UNIT-IV

R Overview

1. R is built on 'S' programming language and was developed by Ross Ihaka and Robert Gentleman in 1993, at the university of Auckland.
2. R is a programming language and development environment specially designed for doing statistical computing and graphics.
3. R is a free and open source.
4. R has a vast and diverse package ecosystem.
5. There are around 9000 packages available from multiple repositories.
6. R is recognized for its beautiful visualization capabilities.

Advantages and Limitations of R Programming:

Advantages:

a. R is simple and effective programming language which has been well-developed, as well as R is data analysis software.
b. R is well-designed, easy and effective language that has the concepts of conditional, looping, user defined recursive procedures and various I/O facilities.
c. R has a large, consistent and incorporated set of tools used for data analysis.
d. R contains set of operators for different types of calculations on arrays, lists and vectors.
e. R provides highly extensible graphical techniques.
f. R graphical techniques for data analysis output either directly display to the computer, or can be print on paper.
g. R has an effective data handling and storage facility.
h. R is an online vibrant community.
i. R is free, open source, powerful and highly extensible.

Disadvantages/Limitations:

1. Very difficult to learn and need to put much effort to learn R programming.
2. Sometimes patchy documentation, hard to understand by non-statistician.
3. Quality of some package is less than perfect
4. Memory management issues.

BASIC SYNTAX:

a. Once we have completed setting up R environment, we can easily start R command prompt.
b. The command R launch the R interpreter with '>' prompt. Then we can start writing our programs.

```
>str <- "Hello World"
>print(str)
[1] "Hello World"
```

a. We will write programs in script files and then implement those scripts at command prompt using the help of R interpreter called Rscript.

```
Str< - "Welcome to R"

P,rint(str)
```

Save the code with .R extension and execute code using the command as shown below:

Rscript test.R

b. Comments are helping text within R source code and these statements get ignored by the interpreter while running your actual program.
c. A single line comments is written with the starting symbol '#' in the beginning of the statement.
   #MyFirst R Program
d. R does not support multi line comments.

DATA TYPES

A Data type is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.

The following are the list of data types provided by R:

1. Numeric
2. Integer
3. Complex
4. Logical
5. Character

Numeric Data Type:

Decimal values are referred as numeric data types in R. This is the default working out data type. If you assign a decimal value to any variable it will become a numeric type.

```
> x = 93.34  # assign a decimal value to x
> x          # print the variable's value - x
```

Integer Data Type:

A number without any decimal place is referred as integer. We can create integers in two ways.

1. Using as.integer( ) function
2. Place L followed by number.

```
> a = as.integer(3)
> a = 3L
```

Both statements creates an integer value.

a = as.integer(3.1412) stores only integer 3 into a. It works like type casting where the value 3.1412 gets changed to 3.

Complex Data Type:

A complex value for coding R can be defined using the pure imaginary values 'i'.

```
> x = 10 + i20 # creating complex number
> x            # printing the value of x
```

Logical Data Type:

A logical value is mostly created when comparison between variables is performed. Logical data type can store only two possible values i.e., TRUE and FALSE.

```
> x = 10 ; y = 20  # sample values
> z = x > y        # is x is larger than y?
> z               # print the logical value
[1] FALSE
```

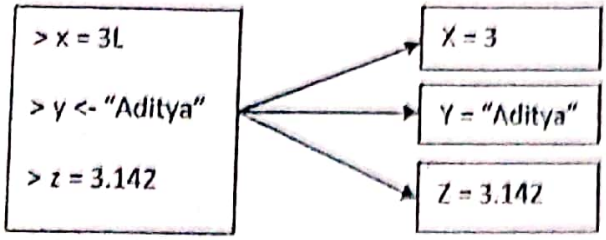
Character Data type:

A character object can be used for representing string values in R. We can convert objects into character values using the as.character( ) function.

```
> ch = as.character(1.43)
> ch
[1] "1.43"
> class(ch)
[1] "character"
```

VARIABLES:

1. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
2. Unique name has to be given to variable is known as identifier.
3. Identifiers are combination of alphabets, digits, dot and underscore.
4. It is mandatory to start an identifier with a character or dot.
5. When an identifier begins with a dot then it should not be followed by a digit.
6. Reserved words in R cannot be used as identifiers.



OPERATORS:

An operator is a symbol, operators are the constructs which can manipulate the value of operands. R has many operators to carry out different mathematical and logical operations. Operators in R are mainly classified into the following categories.

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators

Arithmetic Operators:

These operators are used to carry out mathematical operations like addition, multiplication etc. The following are the list of arithmetic operators available in R.

operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
%%	Modular (Remainder)
%/%	Integer Division (Quotient)

Example:

```
> x <- 5
> y <- 16
```

> x+y [1] 21	> x-y [1] -11	> x*y [1] 80	> y/x [1] 3.2	> y%/%x [1] 3	> y%%x [1] 1	> y^x [1] 1048576

Relational Operators:

Relational operators are used to compare between values. The following are the list of relational operators available in R.

