5. Programming Concepts

5.1 Programming Concepts

- R is a computer language which is processed by a special program called an interpreter.
- This program reads and evaluates R language expressions, and prints the values determined for the expressions.
- The interpreter indicates that it is expecting input by printing its prompt at the start of a line.
- By default, the R prompt is a greater than sign >.

5.1 Programming Concepts - continued

- R can be used as a Calculator
- For grouping and evaluation. normal arithmetic rules apply; multiplication and division occur before addition and subtraction.
- Operator precedence

Operator	Description
^	Exponentiation
- +	unary minus and plus
:	sequence operator
%/% %%	integer division, remainder
* /	multiplication, division
+ -	addition, subtraction

Evaluation of equal precedence takes place left-to-right (except for exponentiation, which takes place right-to-left)

5.2 Vectorized calculations

1. Element-wise Operations on Vectors

Suppose we have a function f() that we wish to apply to all elements of a vector x.

```
> u <- 10:15
> v <- 1:6
> u + v # Sum of two vectors u and v
[1] 11 13 15 17 19 21
> u - v # Difference between two vectors u and v
[1] 9 9 9 9 9 9
> u * v # Product of two vectors u and v
[1] 10 22 36 52 70 90
> u / v # Division of two vectors u and v
[1] 10.00  5.50  4.00  3.25  2.80  2.50
>
```

5.2 Vectorized calculations - continued

1. **Element-wise Operations on Vectors – continued.**

   In many cases, we can accomplish this by simply calling f() on x itself.

```
> z12 <- function(x) return(x^0.5)
> z12(c(4,9,16,25))
[1] 2 3 4 5
>
```

5.2 Vectorized calculations - continued

2. Filtering

```
> z <-c(5,2,-3,8)
> w <-z[z*z >20]
> w
[1] 5 8
```

Here is what happened: We asked R to find the indices of all the elements of z whose squares were greater than 20, then use those indices in an indexing operation on z, then finally assign the result to w.

```
> which(z*z>20)
[1] 1 4
```

We may just want to find the positions within z at which the condition occurs. We can do this using which():

5.2 Vectorized calculations - continued

2. Filtering

```
> a <- c(12,13,14,15,16,17,18)
> ifelse(a >15,a-10,a)
[1] 12 13 14 15  6  7  8
>
```

- The form is ifelse(b,u,v) where b is a boolean vector, and u and v are vectors.
- The return value is a vector, element i of which is u[i] if b[i] is true, or v[i] if b[i] is false.

5.3 Control structures

- Computation in R consists of sequentially evaluating statements.
- Statements, such as x <- 10:100 or mean(x), can be separated by either a semi-colon or a new line.
- Statements can be grouped together using braces '{' and '}'. A group of statements is called a block. Single statements are evaluated when a new line is typed at the end of the syntactically complete statement.

```
> { i <- 50
+ i /10
+ }
[1] 5
>
```

5.3 Control structures - continued

- The following are the basic control-flow constructs of the R language:
- They function in much the same way as control statements in any programming language.

a. Statements that perform testing that shifts flow

1. if statements
2. switch statements

b. Statements that perform / control looping

1. for
2. while
3. repeat
4. break
5. next

5.3 Control structures - continued

```
> x <- 8
> if (x < 2) {
+   cat ("Increment that number \n")
+ } else
+ {
+   x <- x -2
+   cat ("Make it smaller \n" )
+ }
Make it smaller
>
```

5.3 Control structures - continued

```
> basic_fn<-function(x)
+ {
+ switch(x,
+ "mean" = { Ans <- mean(y); cat("mean",Ans,sep=" ","\n")},
+ "median" = { Ans <- median(y); cat("median",Ans,sep=" ","\n")},
+ "sd" = { Ans <- sd(y); cat("sd",Ans,sep=" ","\n")}
+ )
+ }
> y <- rnorm(10) # this generates 10 random variables of normal distribution
> x = "mean";basic_fn(x)
mean 0.1650489
> x = "median";basic_fn(x)
median 0.1543456
>
```

