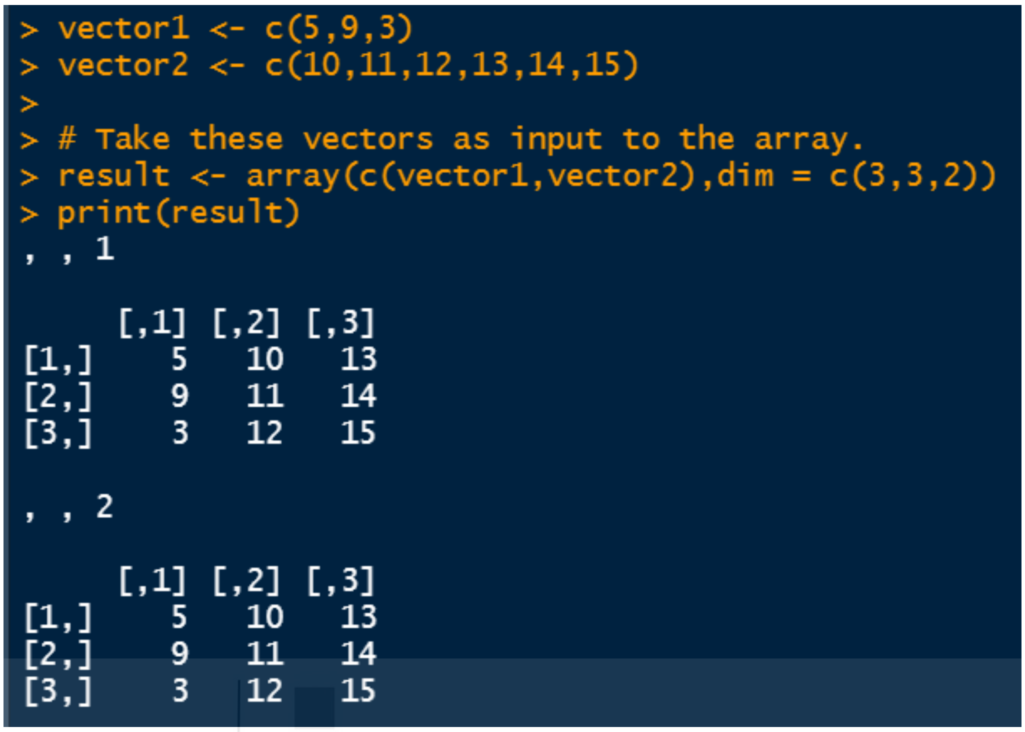


```
> # Multiply the matrices.
> result <- matrix1 * matrix2
> cat("Result of multiplication","\n")
Result of multiplication
> print(result)
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24
>
> # Divide the matrices
> result <- matrix1 / matrix2
> cat("Result of division","\n")
Result of division
> print(result)
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Arrays:

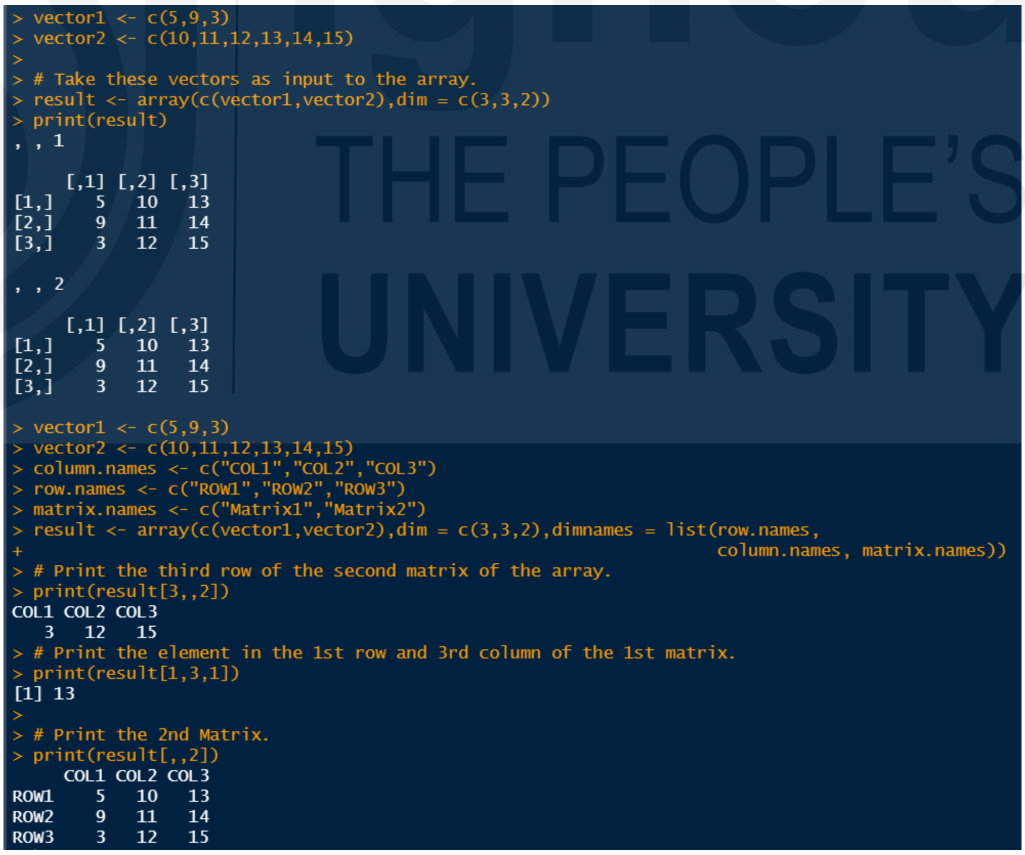
An array is a data object in R that can store multi-dimensional data that have the same data type. It is used using the array() function and can accept vectors as an input. An array is created using the values passed in the *dim* parameter.

For instance, an array is created with dimensions (2,3,5); then R would create 5 rectangular matrices comprising of 2 rows and 3 columns each. However, the data elements in each of the array will be of the same data type.



