

# MOOC Course - Foundations of R Software

July 2022

## Assignment 1

1. Which of the following command is used to get help on the `mean` function?

- a. `"?mean"`
- b. `?'mean'`
- c. `?mean`
- d. `mean?`

2. Which of the following is the correct option to provide help in R?

- a. `help.search()`
- b. `start.help()`
- c. `search.help()`
- d. `help.start()`

3. Which of the following function provides the demonstration of the package `lm.glm` in R?

- a. `demons(lm.glm)`
- b. `Demo('lm.glm')`
- c. `demo(lm.glm)`
- d. `demo(!lm.glm)`

4. Which of the function provides the help on finding an example on the function `var`?

- a. `example("var")`
- b. `ex(var)`
- c. `show.example(var)`
- d. `show.example("var")`

5. Which of the function does not provide the help in finding the contents of the library **bayesm**?

- a. `library(help=bayesm)`
- b. `library(help="bayesm")`
- c. `library(help='bayesm')`
- d. `library(help=bayesm?)`

6. Which of the following command installs the package **bayesm**?

- a. `install.packages(bayesm)`
- b. `install.packages("bayesm")`
- c. `install.packages(bayesm?)`
- d. `install.package("bayesm")`

7\*. Which of the following command is used to update the package **bayesm**?

- a. `update.packages(bayesm)`
- b. `update.packages("bayesm")`
- c. `update.packages(bayesm?)`
- d. `update.package("bayesm")`

8. Which of the following command detaches the package **bayesm**?

- a. `detache(package:bayesm, unload=TRUE)`
- b. `detached("package:bayesm ", unload=TRUE)`
- c. `Detach("package:bayesm", unload=FALSE)`
- d. `detach("package:bayesm", unload=TRUE)`

9. Which of the following command removes the package **bayesm**?

- a. `remove.packages(bayesm)`
- b. `remove.packages("bayesm")`**
- c. `remove.packages(bayesm?)`
- d. `remove.package("bayesm")`

10. Which of the following command provides the description of the package **cluster**?

- a. `packageDescription("cluster")`
- b. `packageDescribe("cluster")`
- c. `PackageDescription("cluster")`
- d. `packageDescription("cluster")`**

## **MOOC Course – Foundations of R Software**

### **Answers of Assignment 1**

1. c

2. d

3. c

4. a

5. d

6. b

7. b

8. d

9. b

10. d

# MOOC Course - Foundations of R Software

July 2022

## Assignment 2

1. Which of the following command is used to search the web for information and answers about keyword `mean` ?

- a. `RsiteSearch("mean")`
- b. `RSiteSearch(mean)`
- c. `RSiteSearch("mean")`
- d. `rsitesearch("mean")`

2. Which of the following is the correct matching of the commands in column A and the outputs in Column B when executed in R console where `x = 20` and `y = "20"`?

Column A		Column B	
(1)	<code>is.character(x)</code>	(i)	<code>TRUE</code>
(2)	<code>is.character(y)</code>	(ii)	<code>FALSE</code>

- a. (1)-(ii), (2)-(i)
- b. (1)-(i), (2)-(ii)
- c. (1)-(i), (2)-(i)
- d. (1)-(ii), (2)-(ii)

3. Which of the following is the correct matching of the commands in column A and the outputs in Column B when executed in R console where `x = 20` and `y = "20"`?

Column A		Column B	
(1)	<code>is.numeric(x)</code>	(i)	<code>TRUE</code>
(2)	<code>is.numeric(y)</code>	(ii)	<code>FALSE</code>

- a. (1)-(ii), (2)-(i)
- b. (1)-(i), (2)-(ii)**
- c. (1)-(i), (2)-(i)
- d. (1)-(ii), (2)-(ii)

4. Which of the following is the correct matching of the commands in column A and the outputs in Column B when executed in R console where `x = 20` and `y = "20"`?

Column A		Column B	
(1)	<code>mode(x)</code>	(i)	<code>numeric</code>
(2)	<code>mode(y)</code>	(ii)	<code>character</code>

- a. (1)-(ii), (2)-(i)
- b. (1)-(i), (2)-(ii)**
- c. (1)-(i), (2)-(i)
- d. (1)-(ii), (2)-(ii)

5. Which of the following is the correct matching of the commands in column A and the outputs in Column B when executed in R console where `x = 20` and `y = "20"`?

Column A		Column B	
(1)	<code>storage.mode(x)</code>	(i)	<code>double</code>
(2)	<code>storage.mode(y)</code>	(ii)	<code>character</code>

- a. (1)-(ii), (2)-(i)
- b. (1)-(i), (2)-(ii)**
- c. (1)-(i), (2)-(i)
- d. (1)-(ii), (2)-(ii)

6. Which of the following is the correct matching of the commands in column A and the outputs in Column B when executed in R console where `x = 20` and `y = 20 + Inf`?

Column A		Column B	
(1)	<code>is.finite(x)</code>	(i)	<code>TRUE</code>
(2)	<code>is.infinite(y)</code>	(ii)	<code>FALSE</code>

- a. (1)-(ii), (2)-(i)
- b. (1)-(i), (2)-(ii)
- c. (1)-(i), (2)-(i)**
- d. (1)-(ii), (2)-(ii)

7. What will be the outcome of following commands when executed over the R console

```
x=43  
y=x^3  
z=y^2+x^4  
z
```

- a. 6231634898
- b. 6321364898
- c. 6324781850
- d. 6342718850

8. Which one of the following is the correct outcome of the command

$5-3+4/6*7-3^3*4-7/6*2-3^{**3}/2+6-7+2$  ?

- a. -107.1667
- b. -116.1667
- c. -125.5
- d. 116.1667

9. Which one of the following is the correct outcome of the command

$15/(-5)*2+8^(-3)/(65/32)-4^{**3}+4-81^(-1/4)+2^{(-4^{**3})}$  ?

- a. -66.33237
- b. -77.33237
- c. -65.32148
- d. None of these

10. Which one of the following is the correct outcome of the command

`c(12,13,15,17)/10 + c(1.2,1.3,1.5,1.7)*10?`

- a. `24 26 30 34`
- b. `2.4 2.6 3.0 3.4`
- c. `13.2 14.3 16.5 18.7`
- d. None of these

## MOOC Course – Foundations of R Software

### Answers of Assignment 2

- 1. c
- 2. a
- 3. b
- 4. b
- 5. b
- 6. c
- 7. c
- 8. b
- 9. a
- 10. c

## MOOC Course - Foundations of R Software

July 2022

### Assignment 3

1. Which one of the following is the correct outcome of the command

```
c(1,2,3,4)/c(1,2,2,4)*c(2,1,3,2)**c(1,2,3,3)-
c(2,2,32,5) ?
```

- a. 0.0 0.0 -18.5 1.0
- b. 0.0 -1.0 8.5 3.0**
- c. -1.000 0.000 -30.875 -4.250
- d. Error:....

2. Which one of the following is the correct outcome of the command

```
c(9,16,6,20)/c(3,4)+ c(4,2,4,2)^c(2,3)+c(2,4,8,6)/c(2,2)-
c(3,8,2,4)*c(3,1) ?
```

- a. 11 6 16 12**
- b. 9 2 8 6
- c. -23 -14 -24 -10
- d. None of these

3. Which one of the following command will give a result without a warning?

- a. `c(1,3,5,7)^c(8,9,4) - c(2,3,4,5)**c(12,3,14,5)- c(240,30, 76,98)/c(8,3)`
- b. `c(1,3,5,7)^c(8,4) - c(2,3,4,5)**c(12,3,14,5)- c(240,30, 76,98)/c(8,3)`
- c. `c(1,3,5,7)^c(8,4) - c(2,3,4,5)**c(12,3,5)- c(240,30, 76,98)/c(8,3)`
- d. `c(1,3,5,7)^c(8,9,4,7) - c(2,3,4,5)**c(12,3,14,5)- c(240,30, 76,98)/c(8,3,7)`

4. Which one of the following is the correct outcome of the command

```
sqrt(c(2,6,7,8)*c(4,2,2,4)+ c(2,4,2,4)*c(-2,2)+  
c(1,2,4,4)**c(9,2,4,1))?
```

- a. `2.236068 4.898979 16.309506 6.633250`
- b. `3.605551 4.898979 5.099020 6.633250`
- c. `3.605551 2.828427 16.552945 5.291503`
- d. `Error:...`

5. Which one of the following is the correct outcome of the command

```
c(1,2,1,2)*-c(1,2,-1,-2)**c(-1,-2,1,2)*-c(1,2,-1,-2)?
```

- a. `1 1 1 16`
- b. `1 1 1 -16`
- c. `-1 -16 1 16`
- d. `-1 -1 -1 -16`

6. Which one of the following is the correct outcome of the command

```
c(5,6,7,8) %/% 4 - c(64,64,24,24) %/% c(4,3,5,7) +  
c(153,646,817,788) %% c(71, 63) ?
```

- a. 4 4 33 31
- b. -4 -4 -33 -31
- c. -4 -4 33 31
- d. 4 4 -33 -31

7. Which one of the following is the correct outcome of the command

```
min(c(98,88,44,76)**c(-2,3))/min(c(49,44,22,38)^c(-2,-3)) -  
prod(c(1,2,1,2)^c(1,2))/max(c(9,4,4,9,26)*min(c(4,3,2,6)))  
+max(c(22,23,24,25)^c(2,3)) ?
```

- a. -16680439
- b. 16680439
- c. 15633.56
- d. None of these

8. Which one of the following is the correct outcome of the command?

```
round(prod(c(1,2,1,2)^c(1,2)) * sum(c(1,2,1,2)^c(2,3)) +  
prod(c(1,2,1,2)^c(1,2,3,7)) + ceiling(c(5,6,7,8)^c(2,3)))
```