5.3 Control structures - continued

```
> for (i in 1:8)
+ {
+       print(paste("i =",i));
+ }
[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
[1] "i = 4"
[1] "i = 5"
[1] "i = 6"
[1] "i = 7"
[1] "i = 8"
```

```
> i = 1
> while ( i < 9)
+ {
+ print(paste("i =",i))
+ i = i + 1
+ }
[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
[1] "i = 4"
[1] "i = 5"
[1] "i = 6"
[1] "i = 7"
[1] "i = 8"
```

5.3 Control structures - continued

The *break* statement breaks out of a *for*, *while* or *repeat* loop.

```
> i = 1
> repeat
+ {
+  print(paste("i =",i))
+  if ( i >7) break;
+  i <- i + 1
+ }
[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
[1] "i = 4"
[1] "i = 5"
[1] "i = 6"
[1] "i = 7"
[1] "i = 8"
>
```

5.3 Control structures - continued

The *next* statement halts the processing of the current iteration and advances the looping index.

```
> i = 0
> num_ch <- c("One","Two","Three","Four","Five","Six","Seven")
> a <- c("","","","","","","")
> while (i < 7)
+ {
+   i <- i + 1
+   if (i == 5) next
+   a[i] <- num_ch[i]
+ }
> print(a)
[1] "One"   "Two"   "Three" "Four"  ""      "Six"   "Seven"
>
```

5.4 Scoping rules

- The symbols which occur in the body of a function can be divided into three classes:
  1. formal parameters
  2. local variables
  3. free variables

- The formal parameters of a function are those occurring in the argument list of the function.
- Their values are determined by the process of binding the actual function arguments to the formal parameters.

5.4 Scoping rules - continued

- Local variables are those whose values are determined by the evaluation of expression in the body of the functions.

- Variables which are not formal parameters or local variables are called free variables.

```
> f <- function(x) {
+ y <- x + 2
+ x+y+a
+ }
> a <- 4
> f(44)
[1] 94
```

In this function, f

- *x is a formal parameter,*
- *y is a local variable and*
- *a is a free variable*

5.5 Writing functions

- **The syntax for writing a function is**
  function (arglist) body

R Information

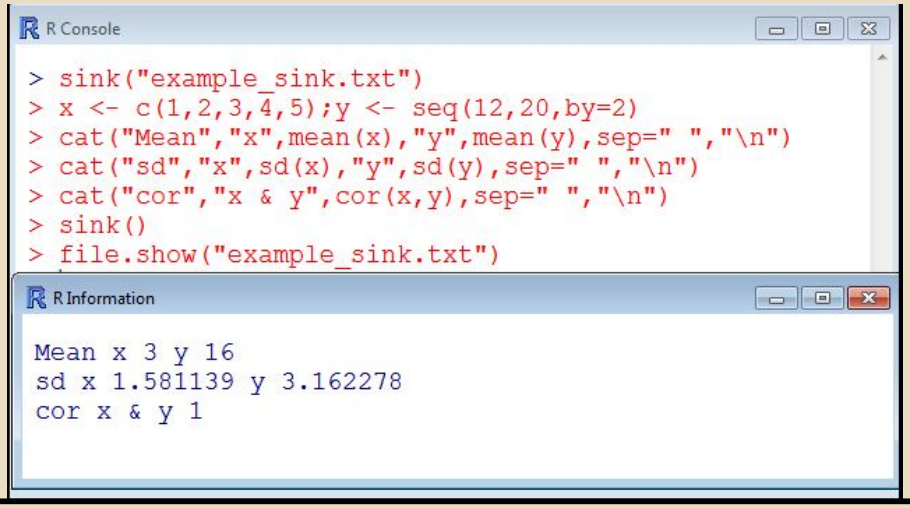
```
freqmatrix <- function(x) {
          cumfreq   =   cumsum(x)
          sumv = sum(x)
          morethanfrq = sumv
          xlen = length(x)
          rvcumfrq = rep(0,xlen)
          for ( i in 1:xlen) {
                    rvcumfrq[i] =  morethanfrq
                    morethanfrq = morethanfrq - x[i]
           }
          m <- as.matrix(cbind(x,cumfreq,rvcumfrq))
          return (m)
}
```