```
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
>
> # Take these vectors as input to the array.
> result <- array(c(vector1,vector2),dim = c(3,3,2))
> print(result)
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

Accessing Array Elements:



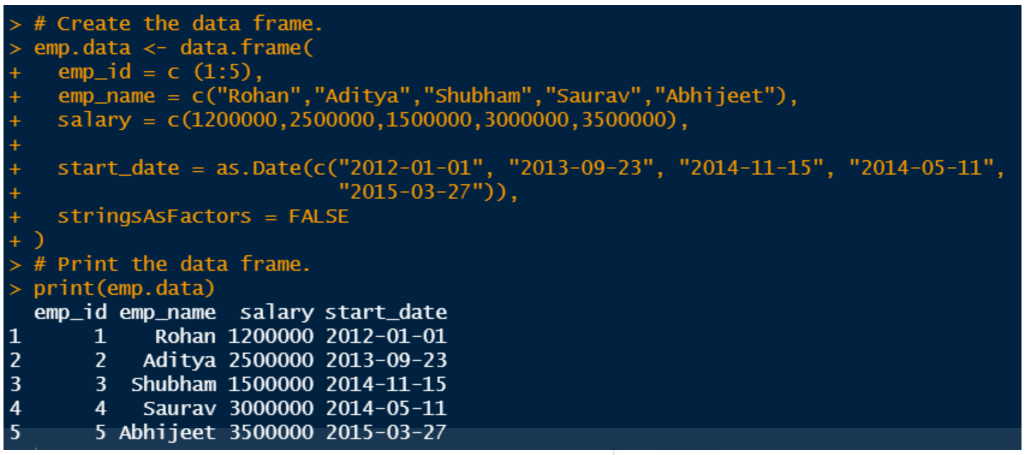
```
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
>
> # Take these vectors as input to the array.
> result <- array(c(vector1,vector2),dim = c(3,3,2))
> print(result)
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
> result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,
+                                                   column.names, matrix.names))
> # Print the third row of the second matrix of the array.
> print(result[3,,2])
COL1 COL2 COL3
   3   12   15
> # Print the element in the 1st row and 3rd column of the 1st matrix.
> print(result[1,3,1])
[1] 13
>
> # Print the 2nd Matrix.
> print(result[,,2])
     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
```

Dataframe:

A data frame represents a table or a structure similar to an array with two dimensions It can be interpreted as matrices where each column of that matrix can be of different data types.