Symbol	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

```
> x <- 5
> y <- 16
```

> x<y [1] TRUE	> x>y [1] FALSE	> x<=5 [1] TRUE	> y>=20 [1] FALSE	> y — 16 [1] TRUE	> x !- 5 [1] FALSE

Logical Operators:

Logical operators are used to carryout boolean operations like AND, OR, NOT. The value '0' is considered as FALSE and any non zero value is considered as TRUE. The following are the list of logical operators in R.

Symbol	Meaning
!	Logical NOT
&	Element-wise Logical AND
&&	Logical AND
|	Element-wise Logical OR
||	Logical OR

Example:

```
> x<- c(TRUE,FALSE,0,6)
> y <- c(FALSE,TRUE,FALSE,TRUE)
> x
[1] FALSE TRUE TRUE FALSE
> x&y
[1] FALSE FALSE FALSE TRUE
> x&&y
[1] FALSE
> x |y
[1] TRUE TRUE FALSE TRUE
> x||y
[1] TRUE
```

Assignment Operators:

Assignment operators are used to assign a value to a variable. The following are the list of assignment operators in R.

Symbol	Meaning
<-, <<-,=	Leftwards assignment
->,->>	Rightwards assignment

Example:

```
> x <- 5
> x
[1] 5
```

```
> x = 9
> x
[1] 9
```

```
> 10 -> x
> x
[1] 10
```

DECISION MAKING:

R programming provides three decision making statements that allows programmers to control their statements within source code. The following are the list of decision making statements in R.

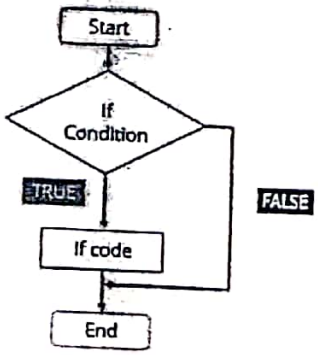
1. if statement
2. if....else statement
3. switch statement

if statement:

The simplest form of decision controlling statement for conditional execution is the 'if' statement. The 'if' produces a logical value (more exactly, a logical vector having length one) and carries out the next statement only when that value becomes TRUE. In other words, an 'if' statement is having a Boolean expression followed by single or multiple statements.

Syntax:

```
if(expression)
{
      //statements
}
```



Example:

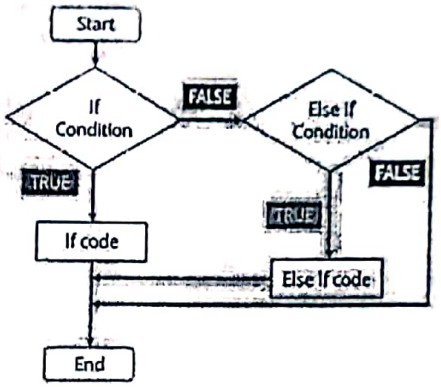
```
if (TRUE) print ("One line executed")
## One line executed
if (FALSE) print ("Line not executed")
## Line not executed
if (NA) print ("Don't know whether true or not.")
## Error: missing value where TRUE/FALSE needed
```

if...else statement:

In this type of statements the 'if' statement is usually followed by an optional 'else' statement that gets executed when the Boolean expression becomes false. This statement is used when you will be having multiple statements with multiple conditions to be executed.

Syntax :

```
if(expression-1)
{
        //statements-1
}
else if(expression-2)
{
        //statements-2
}
```



Example:

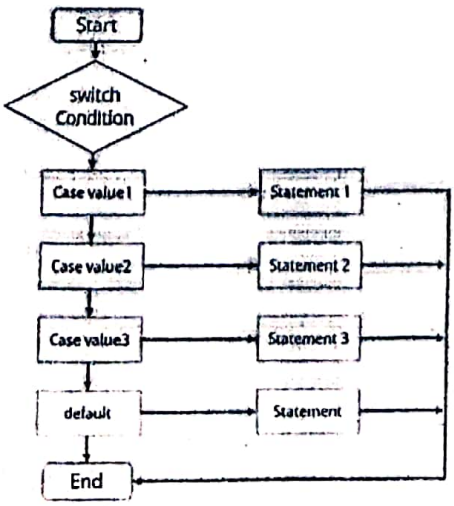
```
if (TRU E)
{
print ("This will execute...")
} else
{
print ("but this will not.")
}
## This will execute...
```

Switch Statement:

A switch statement permits a variable to be tested in favor of equality against a list of case values. In the switch statement, for each case the variable which is being switched is checked. This statement is generally used for multiple selection of condition based statement.

```
switch(expression,
case 1=#statements-1
case 2=#statements-2
case 3=#statements-3

default statement
)
```



Example:

```
gk <- switch (
2,
"First",
"Second",
"Third",
"Fourth"

print (gk)
## [1] "Second"
```

LOOPS:

Looping control statements are used to repeat a specific block of code for a certain number of times based on a condition. The following are the list of looping control statements in R.