5.5 Writing functions - continued

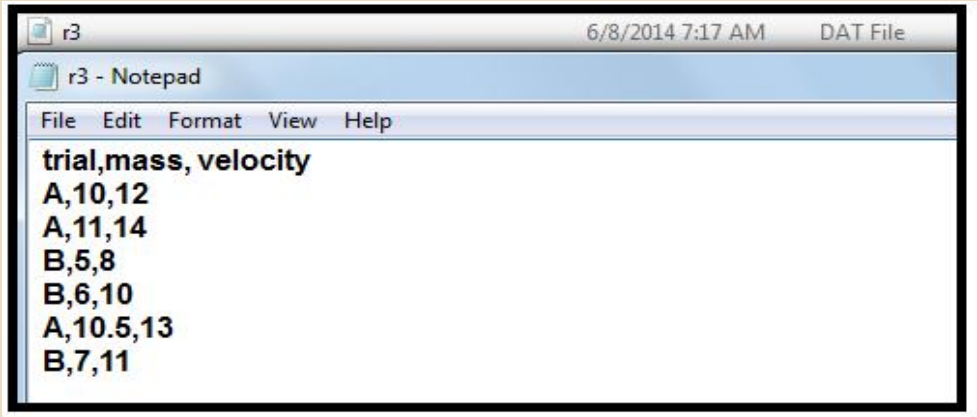
- The first component of the function declaration is the keyword *function* which indicates to R that you want to R that you want to create a function.
- An argument list is a comma separated list of formal arguments. A formal argument can be a symbol, a statement of the form *'symbol = expression'*, or the special formal argument '....'.
- The *body* can be any valid R expression. Generally, the body is a group of expressions contained in curly braces '{' and '}'.

5.6 Directing console output to a file

- sink(file) diverts R output to a connection; where file is a writable connection or a character string naming the file to write to, or NULL to stop sink-ing.
- The command file.show displays one or more files.



5.7 CSV / fixed-width file operations



The following command will read in the data and assign it to a variable called "a".

Note: Last line in r3.dat is a blank line.

```
> a <- read.csv(file='r3.dat',head=TRUE,sep=",")
> a
  trial mass velocity
1     A 10.0       12
2     A 11.0       14
3     B  5.0        8
4     B  6.0       10
5     A 10.5       13
6     B  7.0       11
```

5.7 CSV / fixed-width file operations - continued

Fixed width file

- **Here, the data is organized in flat files and delimited at preset locations on each line.**
- **The command to deal with these kind of files is read.fwf.**

File contents of r2.dat

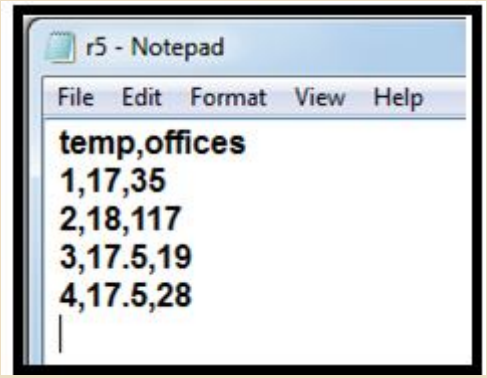
```
1 17.000000035
2 18.000000117
3 17.500000019
4 17.500000028
<blank line>
```

```
> atemp = read.fwf('r2.dat',widths=c(-2,7,9),col.names=c('temp','offices'),nrows=4)
> atemp
  temp offices
1 17.0      35
2 18.0     117
3 17.5      19
4 17.5      28
```