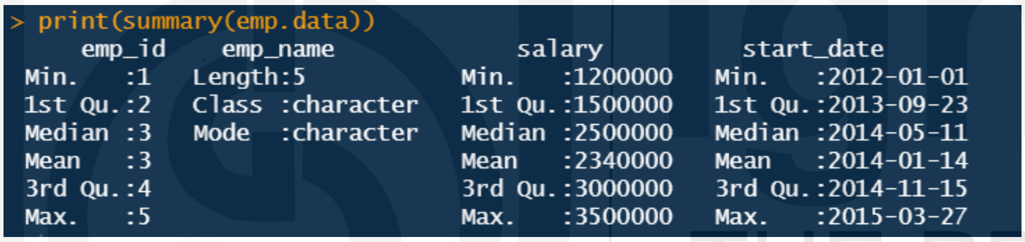
The characteristics of a data frame are given as follow

- The names of the columns should not be left blank
- The row names should be unique.
- The data frame can contain elements with numeric, factor or character data type
- Each column should contain same number of data items.



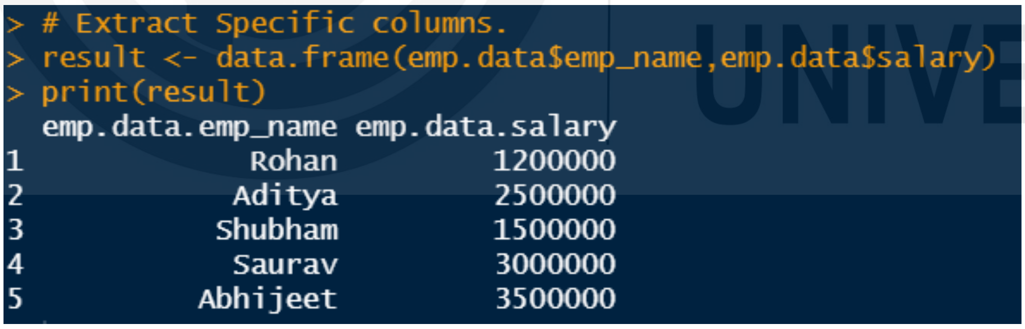
```
> # Create the data frame.
> emp.data <- data.frame(
+   emp_id = c (1:5),
+   emp_name = c("Rohan","Aditya","Shubham","Saurav","Abhijeet"),
+   salary = c(1200000,2500000,1500000,3000000,3500000),
+
+   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+                          "2015-03-27")),
+   stringsAsFactors = FALSE
+ )
> # Print the data frame.
> print(emp.data)
  emp_id emp_name  salary start_date
1      1    Rohan 1200000 2012-01-01
2      2   Aditya 2500000 2013-09-23
3      3  Shubham 1500000 2014-11-15
4      4   Saurav 3000000 2014-05-11
5      5 Abhijeet 3500000 2015-03-27
```

Statistical summary of the dataframe can be fetched using summary() function.