1. for
2. while
3. repeat

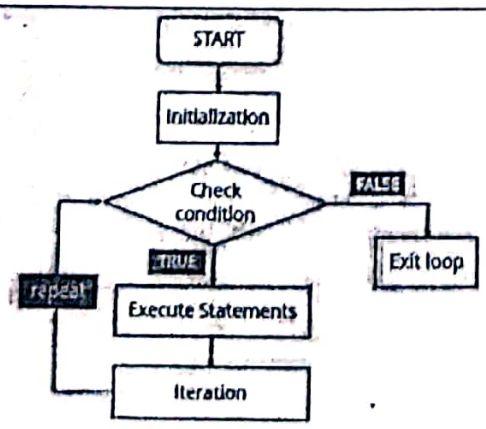
for loop:

For loop is used to read each and every element from a vector and performs an operation on it. For loop takes a variable and a vector.

Syntax:

```
for (val in vector)
{
statement
}
```

Flow Chart:



Example:

```
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x)
{
 if(val %% 2 == 0)
 count = count+1
}
print(count)
```

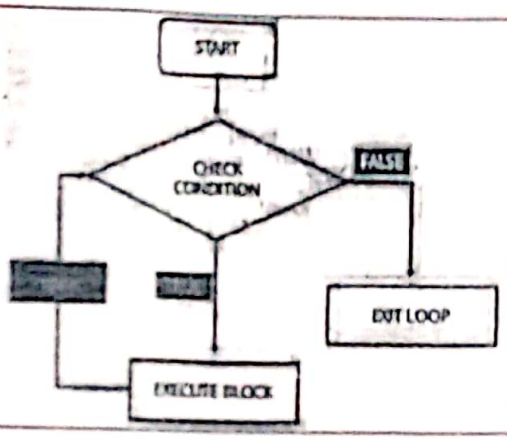
Output

```
[1] 3
```

While loop:

While loop is used to repeats a statement or group of statements until given condition is TRUE. It tests the condition before executing the loop body.

```
while(condition)
{
   //statements
}
```

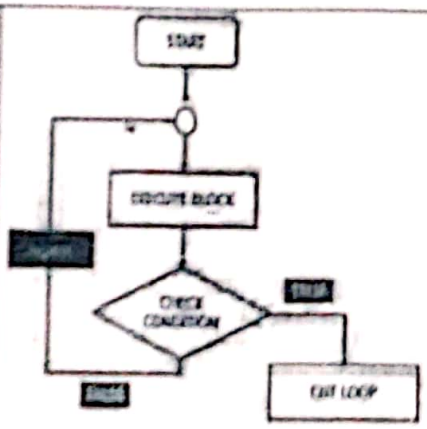



```
i <- 1
while (i < 6)
{
  print(i)
  i = i+1
}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Repeat loop:

Repeat loop is used to execute a block of code multiple number of times. There is no condition is associated with repeat loop to terminate. We have to specify the condition explicitly to terminate the repeat loop.

```
repeat
{
   commands
   if(condition)
   {
      break
   }
}
```




```
x <- 1
repeat
{
   print(x)
   x = x+1
   if (x == 6)
   {
      break
   }
}
Output
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

FUNCTIONS:

a. A function is a block of organized, reusable code that is used to perform a single, related action.
b. A function can be defined as a collection of statements structured together for carrying out a definite task.
c. A function is treated as object in R.
d. The keyword "function" is used to create a function in R.

Syntax:

```
function_name <- function(arg-1,arg-2,...arg-N)
{
  #Function Definition
}
```

e. Functions are classified into two types.
   1. Built In Functions
   2. User defined functions

BUILT IN FUNCTIONS

Built in functions are those functions whose meaning and working is already defined within the function's body and they are kept somewhere within the packages or libraries of R language. These pre-defined functions make programmers task easier.

seq(),max(), mean(), sum(x), paste(...) etc.

```
print (seq (12,30))
```

This creates a sequence of number from 12 to 30 using the predefined function seq().

```
print (mean (4:26))
```

This calculates the mean of all the numbers ranging from 4 to 26

USER DEFINED FUNCTIONS

Syntax:

```
func_name <- function (argument)
{
  statement
}
```

Example:

```
pow <- function(x, y)
{
  # function to print x raised to the power y
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
```

- Here, we created a function called pow().
- It takes two arguments, finds the first argument raised to the power of second argument and prints the result in appropriate format.
- We have used a built-in function paste() which is used to concatenate strings.
- We can call the above function as follows:

```
>pow(8, 2)
[1] "8 raised to the power 2 is 64"
> pow(2, 8)
[1] "2 raised to the power 8 is 256"
```

Default Values for Arguments:

We can assign default values to arguments in a function in R. This is done by providing an appropriate value to the formal argument in the function declaration. Here is the above function with a default value for y.

```
pow <- function(x, y=2)
{
  # function to print x raised to the power y
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
```

The use of default of value to an argument makes it optional when calling the function.

```
> pow(3)
[1] "3 raised to the power 2 is 9"
> pow(3,1)
[1] "3 raised to the power 1 is 3"
```

Here, y is optional and will take the value 2 when not provided.