5.7 CSV / fixed-width file operations - continued

The function write.table writes file, an object typically a data frame but this could well be another kind of object vector, matrix,…).

```
> write.table(atemp,file="r5.dat",append=FALSE,quote=FALSE,sep=",",eol="\n",
 na="NA",row.names=TRUE,col.names=TRUE,qmethod=c("escape","double"))
```



5.7 Debugging

- Let us make a mistake in the following R script, by replacing meana with meanc, which is non-existent.
- The traceback() function prints call stack, the list of functions which were called before the error occurred.

```
> # traceback
> x<- c(1,2,3,4,5)
> y<-c(11,12,13,14,15)
> meanx<- mean(x)
> meany<-mean(y)
> cat("Mean", "x",meanc,"y",meany,sep=" ","\n")
Error in cat("Mean", "x", meanc, "y", meany, sep = " ", "\n") :
  object 'meanc' not found
> cat("Mean", "x",meanx,"y",meany,sep=" ","\n")
Mean x 3 y 13
> traceback()
1: cat("Mean", "x", meanc, "y", meany, sep = " ", "\n")
```

R Programming Course

R Programming Course

1. **The following R code changes the value of its argument inside the function. What it is the value of x after running the below piece of R**

```
> x <- 1
> cx  <-  function() { x <- x*2; print (x*x)}
> cx()
[1] 4
> x
```

a. 4
b. 2
c. 1
d. None of the above

R Programming Course

2. The output after running the following R code is

```
> g <- c("M","F","F","F","F","M","M","F")
> ifelse(g == "M",1,2)
```

a) `1] 2 1 1 1 1 2 2 1`

b) `[1] 1 2 2 2 2 1 1 2`

c) Error message

d) None of the above

1. c)

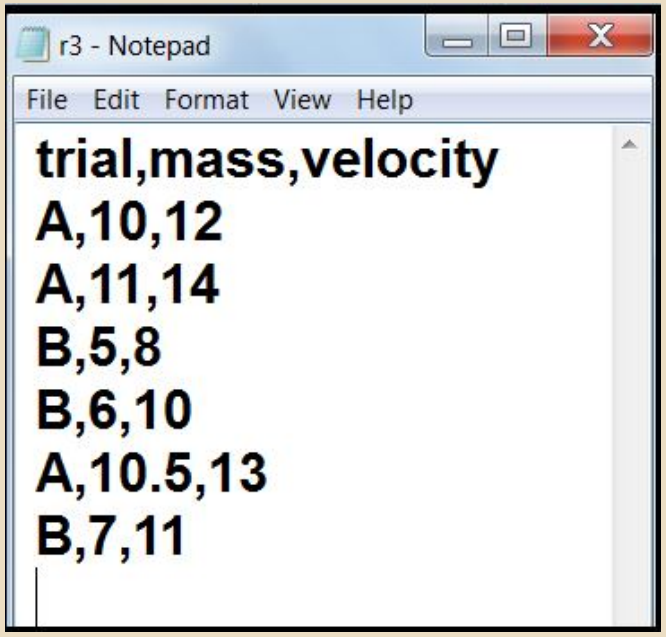
2. b)

R Programming Course

R Programming Course

Lab Exercise 1:

- **Read the csv type file with name "r3.dat" into an object a and print its contents.**
- **The file r3.dat and available in your local drive under the folder R_data.**



Lab Exercise 1 - continued:

- **The command to read the data file is read.csv.**
- **The first argument is the name of the file.**
- **The second argument indicates whether or not the first row is a set of labels.**

```
> a <- read.csv("r3.dat",head=TRUE)
> a
  trial mass velocity
1     A 10.0       12
2     A 11.0       14
3     B  5.0        8
4     B  6.0       10
5     A 10.5       13
6     B  7.0       11
>
```