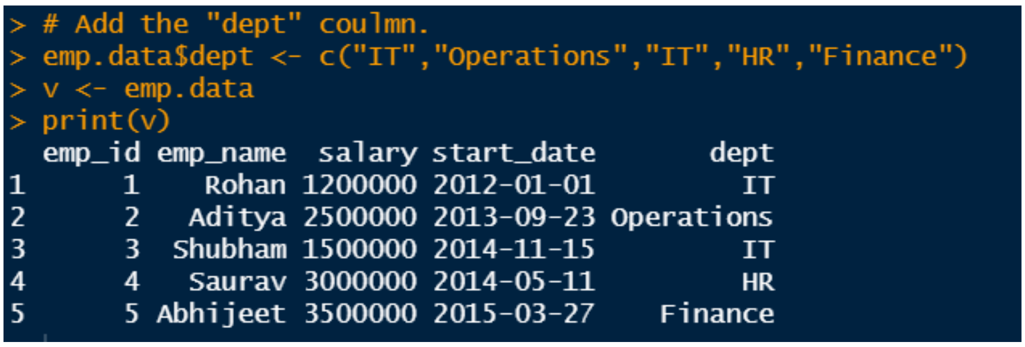
```
> print(summary(emp.data))
     emp_id    emp_name             salary          start_date
 Min.   :1   Length:5           Min.   :1200000   Min.   :2012-01-01
 1st Qu.:2   Class :character   1st Qu.:1500000   1st Qu.:2013-09-23
 Median :3   Mode  :character   Median :2500000   Median :2014-05-11
 Mean   :3                      Mean   :2340000   Mean   :2014-01-14
 3rd Qu.:4                      3rd Qu.:3000000   3rd Qu.:2014-11-15
 Max.   :5                      Max.   :3500000   Max.   :2015-03-27
```

Extracting specific data from data frame by specifying the column name.



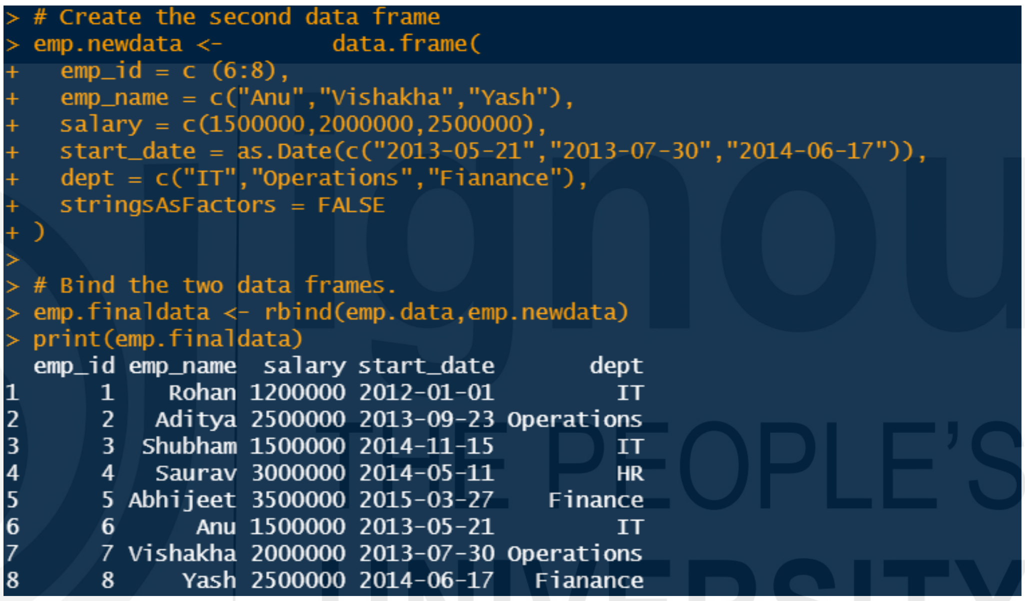
```
> # Extract Specific columns.
> result <- data.frame(emp.data$emp_name,emp.data$salary)
> print(result)
  emp.data.emp_name emp.data.salary
1             Rohan         1200000
2            Aditya         2500000
3           Shubham         1500000
4            Saurav         3000000
5          Abhijeet         3500000
```

Expanding the data frame by Adding additional column.



```
> # Add the "dept" coulmn.
> emp.data$dept <- c("IT","Operations","IT","HR","Finance")
> v <- emp.data
> print(v)
  emp_id emp_name  salary start_date       dept
1      1    Rohan 1200000 2012-01-01         IT
2      2  Aditya 2500000 2013-09-23 Operations
3      3  Shubham 1500000 2014-11-15         IT
4      4   Saurav 3000000 2014-05-11         HR
5      5 Abhijeet 3500000 2015-03-27    Finance
```