- a. -199 -8 -175 288
- b. 825 1016 849 1312
- c. -825 1016 -849 1312
- d. None of these

9. Which one of the following is the correct outcome of the command

```
floor(c(5,6,7,8)^c(2,3))- ceiling(c(5,6,7,8)^c(2,3))+  
ceiling(c(2,3,4,5)^-c(1,-2)) - round(c(5,6,7,8)^c(2,3)) +  
floor(c(5,6,7,8)^c(2,3))?
```

a. 51 441 99 1049

b. 1 9 1 25

c. -1 -9 -1 -25

d. None of these

10. Which one of the following is the correct outcome of the command

```
ceiling(c(4,8,4,5)^-c(3,9) + sqrt(c(36,16,81,64)**c(2,3))*  
sqrt(c(2,3,4,5)^-c(1,2)) + round(c(4,7,8,9)*round(c(7,9))))  
?
```

a. 54 85 97 184

b. -2 -41 -15 22

c. 3 42 16 -21

d. None of these

11. Which one of the following is the correct outcome of the command

```
ceiling(prod(c(2,3,8,9))- sum(c(2,22,23,24,151))) -  
floor(prod(c(2,3,8,9))+ round(c(2,22,23,24,151))))?
```

a. -220 -200 -199 -198 -71

b. 640 620 619 618 491

c. -224 -244 -245 -246 -373

d. None of these

12. Which one of the following is the value of `x2` when the following commands are executed over the R console?

```
x1 = c(23,25,18,24)
```

```
x2 = sqrt(x1^2)+x1*6/x1^2-x1**(1/2)+abs(x1)
```

- a. 41.46504 45.24000 32.09069 43.35102
- b. 40.94330 44.76000 31.42403 42.85102
- c. -4.534962 -4.760000 -3.909307 -4.648979
- d. None of these

13. Which one of the following is the correct outcome of the command

```
c(21,22,24,34)^c(-3,-2,2,3)/c(4,3,2,1) +
c(2,4,1,6)%%c(3,4)*c(2,5,3,6) +
max(c(20,140,114,215)^c(1,2))/min(c(6,2,7,15)^c(2,2))?
```

- a. 5.000027 10.000689 293.000000 39315.000000
- b. 11556.25 -11551.25 11844.25 -50854.25
- c. 11556.25 11561.25 11844.25 50866.25
- d. None of these

14. Which one of the following is the correct command to obtain the following matrix?

$$x = \begin{pmatrix} 21 & 24 & 27 \\ 22 & 25 & 28 \\ 23 & 26 & 29 \end{pmatrix}$$

- a. `x=matrix(21:29,3,3,byrow=T)`
- b. `x=mat(21:29,3,3, byrow=T)`
- c. `x=matrix(21:29,3,3,byrow=F)`
- d. `x=mat(21:29,3,3,byrow=F)`

15. Which one of the following is the correct command to obtain the following matrix?

$$z = \begin{pmatrix} 15 & 11 & 17 \\ 8 & 19 & 12 \end{pmatrix}$$

- a. `z = matrix(nrow=2, ncol=3, data=c(15,11,17,8,19,12), byrow=T)`
- b. `z = matrix(nrow=3, ncol=2, data=c(15,8,11,19,17,12) , byrow=T)`
- c. `z = matrix(nrow=2, ncol=3, data=(15,8,11,19,17,12) , byrow=T)`
- d. `z = matrix(nrow=3, ncol=2, data=(15,8,11,19,17,12) , byrow=T)`

16. Which one of the following is the correct command to obtain the third column and first row of the following matrix?

$$x = \begin{pmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{pmatrix}$$

- a. `x(3, )` and `x( ,1)` respectively.
- b. `x(3, )` and `x(1, )` respectively.
- c. `x[, 3]` and `x[1, ]` respectively.
- d. `x[ ,1]` and `x[3, ]` respectively.

17. Which one of the following is the correct outcome of the command `x[3,2]` for the matrix constituted by the command `x=matrix(61:69,3,3,byrow=F)` ?

- a. 63
- b. 66
- c. 67
- d. 68

18. Which one of the following is the correct outcome of the commands `dim(x)` and `dim(y)` for the matrices obtained by `x=matrix(101:150,25,2,byrow=T)` and `y=matrix(201:250,25,2,byrow=F)` ?

- a. 25 2 and 2 25 respectively.
- b. 2 25 and 25 2 respectively.
- c. 2 25 and 2 25 respectively.
- d. 25 2 and 25 2 respectively.

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 3**

1. b

2. a

3. b

4. a

5. b

6. c

7. c

8. b

9. b

10. a

11. c

12. a

13. c

14. c

15. a

16. c

17. b

18. d

# MOOC Course - Foundations of R Software

July 2022

## Assignment 4

1. Which one of the following is the correct outcome of the command `rowSums (X)` for the matrix constituted by the command `X=matrix(61:69,3,3,byrow=F)`?

- a. 69 65 61
- b. 186 195 204
- c. 61 65 69
- d. 192 195 198

2. Which one of the following is the correct outcome of the command `colSums (X)` for the matrix constituted by the command `X=matrix(61:69,3,3,byrow=F)`?

- a. 69 65 61
- b. 186 195 204
- c. 61 65 69
- d. 192 195 198

3. Which one of the following is the correct outcome of the command `rowMeans (X)` for the matrix constituted by the command `X=matrix(61:69,3,3,byrow=F)`?

- a. 64 65 66
- b. 62 65 68
- c. 186 195 204
- d. 192 195 198

4. Which one of the following is the correct outcome of the command `colMeans(x)` for the matrix constituted by the command `x=matrix(61:69,3,3,byrow=F)` ?

- a. 64 65 66
- b. 62 65 68
- c. 186 195 204
- d. 192 195 198

5. Which one of the following is the correct outcome matrix of the command

`x = diag(5, nrow=2, ncol=2) ?`

a.  $x = \begin{pmatrix} 5 & 5 \\ 0 & 5 \end{pmatrix}$

b.  $x = \begin{pmatrix} 0 & 5 \\ 5 & 0 \end{pmatrix}$

c.  $x = \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix}$

d.  $x = \begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix}$

6. Which one of the following is the correct outcome of the command `t(x)` for

```
x=matrix(nrow=3, ncol=2, data=30:35, byrow=T) ?
```

a.

	[,1]	[,2]	[,3]
[1,]	30	32	34
[2,]	31	33	35

b.

	[,1]	[,2]
[1,]	30	31
[2,]	32	33
[3,]	34	35

c.

	[,1]	[,2]
[1,]	30	33
[2,]	31	34
[3,]	32	35

d.

	[,1]	[,2]	[,3]
[1,]	30	31	32
[2,]	33	34	35

7. Which one of the following is the correct command to obtain the multiplication of two square matrices **x** and **y** of the same order?

a. **x\*\*y**

b. **x\*%\*y**

c. **x%\*%y**

d. **x%\*\*\*%y**

8. Which one of the following is the correct command to obtain the multiplication of two matrices  $x = \begin{pmatrix} 4 & 7 \\ 8 & 2 \end{pmatrix}$  and  $y = \begin{pmatrix} 12 & 13 \\ 16 & 11 \end{pmatrix}$  along with its correct answer?

a. **x\*y**

and its correct answer is

[,1] [,2]  
[1,] 48 128  
[2,] 91 22

b. **x\*%\*y**

and its correct answer is

[,1] [,2]  
[1,] 48 91  
[2,] 128 22

c. **x%\*%y**

and its correct answer is

[,1] [,2]  
[1,] 160 129  
[2,] 128 126

d. **x%%\*%%y**

and its correct answer is

[,1] [,2]  
[1,] 160 128  
[2,] 129 126

9. Let `x= matrix(nrow=3, ncol=3, data=49:41, byrow=T)` then which one of the following is the correct outcome of `3*x` ?

a.

```
[,1] [,2] [,3]  
[1,] 147 144 141  
[2,] 138 135 132  
[3,] 129 126 123
```

b.

```
[,1] [,2] [,3]  
[1,] 147 138 129  
[2,] 144 135 126  
[3,] 141 132 123
```

c.

```
[,1] [,2] [,3]  
[1,] 141 132 123  
[2,] 144 135 126  
[3,] 147 138 129
```

d.

```
[,1] [,2] [,3]  
[1,] 129 126 123  
[2,] 138 135 132  
[3,] 147 144 141
```

10. Which one of the following is the correct command to obtain of the addition of two matrices  $x = \begin{pmatrix} 4 & 8 \\ 7 & 2 \end{pmatrix}$  and  $y = \begin{pmatrix} 12 & 16 \\ 13 & 11 \end{pmatrix}$  along with its correct answer?

a.

`x + y`

and its correct answer is

`[,1] [,2]`

`[1,] 16 24`

`[2,] 20 13`

b.

`x %+% y`

and its correct answer is

`[,1] [,2]`

`[1,] 16 24`

`[2,] 20 13`

c.

`x %%+%% y`

and its correct answer is

`[,1] [,2]`

`[1,] 20 13`

`[2,] 16 24`

d.