R Programming Course

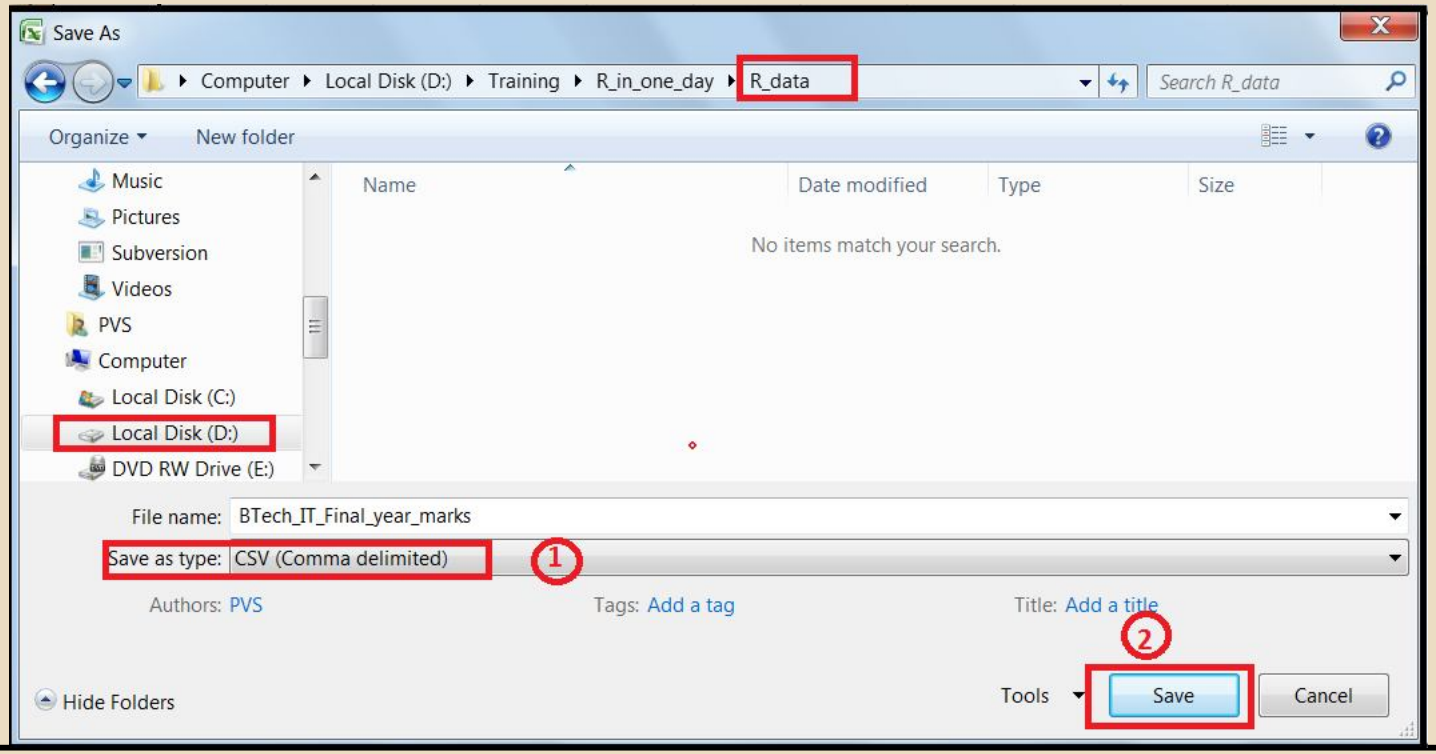
Lab Exercise 2:

- **Assume you are processing the examination results of the BTech - IT students.**
- **You have the Roll number, name, marks (max=100) in each of the five exam papers.**
- **You declare the result as "pass", if the student gets not less than 50 in each paper; if the student gets less than 50 marks, the result is "fail".**
- **Print a tabulated mark register containing the following details:**
  1. **Serial Number**
  2. **Roll number**
  3. **Name of the student**
  4. **Marks obtained in each of the five exam papers**
  5. **Result**

R Programming Course

Lab Exercise 2- continued:

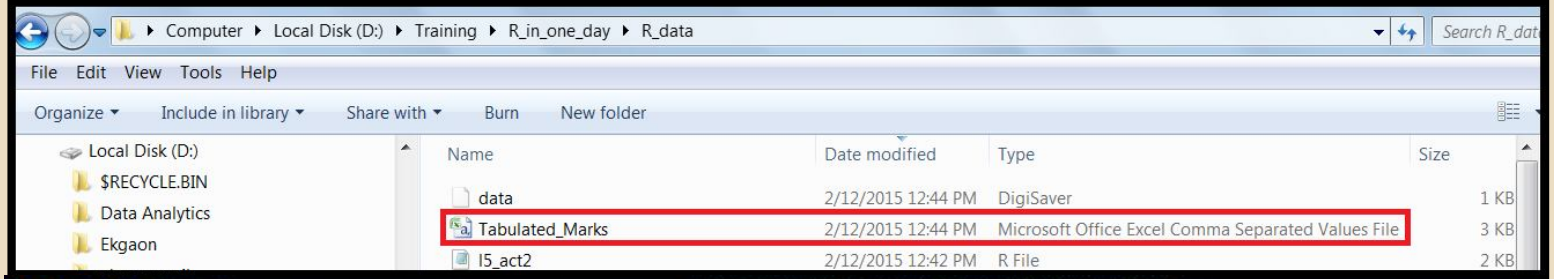
- **The file "Btech_IT_Final_year_marks" is an excel file containing the data.**
- **Convert the same to a csv file as follows:**

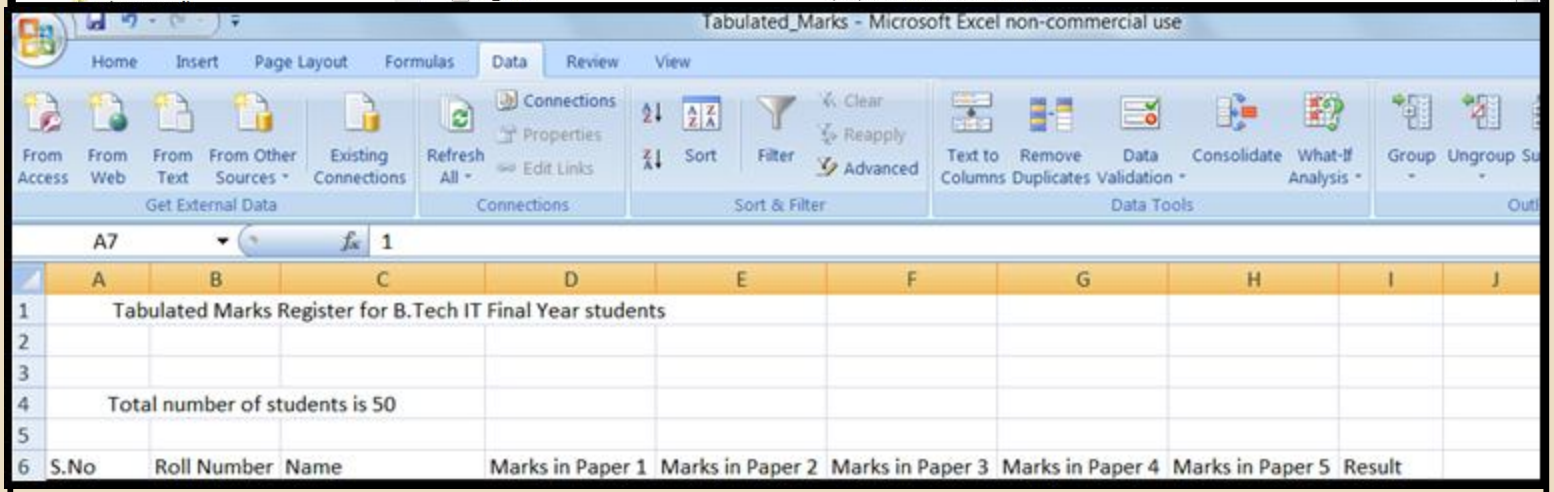


R Programming Course

Lab Exercise 2- continued:

```
> source("l5_act2.R")
Warning message:
In write.table(my_df, file = "Tabulated_Marks.csv", sep = ",", append = TRUE,  :
  appending column names to file
```