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the rbind() function.



```
> # Create the second data frame
> emp.newdata <-         data.frame(
+   emp_id = c (6:8),
+   emp_name = c("Anu","Vishakha","Yash"),
+   salary = c(1500000,2000000,2500000),
+   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
+   dept = c("IT","Operations","Fianance"),
+   stringsAsFactors = FALSE
+ )
>
> # Bind the two data frames.
> emp.finaldata <- rbind(emp.data,emp.newdata)
> print(emp.finaldata)
  emp_id emp_name  salary start_date       dept
1      1    Rohan 1200000 2012-01-01         IT
2      2   Aditya 2500000 2013-09-23 Operations
3      3  Shubham 1500000 2014-11-15         IT
4      4   Saurav 3000000 2014-05-11         HR
5      5 Abhijeet 3500000 2015-03-27    Finance
6      6      Anu 1500000 2013-05-21         IT
7      7 Vishakha 2000000 2013-07-30 Operations
8      8     Yash 2500000 2014-06-17   Fianance
```

Check Your Progress 2

1. Why are Matrices data structure not used that often?

……………………………………………………………………………………

…………………………………………………………………………………

2. What are the different data structures in R? Briefly explain about them.

……………………………………………………………………………………

……………………………………………………………………………………

3. What is the function used for adding datasets in R?

……………………………………………………………………………………

……………………………………………………………………………………

13.6 SUMMARY

The unit introduces you to the basics of R programming. It explains about the environment of R, a virtual space having collection of objects and how a new environment can be created within the global environment. The unit also explains about the various types of data associated with the variables that allocates a memory space and stores the values that can be manipulated. It also gives the details of the five types of operators in R programming. It also explains about factors that are the data objects used for organizing and storing the data as levels. The concept of decision making is also been discussed in detail that requires the programmer to specify one or more conditions to be evaluated or tested by the program. The concept of loops and their types has also been defined in this unit. It gives the details of function in R that is a set of instructions that is required to execute a a command to achieve a task in R. There are several built-in functions available in R. Further, users may create a function basis their requirements. The concept of matrices, arrays, dataframes etc have also been discussed in detail.

13.7 ANSWERS

Check Your Progress 1

1. The various operators in R are Arithmetic, Relational, Logical, assignment and Miscellaneous Operators. All of the above briefly explained in section 5.3

2. %*% Operator: It helps in multiplying a matrix with its transpose.

3. .5Var is an Invalid variable name as the dot is followed by a number

Check Your Progress 2

1. Matrices are not used much often as they contains only one data type and that too usually character or logical values.

2. Various Data Structures in R:

Data Structure	Description
Vector	A vector is a one-dimensional array of data elements that have same data type. These data elements in a vector are referred to as components.
List	Lists are the R objects which contain elements of different types like- numbers, strings, vectors or another list inside it.

Matrix	A matrix is a two-dimensional data structure. Matrices are used to bind vectors from the same length. All the elements of a matrix must have the same data type, i.e. (numeric, logical, character, complex).
Dataframe	A dataframe is more generic than a matrix, i.e. different columns can have different data types (numeric, logical etc). It combines features of matrices and lists like a rectangular list.

3. Rbind() is the function used to add datasets in R.

13.8 REFERENCES AND FURTHER READINGS

1. De Vries, A., & Meys, J. (2015). *R for Dummies*. John Wiley & Sons.
2. Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Victoria, BC, Canada: Leanpub.
3. Schmuller, J. (2017). *Statistical Analysis with R For Dummies*. John Wiley & Sons.
4. Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage publications.
5. Lander, J. P. (2014). *R for everyone: Advanced analytics and graphics*. Pearson Education.
6. Lantz, B. (2019). *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd.
7. Heumann, C., & Schomaker, M. (2016). *Introduction to statistics and data analysis*. Springer International Publishing Switzerland.
8. Davies, T. M. (2016). *The book of R: a first course in programming and statistics*. No Starch Press.
9. https://www.tutorialspoint.com/r/index.html