`x %+%+ y`

and its correct answer is

`[,1] [,2]`

`[1,] 20 13`

`[2,] 16 24`

11. Let `x= matrix(nrow=4, ncol=4, data=16:31, byrow=F)` then which one of the following is the correct outcome of `2+4%*%x` ?

a.

```
[,1] [,2] [,3] [,4]  
[1,] 66 80 96 112  
[2,] 68 84 100 116  
[3,] 72 88 104 120  
[4,] 76 92 108 124
```

b.

```
[,1] [,2] [,3] [,4]  
[1,] 66 82 98 114  
[2,] 70 86 102 118  
[3,] 74 90 106 122  
[4,] 78 94 110 126
```

c. Error...

d. None of these

12. Which one of the following is the correct outcome of `x[3 ,]` for the matrix specified by

```
x=matrix(nrow=3, ncol=3, data=c(10,20,30,40,50,60,70,80,90),  
byrow=F) ?
```

- a. [1] 30 60 90
- b. [1] 10 20 30
- c. [1] 40 50 60
- d. None of these

13. Which one of the following is the correct outcome of `x[1:2, 2:3]` for the matrix specified by `x= matrix(nrow=4, ncol=4, data=16:31, byrow=T)`?

a.

```
[,1] [,2]  
[1,] 21 22  
[2,] 17 18
```

b.

```
[,1] [,2]  
[1,] 17 18  
[2,] 21 22
```

c.

```
[,1] [,2]  
[1,] 17 21  
[2,] 18 22
```

d.

```
[,1] [,2]  
[1,] 17 17  
[2,] 17 17
```

14. Which one of the following is the correct command to get the matrix  $\begin{pmatrix} 20 & 30 \\ 50 & 60 \\ 80 & 90 \end{pmatrix}$  from the matrix specified by

```
X=matrix(nrow=4, ncol=3, data=c(10,20,30,40,50,60,70,80,90,  
100,110,120), byrow=T) ?
```

- a. `X[2:3, 1:3]`
- b. `X[1:3, 2:3]`
- c. `X[1:1, 2:2, 3:3]`
- d. `X[3:3, 2:2, 1:1]`

15. Which one of the following is the correct output of the command `cbind(x,y)` for the matrices specified by

```
x=matrix(nrow=3, ncol=2, data=30:35, byrow=T)
```

```
y=matrix(nrow=3, ncol=2, data=c(10,20,30,40,50,60), byrow=T) ?
```

a.

```
[,1] [,2]  
[1,] 30 31  
[2,] 32 33  
[3,] 34 35  
[4,] 10 20  
[5,] 30 40  
[6,] 50 60
```

b.

```
[,1] [,2] [,3] [,4]  
[1,] 10 20 30 31  
[2,] 30 40 32 33  
[3,] 50 60 34 35
```

c.

```
[,1] [,2] [,3] [,4]  
[1,] 30 31 10 20  
[2,] 32 33 30 40  
[3,] 34 35 50 60
```

d.

```
[,1] [,2]  
[1,] 10 20  
[2,] 30 40  
[3,] 50 60  
[4,] 30 31  
[5,] 32 33  
[6,] 34 35
```

16. Which one of the following is the correct output of the command `rbind(y, x)` for the matrices specified by

```
x=matrix(nrow=3, ncol=2, data=30:35, byrow=T)
```

```
y=matrix(nrow=3, ncol=2, data=c(10,20,30,40,50,60), byrow=T) ?
```

a.

```
[,1] [,2]  
[1,] 30 31  
[2,] 32 33  
[3,] 34 35  
[4,] 10 20  
[5,] 30 40  
[6,] 50 60
```

b.

```
[,1] [,2] [,3] [,4]  
[1,] 10 20 30 31  
[2,] 30 40 32 33  
[3,] 50 60 34 35
```

c.

```
[,1] [,2] [,3] [,4]  
[1,] 30 31 10 20  
[2,] 32 33 30 40  
[3,] 34 35 50 60
```

d.

```
[,1] [,2]  
[1,] 10 20  
[2,] 30 40  
[3,] 50 60  
[4,] 30 31  
[5,] 32 33  
[6,] 34 35
```

17. If  $x = \begin{pmatrix} 13 & 1 & 12 & 5 \\ 4 & 20 & 14 & 6 \\ 5 & 3 & 16 & 7 \\ 16 & 8 & 12 & 8 \end{pmatrix}$  then which one of the following is the correct

command and its outcome for obtaining the inverse of the sub-matrix of `x` formed by first, second, and third rows and second, third, and fourth columns?

a.

```
solve(x[1:3,2:4])=
```

	[,1]	[,2]	[,3]
[1,]	-0.02777778	0.05555556	-0.02777778
[2,]	1.69444444	0.11111111	-1.30555556
[3,]	-3.86111111	-0.27777778	3.13888889

b.

```
inv(x[1:3,2:4])=
```

	[,1]	[,2]	[,3]
[1,]	-0.02777778	0.05555556	-0.02777778
[2,]	1.69444444	0.11111111	-1.30555556
[3,]	-3.86111111	-0.27777778	3.13888889

c.

```
inv(x[1:3,2:4])=
```

	[,1]	[,2]	[,3]
[1,]	-0.02777778	1.69444444	-3.86111111
[2,]	0.05555556	0.11111111	-0.27777778
[3,]	-0.02777778	-1.30555556	3.13888889

d.

```
solve((x[1:3,2:4]))=
```

	[,1]	[,2]	[,3]
[1,]	-0.02777778	1.69444444	-3.86111111
[2,]	0.05555556	0.11111111	-0.27777778

```
[3,] -0.02777778 -1.3055556  3.1388889
```

18. Which one of the following is the correct outcome of the command

`(x < 15) || (x > 12) & (x < 15) && (x > 12) || (x == 17)` when  
`x = 13` and when `x = -13` ?

- a. `FALSE` and `FALSE` respectively.
- b. `TRUE` and `TRUE` respectively.
- c. `FALSE` and `TRUE` respectively.
- d. `TRUE` and `FALSE` respectively.

19. Suppose `x = 13:16` . Then which one of the following is the correct outcome of the command

`(x > 13) & (x < 15) ?`

- a. `FALSE TRUE FALSE FALSE`
- b. `TRUE FALSE TRUE TRUE`
- c. `TRUE TRUE TRUE FALSE`
- d. `FALSE FALSE FALSE TRUE`

20. Suppose `x = 3:18` . Then which one of the following correctly specifies the outcome of the following statement: `x[(x > 12) & (x < 17)]` and `x[(x > 15) | (x < 5)]` ?

- a. are different for both as `13 14 15 16` and `3 4 16 17 18` respectively.
- b. are the same for both as `13 14 15 16`
- c. are different for both as `5 6 7 8` and `13 14 15 16` respectively.

d. are the same for both as 5 6 7 8

21. Suppose `x = 10:50`. Then which one of the following is the correct command to know that which of the values in `x` are more than 30 and less than 40?

- a. `x[(x > 30) & (x < 40)]`
- b. `(x > 40) & (x < 50)`
- c. `x[(x >= 40) & (x >= 40)]`
- d. `x[(x <= 40) & (x <= 40)]`

22. Suppose `x = 45:80`. Then which one of the following is the correct outcome of `(x > 80) && (x < 70)` and `(x > 80) || (x < 70)` ?

- a. `TRUE` and `FALSE` respectively.
- b. `FALSE` and `FALSE` respectively.
- c. `TRUE` and `TRUE` respectively.
- d. `FALSE` and `TRUE` respectively.

23. Suppose `x = 13:17` then which one of the following is the correct outcome of `x[(x > 12) || (x < 15)]` ?

- a. `TRUE`
- b. `FALSE`
- c. 13 14 15 16 17

d. 17 16 15 14 13

24. Suppose `x = c(10,20,30,40,50)` and `y = c(40,50,60,70,80)` then which one of the following is the correct outcome of `x != y` ?

a. TRUE TRUE TRUE TRUE TRUE

b. FALSE FALSE FALSE FALSE FALSE

c. TRUE

d. FALSE

25. Suppose `x = c(10,20,30,40,50)` and `y = c(40,50,60,70,80)` then which one of the following is the correct outcome of `x == y` ?

a. TRUE TRUE TRUE TRUE TRUE

b. FALSE FALSE FALSE FALSE FALSE

c. TRUE

d. FALSE

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 4**

1. d
2. a
3. b
4. a
5. d
6. a
7. c
8. c
9. a
10. a
11. c
12. a
13. b
14. b
15. c
16. d
17. a
18. b
19. a
20. a
21. a
22. d
23. c
24. a
25. b

## MOOC Course - Foundations of R Software

July 2022

### Assignment 5

1. Suppose `x` is any vector as `x=c(10, 80, 90, 101:200, NA)` then which one of the following is the correct outcome of the command `mean(x, na.rm=FALSE)`?

a. `147.8641`

b. `96.2`

c. `Error...`

d. `NA`

2. Suppose `x` is any vector as `x=c(10, 80, 90, 101:200, NA)` then which one of the following is the correct outcome of the command `mean(x, na.rm=TRUE)`?

a. `147.8641`

b. `96.2`

c. `Error...`

d. `NA`

3. Suppose `x` is any vector as `x=c(10, 80, 90, 101:200, NA)` then which one of the following is the correct outcome of the command  
`mean(na.omit(x))`?

a. **147.8641**

b. **96.2**

c. **Error...**

d. **NA**

4. If `x = 23` then which one of the following is the correct outcome of `y` where

`y=if ( x==23 ) { x = x^3+3 } else { x = x^2-3 } ?`

a. **12164**

b. **12170**

c. **526**

d. **123**

5. Which one of the following is the correct syntax to compute  $y$  for the following function?

$$y = \begin{cases} 2x^{3/2} + |x^4 - 66| & \text{if } x \leq 0 \\ 3\exp(x^3) - \sqrt{x^{5/2}} & \text{if } x > 0 \end{cases}$$

- a. `if ( x <= 0 ) [ y = 2 * x^(3/2) + abs(x^4 - 66) ] else [ y = 3 * exp(x^3) - sqrt(x^(5/2)) ]`
- b. `if ( x <= 0 ) ( y = 2 * x^(3/2) + abs(x^4 - 66) ) else ( y = 3 * exp(x^3) - sqrt(x^(5/2)) )`
- c. `if ( x <= 0 ) { y = 3 * exp(x^3) - sqrt(x^(5/2)) } else { y = 2 * x^(3/2) + abs(x^4 - 66) }`
- d. `if ( x <= 0 ) ( y = 3 * exp(x^3) - sqrt(x^5/2)) else ( y = 2 * x^3/2 + abs(x^4 - 66) )`