R Programming Course

Lab Exercise 2- continued:

37	31	4015042	Ramya C	52	45	72	68	86	Fail
38	32	4015043	Sangeetha Nagaraj	49	58	90	60	78	Fail
39	33	4015044	Siva S	64	81	74	88	83	Pass
40	34	4015045	Shivakumar S	93	80	81	54	44	Fail
41	35	4015046	Sowmya K	75	69	88	56	62	Pass
42	36	4015047	Soundharya L	82	78	50	81	75	Pass
43	37	4015048	Sunitha S G	98	40	54	97	49	Fail
44	38	4015049	Suresh A	94	77	98	99	81	Pass
45	39	4015050	Swathy K K	86	48	53	51	91	Fail
46	40	4015051	Radithiuma B	74	64	93	93	51	Pass
47	41	4015052	Vani P	95	95	71	98	96	Pass
48	42	4015053	Vasanth N	72	56	41	73	89	Fail
49	43	4015054	Vedha R	55	91	44	84	42	Fail
50	44	4015055	Babu V L	53	47	66	76	85	Fail
51	45	4015056	Vrinidha K S	58	46	84	92	95	Fail
52	46	4015057	Vignesh V	90	44	43	48	100	Fail
53	47	4015058	Viswanathan V S	44	50	83	75	82	Fail
54	48	4015059	Bharathi A S	92	92	77	42	54	Fail
55	49	4015060	Gunasekar D	67	98	87	78	60	Pass
56	50	4015061	Alwin A G	57	54	62	69	52	Pass

Tabulated_Marks

R Programming Course

Lab Exercise 2- continued:

D:\Training\R_in_one_day\R_data\l5_act2.R - R Editor

```
#-------------------------------------------------------------------------------
# l5_act2.R
#-------------------------------------------------------------------------------
marks_students    <- read.csv(file="BTech_IT_Final_year_marks.csv",head=TRUE)
sno               <-   seq(1,length(marks_students$Marks.in.Paper.1))
m1                <-    marks_students$Marks.in.Paper.1
m2                <-    marks_students$Marks.in.Paper.2
m3                <-    marks_students$Marks.in.Paper.3
m4                <-    marks_students$Marks.in.Paper.4
m5                <-    marks_students$Marks.in.Paper.5
pass              <-    "Pass"
fail              <-    "Fail"
res1              <-    ifelse(m1 < 50,FALSE,TRUE)
res2              <-    ifelse(m2 < 50,FALSE,TRUE)
res3              <-    ifelse(m3 < 50,FALSE,TRUE)
res4              <-    ifelse(m4 < 50,FALSE,TRUE)
res5              <-    ifelse(m5 < 50,FALSE,TRUE)
f_res             <-    res1 & res2 &res3 & res4 & res5
final_res         <-    ifelse(f_res,pass,fail)
my_df             <-    cbind(sno,marks_students,final_res)
#
# Write title of the report
#
write(cat(sprintf("            Tabulated Marks Register for B.Tech IT Final Year students \n\n
          Total number of students is %d \n\n",length(sno)),
file="Tabulated_Marks.csv",sep=" ",append = FALSE))
#
write.table(my_df,file="Tabulated_Marks.csv",sep=",",append = TRUE,
row.names = FALSE,col.names=c("S.No","Roll Number","Name","Marks in Paper 1",
"Marks in Paper 2","Marks in Paper 3","Marks in Paper 4",
"Marks in Paper 5","Result"),quote=FALSE,eol="\n")
#-------------------------------------------------------------------------------
```

Activity 1:

Refer to the file

“U05_R Programming_v1.2.pdf“