6. Suppose `x = c(12, 4, 16, 8, 20, 5, 14)` then which one of the following is the correct output of

`ifelse(x < 10, 2+x, 2*x) ?`

- a. [1] 14 8 18 16 22 10 16
- b. [1] 14 6 18 10 22 7 16
- c. [1] 24 8 32 16 40 10 28
- d. [1] 24 6 32 10 40 7 28

7. Which one of the following is the correct outcome of the command

`switch(2, "UP", "MP", "J&K", "Bihar", 20, 40, 60+40)?`

- a. **100**
- b. **"UP"**
- c. **"MP"**
- d. **40**

8. Which one of the following is the correct outcome of the command

`switch(7, "UP", "MP", "J&K", "Bihar", 20, 40, 60+40)?`

- a. **100**
- b. **"UP"**
- c. **60+40**
- d. **40**

9. Which one of the following is the correct outcome of the command

`switch("S3", "S1"= "UP", "S2" = "MP", "S3" = "J&K", "S4" = "Bihar", N1 = 20, N2 = 40, N3 = 60+40)?`

- a. **"S3"**
- b. **"MP"**
- c. **"J&K"**
- d. **100**

10. Which one of the following is the correct outcome of the command

```
switch("N3", "S1"= "UP", "S2" = "MP", "S3" = "J&K", "S4" =  
"Bihar", N1 = 20, N2 = 40, N3 = 60+40) ?
```

- a. **"S3"**
- b. **"MP"**
- c. **"J&K"**
- d. **100**

11. Suppose **x** is any vector as **x=c("UP", "MP", "J&K", "Bihar", 20,  
40, 60+40)** then which one of the following is the correct outcome of the  
command **which(x == 100)** ?

- a. **60+40**
- b. **7**
- c. **Error...**
- d. **NA**

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 5**

1. d
2. a
3. a
4. b
5. b
6. d
7. c
8. a
9. c
10. d
11. b

# MOOC Course - Foundations of R Software

July 2022

## Assignment 6

1. Which one of the following is the correct outcome of

```
for(i in 12:14) {print(1 - 2i + i^2)} ?
```

a.

```
[1] 121
```

```
[1] 144
```

```
[1] 169
```

b.

```
[1] -23 + i2
```

```
[1] -25 + i2
```

```
[1] -27 + i2
```

c.

```
[1] 145-2i
```

```
[1] 170-2i
```

```
[1] 197-2i
```

d.

```
[1] 169
```

```
[1] 144
```

```
[1] 121
```

2. Which one of the following is the correct outcome of

```
for(i in 10:14) {print(i + 2*i + 3*i**3)} ?
```

a.

[1] i + 2\*i + 3\*i\*\*3

b.

[1] 3030

[1] 4026

[1] 5220

[1] 6630

[1] 8274

c.

[1] 120

[1] 132

[1] 144

[1] 156

[1] 168

d. **Error...**

3. Which one of the following is the correct outcome of the following commands about while loop?

```
y = 5

while (y < 15) {
  print(c("R Course", "through NPTEL is helpful."))
  y = y + 1
  print(c("R Course", "through NPTEL is not helpful."))
  y = y + 3
}
```

a.

```
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is helpful."
```

b.

```
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is helpful."
```

c.

```
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is not helpful."
```

d.

```
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is not helpful."
[1] "R Course"      "through NPTEL is not helpful."
```

4. Which one of the following is the correct outcome of the following while loop?

`x = 20`

`while(x < 35) {x = x^2+10; if (x == 30) break; print(x); }`

a.

`[1] 410`

b.

`[1] 20`

c.

`[1] 410`

`[1] 410`

d.

`[1] 20`

`[1] 20`

`[1] 35`

`[1] 35`

5. Which one of the following is the correct specification to compute

$$y = \sqrt{\exp(x) + x^2} + x^{3/2} - 4000 \text{ and what is its value for } x = 17 ?$$

- a. `y<-function(x){sqrt(exp(x)+x^2)+x^3/2-4000}, y=3371.298`
- b. `y<-f(x) [sqrt(exp(x)+x^2)+x^(3/2)-4000] , y=984.891`
- c. `y<-function(x) {sqrt(exp(x)+x^2)+x^3/2-4000}, y=3371.298`
- d. `y<- function(x) (sqrt(exp(x)+x^2)+x^(3/2)-4000), y  
=984.891`

6. Which one of the following is the correct outcome of `z(14,11)` of the function spe

specified as `z=function(x,y) {sqrt(x^3-y^3+log(x))+log(-(5*x^-2+6*y^-2-60))-(x^3-y^2)^(3/4)}` ?

- a. `-4511644550`
- b. `-324.8028`
- c. `-2758.699`
- d. None of these

7. Which one of the following is the correct outcome of the command

`abs(seq(-30,-35))`?

- a. `[1] 30 31 32 33 34 35`
- b. `[1] -30 -31 -32 -33 -34 -35`
- c. `[1] 35 34 33 32 31 30`
- d. `[1] -35 -34 -33 -32 -31 -30`

8. Which one of the following is the correct output of the command

`sqrt(abs(seq(-15,15, by = 6)))` ?

a. [1] -3.872983 -3.000000 -1.732051 -1.732051 3-.000000 -3.872983

b. [1] 3.872983 3.000000 1.732051 1.732051 3.000000 3.872983

c. [1] 1.732051 3.000000 3.872983 1.732051 3.000000 3.872983

d. [1] 1.732051 3.000000 3.872983 3.872983 3.000000 1.732051

9. Which one of the following is the correct output of the command `seq(20,-20, by = -4)` ?

a. [1] -20 -16 -12 -8 -4 0 4 8 12 16 20

b. [1] -20 -16 -12 -8 -4 0

c. [1] 17 12 7 3 -3 -7 -12 -17

d. [1] 20 16 12 8 4 0 -4 -8 -12 -16 -20

10. Which one of the following is the correct output of the command

`seq(to = 200, length = 10)` ?

a. [1] -209 -208 -207 -206 -205 -204 -203 -202 -201 -200

b. [1] 191 192 193 194 195 196 197 198 199 200

c. [1] -191 -192 -193 -194 -195 -196 -197 -198 -199 -200

d. [1] 200 201 202 203 204 205 206 207 208 209

11. Which one of the following is the correct output of the command `seq(to = -100, length = 10)` ?

- a. [1] -109 -108 -107 -106 -105 -104 -103 -102 -101 -100
- b. [1] 109 108 107 106 105 104 103 102 101 100
- c. [1] 100 101 102 103 104 105 106 107 108 109
- d. [1] -100 -101 -102 -103 -104 -105 -106 -107 -108 -109

12. Which one of the following is the correct output of the command

`seq(to = 112, length = 11, by = 12)` ?

- a. [1] 4 16 28 40 52 64 76 88 100 112 114
- b. [1] 114 112 100 88 76 64 52 40 28 16 4
- c. [1] -8 4 16 28 40 52 64 76 88 100 112
- d. [1] 112 100 88 76 64 52 40 28 16 4 -8

13. Which one of the following is the correct output of the command

`seq(from = -5, length = 5, by = -0.5)` ?

- a. [1] -5.0 -4.5 -4.0 -3.5 -3.0
- b. [1] -5.0 -5.5 -6.0 -6.5 -7.0
- c. [1] 3.0 3.5 4.0 4.5 5.0
- d. [1] 7.0 6.5 6.0 5.5 5.0

14. Which one of the following is the correct output of `y` for the following commands

```
X = c(78, 67, 15, 120, 3, 160, 20, 48, 9)  
Y = seq(along = X) ?
```

a. [1] 1 2 3 4 5 6 7 8 9

b. [1] 7 6 3 8 1 9 4 5 2

c. [1] 2 5 4 9 1 8 3 6 7

d. [1] 9 8 7 6 5 4 3 2 1

15. Which one of the following is the correct output of `y[x[3]]` and `y[x[7]]` for the command

```
Y = c(90, 17, 51, 80, 32, 60, 87)
```

```
X = c(5, 7, 1, 4, 6, 3, 2) ?
```

a. 51 and 87 respectively.

b. 1 and 2 respectively.

c. 17 and 90 respectively.

d. 90 and 17 respectively.

16. Which one of the following is the correct outcome of the command `x[(x>50)]`  
where

`x = c(20, 65, 27, 55, 38, 37, 18, 52, 160, 237, 170, 61, 88, 52, 49, 12, 4, 24, 18, 22) ?`

a. [1] 65 55 52 160 237 170 61 88 52

b. [1] 9

c. [1] TRUE

d. [1] FALSE

17. Which one of the following is the correct outcome of the command

`x[(x - 50 > 30)]` where

`x = c(20, 65, 27, 55, 38, 37, 18, 52, 160, 237, 170, 61, 88, 52, 49, 12, 4, 24, 18, 22) ?`

a. [1] 65 55 38 37 52 160 237 170 61 88 52 49

b. [1] 160 237 170 88

c. [1] NULL

d. [1] 12

18. Which one of the following is the correct outcome of the command

```
x[(x^2 + 6*x < 500)] where  
x = c(20, 65, 27, 55, 38, 37, 18, 52, 160, 237, 170, 61, 88,  
52, 49, 12, 4, 24, 18, 22) ?
```

- a. [1] 20 65 27 55 38 37 52 160 237 170 61 88 52  
49 24 22
- b. [1] 20, 65, 27, 55, 38, 37, 18, 52, 160, 237, 170, 61, 88,  
52, 49, 12, 4, 24, 18, 22
- c. [1] 18 12 4 18
- d. [1] NULL

19. If `y = 15:25` then which one of the following is the correct outcome of the command `y[-(3:10)]` ?

- a. [1] 17 18 19 20 21 22 23 24
- b. [1] 15 16 25
- c. [1] 18 19 20 21 22 23 24 25
- d. [1] -25 -16 -15

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 6**

1. c
2. b
3. c
4. a
5. d
6. b
7. a
8. b
9. d
10. b
11. a
12. c
13. b
14. a
15. d
16. a
17. b
18. c
19. b

## MOOC Course - Foundations of R Software

July 2022

### Assignment 7

1. Which one of the following is the correct command to generate a sequence of three yearly dates 1-July-2020, 1- July-2021 and 1- July-2022?

- a. `seq(as.Date("2020-03-01"), as.Date("2022-03-01"), by = "years")`
- b. `seq(as.date("2020-03-01"), as.date("2021-03-01"), as.date("2022-03-01"), by = "years")`
- c. `seq(as.Date(`2020-03-01`), as.Date(`2022-03-01`), by = `1`)`
- d. `seq(As.Date("2020-03-01"), As.Date("2022-03-01"), by = "years")`

2. Which one of the following is the correct command to generate the dates of 6 consecutive months starting from 15 March 2022?

- a. `seq(As.Date("2022-03-15"), by = "months", length = 6)`
- b. `seq(as.date("2022-03-15"), by = "months", length = 6)`
- c. `seq(as.Date("2022-03-15"), by = "months", length = 6)`
- d. `seq(as.Date("2022-03-15") to as.Date("2022-08-15") by = "30")`

3. Which one of the following is the correct outcome of the command `letter[2:7]`?
- a. [1] "b" "c" "d" "e" "f" "g"
  - b. [1] "B" "C" "D" "E" "F" "G"
  - c. [1] "2" "3" "4" "5" "6" "7"
  - d. Error...
4. Which one of the following is the correct outcome of the command `LETTERS[15:10]`?
- a. [1] "o" "n" "m" "l" "k" "j"
  - b. [1] "O" "N" "M" "L" "K" "J"
  - c. [1] "J" "K" "L" "M" "N" "O"
  - d. [1] "j" "k" "l" "m" "n" "o"
5. Which one of the following is the correct outcome of the command `rep(5:9,3)`?
- a. [1] 5 5 5 6 6 6 7 7 7 8 8 8 9 9 9
  - b. [1] 5 6 6 7 7 7 8 8 8 8 9 9 9 9
  - c. [1] 5 6 7 8 9 5 6 7 8 9 5 6 7 8 9
  - d. [1] 9 8 7 6 5 9 8 7 6 5 9 8 7 6 5

6. Which one of the following is the correct outcome of the command

`rep(50:46, times=4) ?`

- a. [1] 50 49 48 47 46 50 49 48 47 46 50 49 48 47 46 50 49 48 47 46
- b. [1] 46 47 48 49 50 46 47 48 49 50 46 47 48 49 50 46 47 48 49 50
- c. [1] 50 50 50 50 49 49 49 49 48 48 48 48 47 47 47 47 46 46 46 46
- d. [1] 46 46 46 46 47 47 47 47 48 48 48 48 49 49 49 49 50 50 50 50

7. Which one of the following is the correct outcome of the command `rep(35:40, each=4) ?`

- a. [1] 35 36 37 38 39 40 35 36 37 38 39 40 35 36 37 38 39 40 35 36 37 38 39 40
- b. [1] 35 35 35 35 36 36 36 36 37 37 37 37 37 38 38 38 38 38 39 39 39 39 39 40 40 40 40
- c. [1] 40 40 40 40 39 39 39 39 38 38 38 38 37 37 37 37 36 36 36 36
- d. [1] 40 39 38 37 36 35 40 39 38 37 36 35 40 39 38 37 36 35 40 39

8. Which one of the following is the correct outcome of the commands

```
x = matrix(nrow=3, ncol=3, data=1:9, byrow=F)  
rep(x, each=2) ?
```

- a. [1] 9 9 8 8 7 7 6 6 5 5 4 4 3 3 2 2 1 1
- b. [1] 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9
- c. [1] 9 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1
- d. [1] 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

9. Which one of the following is the correct outcome of the command `rep(c("CLASS1", " CLASS2", " CLASS3"), each=3)` ?

- a. [1] "CLASS1" "CLASS1" "CLASS1" " CLASS2" " CLASS2" " CLASS2"  
" CLASS3" " CLASS3" " CLASS3"
- b. [1] "CLASS1" "CLASS2" "CLASS3" " CLASS1" " CLASS2" " CLASS3"  
" CLASS1" " CLASS2" " CLASS3"
- c. [1] CLASS 1 2 3 CLASS 1 2 3
- d. [1] CLASS 1 1 CLASS 2 2 CLASS 3 3

10. Which one of the following is the correct outcome of the command

```
sort(c(15,55,18,32,97,71,82,42,66), decreasing = FALSE) ?
```

- a. [1] 1 5 2 3 9 7 8 4 6
- b. [1] 6 4 8 7 9 3 2 5 1
- c. [1] 15 18 32 42 55 66 71 82 97
- d. [1] 97 82 71 66 55 42 32 18 15

11. Which one of the following is the correct outcome of the command

```
sort(c(15,55,18,32,97,71,82,42,66), decreasing = TRUE) ?
```

- a. [1] 1 5 2 3 9 7 8 4 6
- b. [1] 6 4 8 7 9 3 2 5 1
- c. [1] 15 18 32 42 55 66 71 82 97
- d. [1] 97 82 71 66 55 42 32 18 15

12. Which one of the following is the correct outcome of the command

```
order(c(15,55,18,32,97,71,82,42,66), decreasing = FALSE) ?
```

- a. [1] 15 18 32 42 55 66 71 82 97
- b. [1] 97 82 71 66 55 42 32 18 15
- c. [1] 1 3 4 8 2 9 6 7 5
- d. [1] 5 7 6 9 2 8 4 3 1

13. Which one of the following is the correct outcome of the command

```
mode(c(10, 20, "30", 80+90, "70+90", 16.7, 10*5))?
```

a. character

b. numeric

c. list

d. data frame

14. Which one of the following is the correct outcome of the command `x[[3]]` where,

```
x =list (c("class1", "class2", "class3"), seq(from=50, to=57),  
rep(28:30, each=3)) ?
```

a. [1] "class1" "class2" "class3"

b. [1] 50 51 52 53 54 55 56 57

c. [1] 28 28 28 29 29 29 30 30 30

d. [1] "class3" 52 28

15. Which one of the following is the correct outcome of the command `x[[2]][2]`

```
where x =list (c("class1", "class2", "class3"), seq(from=50, to=57),  
rep(28:30, each=3)) gives an output as
```

a. [1] "class2"

b. [1] 51

c. [1] 29

d. [1] 50 51 52 53 54 55 56 57

16. Which one of the following is the correct outcome of the command `x[2][2]` where

```
x = list (c("class1", "class2", "class3"), seq(from=50, to=57),  
rep(28:30, each=3)) ?
```

- a. "class2"
- b. 51
- c. 29
- d. "NULL"

17. Which one of the following is the correct outcome of the command `x[3]` where

```
x = list (c("class1", "class2", "class3"), seq(from=50, to=57),  
rep(28:30, each=3)) ?
```

- a. [1] "class3"
- b. [1] 28 28 28 29 29 29 30 30 30
- c. [1] 52
- d. [1] 50 51 52 53 54 55 56 57

18. Consider the list `z = list(x1 = "class", x2 = 20:25)`. Which one of the following is the correct outcome of the command `z["x2"]` ?

- a. [1] "class"
- b. [1] "20:25"
- c. [1] 25 24 23 22 21 20
- d. [1] 20 21 22 23 24 25

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 7**

1. a
2. c
3. d
4. b
5. c
6. a
7. b
8. b
9. a
10. c
11. d
12. c
13. a
14. c
15. b
16. d
17. b
18. d

## MOOC Course - Foundations of R Software

July 2022

### Assignment 8

1. Consider the list `x=list("UP", "MP", "J&K", "Bihar", 20, 40, "60+40")`. Which one of the following is the correct outcome of the command `append(x, 200)` ?

a.

`[[1]]  
[1] 200`

`[[2]]  
[1] "UP"`

`[[3]]  
[1] "MP"`

`[[4]]  
[1] "J&K"`

`[[5]]  
[1] "Bihar"`

`[[6]]  
[1] 20`

`[[7]]  
[1] 40`

`[[8]]  
[1] "60+40"`

b.

```
[[1]]
[1] "200"

[[2]]
[1] "UP"

[[3]]
[1] "MP"

[[4]]
[1] "J&K"

[[5]]
[1] "Bihar"

[[6]]
[1] 20

[[7]]
[1] 40

[[8]]
[1] 100
```

c.

```
[[1]]
[1] "UP"

[[2]]
[1] "MP"

[[3]]
[1] "J&K"

[[4]]
[1] "Bihar"

[[5]]
[1] 20

[[6]]
[1] 40

[[7]]
[1] "60+40"

[[8]]
[1] 200
```

d.

```
[[1]]
[1] "UP"

[[2]]
[1] "MP"

[[3]]
[1] "J&K"

[[4]]
[1] "Bihar"

[[5]]
[1] 40

[[6]]
[1] 20

[[7]]
[1] "60+40"

[[8]]
[1] "200"
```

2. Consider the list `x=list("UP", "MP", "J&K", "Bihar", 20, 40, "60+40")`. Which one of the following is the correct outcome of the command `append(x, 200, after =2)` ?

a.

`[[1]]  
[1] "UP"`

`[[2]]  
[1] "MP"`

`[[3]]  
[1] 200`

`[[4]]  
[1] "J&K"`

`[[5]]  
[1] "Bihar"`

`[[6]]  
[1] 20`

`[[7]]  
[1] 40`

`[[8]]  
[1] "60+40"`

b.

`[[1]]  
[1] "UP"`

`[[2]]  
[1] "MP"`

`[[3]]  
[1] "200"`

`[[4]]  
[1] "J&K"`

`[[5]]  
[1] "Bihar"`

`[[6]]`

```
[1] 20  
[[7]]  
[1] 20  
[[8]]  
[1] "60+40"
```

c.

```
[[1]]  
[1] "UP"  
  
[[2]]  
[1] "MP"  
  
[[3]]  
[1] "J&K"  
  
[[4]]  
[1] "Bihar"  
  
[[5]]  
[1] 20
```

```
[[6]]  
[1] 40  
  
[[7]]  
[1] "60+40"  
  
[[8]]  
[1] 200
```

d.

```
[[1]]  
[1] "UP"  
  
[[2]]  
[1] "MP"  
  
[[3]]  
[1] "J&K"  
  
[[4]]  
[1] "Bihar"
```

```
[[5]]  
[1] 20  
  
[[6]]  
[1] 40  
  
[[7]]  
[1] "60+40"  
  
[[8]]  
[1] "200"
```

3. Consider the list `x=list("UP", "MP", "J&K", "Bihar", 20, 40, "60+40")`.

Which one of the following is the correct outcome of the command `x[2:3]` ?

a.

```
[[1]]
```

```
[[6]]  
[1] 20
```

```
[[7]]  
[1] 40
```

b.

```
[[1]]  
[1] "MP"
```

```
[[2]]  
[1] "J&K"
```

C.

```
[2] "MP" [3] "J&K"
```

d.

```
[[1]]  
[1] "UP"
```

```
[[2]]  
[1] "MP"
```

```
[[3]]  
[1] "J&K"
```

```
[[4]]  
[1] "Bihar"
```

```
[[5]]  
[1] 20
```

```
[[6]]  
[1] 40
```

```
[[7]]  
[1] "60+40"
```

```
[[8]]  
[1] "200"
```

4. Consider the list `z = list(x1 = "class", x2 = 20:25)`. Which of the following is the correct command to change the element `x2` by `y2`?

- a. `names(z)[2]= y2`
- b. `names(z)[2]= "y2"`
- c. `names(y)[2]= "z2"`
- d. `names(y)[2]= z2`

5. Which one of the following is the correct outcome of the command

```
factor(c(10,10,20,20,30,30)) ?
```

a.

```
[1] 10 10 20 20 30 30
```

```
Levels: 10 20 30
```

b.

```
[1] 10 20 30
```

```
Levels: 10 20 30
```

c.

```
[1] 10 20 30
```

```
Levels: 10 10 20 20 30 30
```

d.

```
[1] 10 10 20 20 30 30
```

```
Levels: 10 10 20 20 30 30
```

6. Which one of the following is the correct outcome of the following commands?

```
data = c(4,4,2,2,6,6,9)
```

```
factor(data)
```

```
levels(data) = c('A','B','C','D')
```

```
data
```

a.

```
[1] 4 4 2 2 6 6 9
```

```
attr(),"levels")
```

```
[1] "A" "B" "C" "D"
```

b.

```
[1] 4 2 6 9
```

```
attr(),"levels")
```

```
[1] "A"    "B"    "C"    "D"
```

c.

```
[1] A A B B C C D  
Levels: 1 2 3
```

d.

```
[1] B A C D  
attr(,"levels")  
[1] "2"    "4"    "6"    "9"
```

7. Which one of the following is the correct outcome of the command

```
x = factor(c(10,20,20,50,10,20,10,50,60,60),  
levels=c(10,20,50,60),ordered=TRUE) ?
```

a.

```
[1] 10 < 20 < 50 < 60  
Levels: 10 20 20 50 10 20 10 50 60 60
```

b.

```
[1] 10 20 50 60  
Levels: 10 20 20 50 10 20 10 50 60 60
```

c.

```
[1] 10 20 20 50 10 20 10 50 60 60  
Levels: 10 < 20 < 50 < 60
```

d.

```
[1] 10 20 20 50 10 20 10 50 60 60  
Levels: 1 < 2 < 5 < 6
```

8. Which one is the correct outcome of the command

```
factor( c(rep("head",3), rep("tale", 4))) ?
```

a.

```
[1] head tale  
Levels: head head head tale tale tale tale
```

b.

```
[1] head head head head tale tale tale  
Levels: head tale
```

c.

```
[1] tale tale tale tale head head head  
Levels: head tale
```

d.

```
[1] head head head tale tale tale tale  
Levels: head tale
```

9. Which one of the following is the correct outcome of the command

```
unclass(factor( c("high school", "graduation", "post graduation", "graduation", "high school", "post graduation", "high school", "post graduation"), levels=c("high school", "graduation", "post graduation") )) ?
```

a.

```
[1] 1 2 3 2 1 3 1 3  
attr(,"levels")  
[1] "high school"      "graduation"      "post graduation"
```

b.

```
[1] 3 2 1 2 3 1 3 1  
attr(,"levels")  
[1] "post graduation"    "graduation"      "high school"
```

c.

```
[1] 1 2 3  
attr(,"levels")  
[1] "high school"      "graduation"      "post graduation"
```

d.

```
[1] 3 2 1  
attr(,"levels")  
[1] "post graduation"    "graduation"      "high school"
```

10. Which one of the following is the correct outcome of the command

`as.factor(c(1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5) ) ?`

a.

`[1] 1 2 3 4 5`

`Levels: 1 2 3 4 5`

b.

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

`Levels: 5 4 3 2 1`

c.

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

`Levels: 1 2 3 4 5`

d.

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

`Levels: 5 4 3 2 1`

11. Which one of the following is the correct outcome of the command

`print(2/3,digits=6) ?`

a. `[1] 0.66667`

b. `[1] 0.6667`

c. `[1] 0.666667`

d. `[1] 0.667`

12. Which one of the following is the correct outcome of the command

```
print("The average marks of the class is", (40+10)/3, "in first year.", digits=5) ?
```

- a. [1] "The average marks of the class is" 16.667 "in first year."
- b. [1] The average marks of the class is 16.667 in first year.
- c. [1] "The average marks of the class is"
- d. [1] The average marks of the class is 16.66667 in first year.

13. Which one of the following is the correct outcome of the command

```
print("My house has"); print(2+4); print("rooms.") ?
```

- a. [1] "My house has" 6 "rooms."
- b.
- [1] "My house has"
- [1] 6
- [1] "rooms."
- c. [1] "My house has" 2+4 "rooms."
- d. [1] My house has 6 rooms.

14. Which one of the following is the correct outcome of the command

```
format(12345678910, big.mark = "*") ?
```

- a. "1\*2\*3\*4\*5\*6\*7\*8\*9\*1\*0"
- b. "123\*45\*678\*910"
- c. "12\*34\*56\*78\*91\*0"
- d. "12\*345\*678\*910"

15. Which one of the following is the correct outcome of the command

`print( format( 2.5, digits=6, nsmall=10 ) ) ?`

- a. "2.5"
- b. "2.5000"
- c. "2.5000000000"
- d. "2.500000"

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 8**

1. c

2. a

3. b

4. b

5. a

6. a

7. c

8. d

9. a

10. c

11. c

12. c

13. b

14. d

15. c

## **MOOC Course - Foundations of R Software**

**July 2022**

### **Assignment 9**

1. Let `x <- 'Indian Institute of Technology Kanpur!\\n'` then which one of the following are the respective correct outcome of the commands `print(x)` and `cat(x)`?

a.

`[1] Indian Institute of Technology Kanpur!\\n`

and

`[1] Indian Institute of Technology Kanpur!\\n`

b.

`[1] "Indian Institute of Technology Kanpur!"`

and

`Indian Institute of Technology Kanpur!\\n`

c.

`[1] "Indian Institute of Technology Kanpur!"`

and

`Indian Institute of Technology Kanpur!\\n`

d.

`[1] "Indian Institute of Technology Kanpur!\\n"`

and

`Indian Institute of Technology Kanpur!`

2. Which one of the following is the correct outcome of the command

```
print(paste("R course has", 10* 2:4, "students"))?
```

- a. [1] "R course has" 10\* 2 "students" "R course has" 10\* 3 "students" "R course has" 10\* 4 "students"
- b. [1] "R course has 20 students" "R course has 30 students" "R course has 40 students"
- c. [1] "R course has" 10\*2 10\*3 10\*4 "students"
- d. [1] R course has 20 students R course has 30 students R course has 40 students

3. Which one of the following is the correct outcome of the commands

```
x <- 36
```

```
y <- 27
```

```
cat("The square root of", "x", "is", sqrt(x), "and the cube  
root of y is", y^(1/3), "\n") ?
```

- a. The square root of 36 is 6 and the cube root of 27 is 9 !
- b. The square root of x is sqrt(x) and the cube root of y is y^(1/3) !
- c. "The square root of" "x" "is" 6 " and the cube root of y is" 3 !\n
- d. The square root of x is 6 and the cube root of y is 3 !

4. Which one of the following is the correct outcome of the command

```
x <- 7  
cat("The approximate value of square root of", x, "is", format(  
sqrt(x),digits=3),"\n") ?
```

- a. "The approximate value of square root of 7 is 2.65"
- b. "The approximate value of square root of x is 2.65"
- c. The approximate value of square root of 7 is 2.65
- d. The approximate value of square root of x is 2.65

5. Which one of the following is the correct outcome of the command

```
paste("The age of 3 students are", c("18 years", "19 years", "  
20 years"), collapse=", and ") ?
```

- a. [1] "The age of 3 students are 18 years, 19 years and 20 year  
s"
- b. [1] "The age of 3 students are 18 years, and The age of 3 stu  
dents are 19 years, and The age of 3 students are 20 years"
- c. [1] "The age of 3 students are 18 years, and 19 years, and 20  
years"
- d. [1] "The age of 3 students are 18, 19 and 20 years"

6. Which one of the following is the correct outcome of the command

```
paste("Students in class 1, 2 and 3 are", 50:52, sep=":") ?
```

a.

```
[1] "Students in class 1 are:50" "Students in class 2 are:51" "  
Students in class 3 are:52"
```

b.

```
[1] "Students in class 1, 2 and 3 are:50:52" "Students in class  
1, 2 and 3 are:50:52" "Students in class 1, 2 and 3 are:50:52"
```

c.

```
[1] "Students in class 1, 2 and 3 are:50 Students in class 1, 2  
and 3 are:51 Students in class 1, 2 and 3 are:52"
```

d.

```
[1] "Students in class 1, 2 and 3 are:50" "Students in class 1,  
2 and 3 are:51" "Students in class 1, 2 and 3 are:52"
```

7. Which one of the following is the correct outcome of the command

```
paste("Total number of pages in books are", 200:202, sep="+", collapse="") ?
```

a.

```
[1] "Total number of pages in books are+200Total number of pages in books are+201Total number of pages in books are+202"
```

b.

```
[1] "Total number of pages in books are+200+201+202"
```

c.

```
[1] "Total number of pages in books are 603"
```

d.

```
[1] "Total number of pages in books are 200" + "Total number of pages in books are 201" + "Total number of pages in books are 202"
```

8. Which one of the following is the correct outcome of `y` where

```
y=strsplit("Landlord of House Number 100::Landlord of House Num  
ber 200::Landlord of House Number 300","::") ?
```

a.

```
[[1]]
```

```
[1] " Landlord of House Number 100:200::300"
```

b.

```
[[1]]
```

```
[1]
```

```
Landlord of House Number 100, Landlord of House Number 200, Lan  
dlord of House Number 300,
```

c.

```
[[1]]
```

```
[1] "Landlord of House Number 100" "Landlord of House Number 20  
0" "Landlord of House Number 300"
```

d.

```
[[1]]
```

```
[1] "Landlord of House Number 100":"Landlord of House Number 2  
00":"Landlord of House Number 300"
```

9. Let

```
y=strsplit("Landlord of House Number 100::Landlord of House Num  
ber 200::Landlord of House Number 300","::")
```

Which one of the following is the correct command to get the outcome of `y` as "Landlord of House Number 300" ?

a. `y[1][3]`

b. `y[3][1]`

c. `y[[1]][3]`

d. `[[3]][1]`

10. Which one of the following is the correct outcome of the command

```
ncchar("NPTEL - National Programme on Technology Enhanced Learning- is a joint initiative of the IITs and IISc.") ?
```

- a. [1] 99
- b. [1] 100
- c. [1] 101
- d. [1] 103

11. Which one of the following is the correct outcome of the command

```
nzchar("NPTEL - National Programme on Technology Enhanced Learning- is a joint initiative of the IITs and IISc.") ?
```

- a. TRUE
- b. FALSE
- c. 103
- d. 101

12. Which one of the following is the correct outcome of `nzchar(x)` where

```
x=c("NPTEL - National Programme on Technology Enhanced Learning-", "is a joint initiative of the", "IITs and IISc.")?
```

- a. TRUE
- b. FALSE
- c. FALSE FALSE FALSE
- d. TRUE TRUE TRUE

13. Which one of the following is the correct outcome of the command `tolower("InDIan InStItUtE Of TEchNOloGY")` ?

- a. [1] "Indian Institute Of Technology")
- b. [1] "indian institute of technology"
- c. [1] "iNdiAN iNsTiTuTe oF teCHnOLOGY")
- d. [1] "TEchNOloGY Of InStItUtE InDIAn ")

14. Which one of the following is the correct outcome of the command `toupper("Institute Of National Importance in India")` ?

- a. [1] "institute of national importance in india"
- b. [1] "iNSTITUTE oF nATIONAL iMPORTANCE IN iNDIA"
- c. [1] "INSTITUTE OF NATIONAL IMPORTANCE IN INDIA"
- d. [1] "Institute Of National Importance in India"

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 9**

1. d
2. b
3. d
4. c
5. b
6. d
7. a
8. c
9. c
10. d
11. a
12. d
13. b
14. c

# MOOC Course - Foundations of R Software

July 2022

## Assignment 10

1. Which one of the following is the correct outcome of the command

```
sub("8", "12", "Foundations of R Software Course will be of 12  
weeks duration" ) ?
```

- a. [1] "Foundations of R Software Course will be of 8 weeks duration"
- b. [1] "Foundations of R Software Course will be of 12 weeks duration"
- c. [1] "Foundations of R Software Course will be of 8-12 weeks duration"
- d. [1] "Foundations of R Software Course will be of 8, 12 weeks duration"

2. Which one of the following is the correct outcome of the command

```
gsub("Students in mathematics course", "Students in statistics  
course", "mathematics - 250. statistics - 300") ?
```

- a. [1] "mathematics - 250. statistics - 300"
- b. [1] "Students in mathematics course - 250." " Students in  
mathematics course - 300"
- c. [1] "Students in statistics course - 250." " Students in  
statistics course - 300"
- d. [1] "Students in statistics course - 250." "mathematics  
Course - 300"

3. Which one of the following is the correct outcome of the command

```
grep("cdab", c("cdabc", "cdef", "cbcda", "adcdaba", "cdabcdab  
" )) ?
```

- a. [1] "cdabc" "cbcda" "adcdaba" "cdabcdab"
- b. [1] "cdef"
- c. [1] 1 3 4 5
- d. [1] 2

4. Which one of the following is the correct outcome of the command

```
grep("[m-r]", letters) ?
```

- a. [1] m-n-o-p-q-r
- b. [1] m n o p q r
- c. [1] 13-14-15-16-17-18
- d. [1] 13 14 15 16 17 18

5. Which one of the following is the correct outcome of the command

```
grepl("Students",x) where  
x=c("Students in mathematics course", "Students in statistics  
course", "mathematics - 250. statistics - 300") ?
```

- a. TRUE TRUE FALSE
- b. TRUE
- c. FALSE
- d. FALSE FALSE TRUE

6. Which one of the following are the respective correct commands to obtain the names of columns and rows in a data frame `mtcars` ?

- a. `colnames(mtcars)` and `rownames(mtcars)`
- b. `colname(mtcars)` and `rowname(mtcars)`
- c. `coln(mtcars)` and `rown(mtcars)`
- d. `cnames(mtcars)` and `rnames(mtcars)`

7. Which one of the following is the correct command to obtain the dimension, name, and type of each variable in a data frame `mtcars` ?

- a. `string(mtcars)`
- b. `strname(mtcars)`
- c. `str(mtcars)`
- d. `stringname(mtcars)`

8. Which one of the following is the correct command to extract the variable `gear` from a data frame `mtcars` ?

- a. `gear$mtcars`
- b. `name(mtcars$gear)`
- c. `name(gear$mtcars)`
- d. `mtcars$gear`

9. Which one of the following is the correct command to extract the first five data points contained in the variable `mpg` from a data frame `mtcars` ?

- a. `mpg$mtcars[1:5]`
- b. `mtcars$mpg[1:5]`
- c. `mtcars$mpg[1,5]`
- d. `[mtcars,suzuki](1,5)`

**Answer Questions 10-12 on the basis of the following information:**

Consider the data frame `painters` in the library MASS. Use command `library(MASS)` to load the library `MASS` and use the command `attach(painters)` to attach the database `painters`.

10. Which one of the following is respectively the correct command to draw information on those painters who have used the "Colour" coded as 6 and are from "School" F from the data frame `painters` and what is the corresponding outcome?

a.

`subset(painters, Colour==6 & School==F)`

and

	Composition	Drawing	Colour	Expression	School
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F

b.

`subset(painters, Colour=6 & School=F)`

and

	Composition	Drawing	Colour	Expression	School
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F

c.

`subset(painters, Colour=='6' & School=='F')`

and

	Composition	Drawing	Colour	Expression	School
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F

d.

`subset(painters, Colour=='6' and School=='F')`

and

	Composition	Drawing	Colour	Expression
Pourbus	4	15	6	6
Van Leyden	8	6	6	4

11. Which one of the following is respectively the correct command to draw the information on those painters who have used the "Composition" as 15 and "Expression" is more than 9 and what is the output from the data frame `painters`?

a.

```
subset(painters, Composition =='15' & Expression > 9)  
and
```

	Composition	Drawing	Colour	Expression	School
Perino del Vaga	15	16	7	6	A
Tintoretto	15	14	16	4	D
Veronese	15	10	16	3	D
Teniers	15	12	13	6	G

b.

```
subset(painters, Composition =='15' & Expression > 9)  
and
```

	Composition	Drawing	Colour	Expression	School
Da Vinci	15	16	4	14	A
Guilio Romano	15	16	4	14	A
Primiticcio	15	14	7	10	B
Vanius	15	15	12	13	C
Domenichino	15	17	9	17	E
The Carracci	15	17	13	13	E
Rembrandt	15	6	17	12	G
Van Dyck	15	10	17	13	G
Le Suer	15	15	4	15	H
Poussin	15	17	6	15	H

c.

```
subset(painters, Composition =='15' & Expression > '9')  
and
```

	Composition	Drawing	Colour	Expression	School
Da Vinci	15	16	4	14	A
Guilio Romano	15	16	4	14	A
Primiticcio	15	14	7	10	B
Vanius	15	15	12	13	C
Domenichino	15	17	9	17	E
The Carracci	15	17	13	13	E
Rembrandt	15	6	17	12	G
Van Dyck	15	10	17	13	G
Le Suer	15	15	4	15	H
Poussin	15	17	6	15	H

d.

`subset(painters, Composition =15 & Expression > 9)`

and

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
Michelangelo	8	17	4	8	A
Perino del Vaga	15	16	7	6	A
Perugino	4	12	10	4	A
Raphael	17	18	12	18	A
F. Zuccaro	10	13	8	8	B
Fr. Salviata	13	15	8	8	B
Parmigiano	10	15	6	6	B
Primiticcio	15	14	7	10	B
T. Zuccaro	13	14	10	9	B
Volterra	12	15	5	8	B
Barocci	14	15	6	10	C
Cortona	16	14	12	6	C
Josepin	10	10	6	2	C
L. Jordaens	13	12	9	6	C
Testa	11	15	0	6	C
Vanius	15	15	12	13	C
Bassano	6	8	17	0	D
Bellini	4	6	14	0	D
Giorgione	8	9	18	4	D
Murillo	6	8	15	4	D
Palma Giovane	12	9	14	6	D
Palma Vecchio	5	6	16	0	D
Pordenone	8	14	17	5	D
Tintoretto	15	14	16	4	D
Titian	12	15	18	6	D
Veronese	15	10	16	3	D
Albani	14	14	10	6	E
Caravaggio	6	6	16	0	E
Correggio	13	13	15	12	E
Domenichino	15	17	9	17	E
Guercino	18	10	10	4	E
Lanfranco	14	13	10	5	E
The Carracci	15	17	13	13	E
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F
Diepenbeck	11	10	14	6	G
J. Jordaens	10	8	16	6	G
Otho Venius	13	14	10	10	G
Rembrandt	15	6	17	12	G

Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H
Le Sueur	15	15	4	15	H
Poussin	15	17	6	15	H

12. Which one of the following are respectively the correct commands to draw the information and output on those painters who have used the "Colour" coded as 4 , "School" as A when information on the variables "Drawing" and "Expression" is removed from the data frame `painters` and its corresponding outcome?

a.

```
subset(painters, School=A & Colour=, select=c(-2,-4))
```

and

	Composition	Colour	School
Da Vinci	15	4	A
Guilio Romano	15	4	A
Michelangelo	8	4	A

b.

```
subset(painters, School=="A" & Colour=="4", select=c(-2,-4))
```

and

	Composition	Drawing	Colour	Expression	School
Da Vinci	15	16	4	14	A
Guilio Romano	15	16	4	14	A
Michelangelo	8	17	4	8	A

c.

```
subset(painters, School=A & Colour=4, select=c(-2,-4))
```

and

	Composition	Drawing	Colour	Expression	School
Da Vinci	15	16	4	14	A
Guilio Romano	15	16	4	14	A
Michelangelo	8	17	4	8	A

d.

```
subset(painters, School=="A" & Colour=="4", select=c(-2,-4))
```

and

	Composition	Colour	School
Da Vinci	15	4	A
Guilio Romano	15	4	A
Michelangelo	8	4	A

13. Suppose `x = c("Germany", "India", "USA")` and `y = c(10, 20, 30)` then which of the following is the correct outcome of the command `rbind(x,y)` when executed in R console?

A.

```
[,1]      [,2]      [,3]  
x "Germany" "India"  "USA"  
y "10"       "20"    "30"
```

B.

```
x          y  
[1,] "Germany" "10"  
[2,] "India"     "20"  
[3,] "USA"       "30"
```

C.

```
[,1]      [,2]      [,3]  
y "10"       "20"    "30"  
x "Germany"  "India"  "USA"
```

D.

```
y      x  
[1,] "10"  "Germany"  
[2,] "20"  "India"  
[3,] "30"  "USA"
```

14. Suppose `x = c("Germany", "India", "USA")` and `y = c(10, 20, 30)` then which of the following is the correct outcome of the command `cbind(x,y)` when executed in R console?

A.

```
[,1]      [,2]      [,3]  
x "Germany" "India"  "USA"  
y "10"       "20"    "30"
```

B.

```
[,1]      [,2]      [,3]  
y "10"       "20"    "30"  
x "Germany" "India"  "USA"
```

C.

	x	y
[1,]	"Germany"	"10"
[2,]	"India"	"20"
[3,]	"USA"	"30"

D.

	y	x
[1,]	"10"	"Germany"
[2,]	"20"	"India"
[3,]	"30"	"USA"

## **MOOC Course - Foundations of R Software**

### **Answers of Assignment 10**

1. b

2. a

3. c

4. d

5. a

6. a

7. c

8. d

9. b

10. c

11. b

12. d

13. a

14. c

# MOOC Course - Foundations of R Software

July 2022

## Assignment 11

1. A comma separated value data file named as `course.csv` having `header` can be correctly read in R by which of the following command?

- a. `readcsv("course.csv", header= TRUE)`
- b. `read.csv("course.csv", header=FALSE)`
- c. `csvread(course.csv, header=FALSE)`
- d. `read.csv("course.csv", header= TRUE)`

2. A tabular data file named as `course1.txt` where the values are separated by blank space and having `header` can be correctly read in R by which of the following command?

- a. `read.table("course1.txt", header= TRUE, sep = " ")`
- b. `read.txt("course1.txt", header=FALSE, sep = " ")`
- c. `tabularread(course1.txt, header=FALSE, sep = " ")`
- d. `read.table(course1.txt, header= TRUE, sep = )`

3. A spreadsheet created in MS-Excel software is named as `marks.xlsx`. The sheet number 2 of this file can be correctly read in R by which of the following command?

- a. `read_excel("marks.xlsx", sheet=2)`
- b. `read.excel("marks.xlsx", sheetIndex=2)`
- c. `Read_excel("marks.xlsx", sheet=2)`
- d. `read.excel(marks.xlsx, sheetIndex=2)`

4. A spreadsheet created in MS-Excel software is named as `marks.xlsx`. One of the sheets in this file whose name is `Physics` can be read in R by which of the following command?

- a. `read.excel("marks.xlsx", sheetName="Physics")`
- b. `read.excel("marks.xlsx", sheet="Physics")`
- c. `read_excel("marks.xlsx", sheet="Physics")`
- d. `read_excel("marks.xlsx", name="Physics")`

5. Suppose two dice are rolled 10 times and following is the sum of the numbers on the upper faces of the dice: 6,3,12,12,7,7,7,7,6,3,2. Which one of the following is the correct command to obtain the absolute frequencies of this data in R?

- a. `table(c(6,3,12,12,7,7,7,7,6,3,2))`
- b. `table(6,3,12,12,7,7,7,7,6,3,2)`
- c. `table(c(6,3,12,12,7,7,7,7,6,3,2))/length(c(6,3,12,12,7,7,7,7,6,3,2))`
- d. `table(6,3,12,12,7,7,7,6,3,2)/length(6,3,12,12,7,7,7,7,6,3,2)`

6. Suppose two dice are rolled 20 times and the following are the sum of numbers on the upper face: 8, 8, 7, 12, 2, 3, 5, 3, 4, 6, 7, 6, 5, 6, 6, 7, 4, 3, 2, 9, 9. Which one of the following is the correct outcome of the following command in R?

```
table(c(8,8,7,12,2,3,5,3,4,6,7,6,5,6,6,7,4, 3,2,9))
```

a.

2	3	4	5	6	7	8	9	12
0.10	0.15	0.10	0.10	0.20	0.15	0.10	0.10	0.05

b.

2	3	4	5	6	7	8	9	12
1	1	1	1	1	1	1	2	1

c.

2	3	4	5	6	7	8	9	12
2	3	2	2	4	3	2	1	1

d.

2	3	2	2	4	3	2	1	1
2	3	4	5	6	7	8	9	12

7. Which one of the following is the correct command to obtain the 2<sup>nd</sup>, 5<sup>th</sup>, 7<sup>th</sup> and 9<sup>th</sup> deciles of a data vector **x** in R?

- a. `decile(x, probs=c(20%,50%,70%,90%))`
- b. `decile(x, probs=(0.20,0.50,0.70,0.90))`
- c. `quantile(x, probs=(20%,50%,70%,90%))`
- d. `quantile(x, probs=c(0.20,0.50,0.70,0.90))`

8. Which one of the following correctly specify the quantiles of a data vector `marks` obtained as an outcome of the command `quantile(marks, probs = seq(0, 1, 0.15))` in R?

- a. 15<sup>th</sup>
- b. 0<sup>th</sup>,15<sup>th</sup>,30<sup>th</sup>,45<sup>th</sup>,60<sup>th</sup>,75<sup>th</sup>,90<sup>th</sup>
- c. 15<sup>th</sup>,30<sup>th</sup>,45<sup>th</sup>,60<sup>th</sup>,75<sup>th</sup>,90th
- d. 15<sup>th</sup>,45<sup>th</sup>,90<sup>th</sup>

9. Suppose two dice are rolled 10 times and the following sum of the numbers on the upper faces of the dice: 6,3,12,12,7,7,7,7,6,3,2. Which of the following command in R is used to obtain the bar plot of this data based on relative frequencies?

- a. `barplot(6,3,12,12,7,7,7,7,6,3,2)/length(6,3,12,12,7,7,7,7,6,3,2)`
- b. `barplot(table(6,3,12,12,7,7,7,7,6,3,2))/length(6,3,12,12,7,7,7,7,6,3,2))`
- c. `barplot(c(6,3,12,12,7,7,7,7,6,3,2))/length(c(6,3,12,12,7,7,7,7,6,3,2)))`
- d. `barplot(table(c(6,3,12,12,7,7,7,7,6,3,2))/length(c(6,3,12,12,7,7,7,7,6,3,2)))`

10. Suppose two dice are rolled 10 times and the following sum of the numbers on the upper faces of the dice: 6,3,12,12,7,7,7,7,6,3,2. Which of the following command in R is used to obtain the bar plot of this data based on absolute frequencies having the main title as “Sum of Numbers on Dice” and all the bars are filled in red colour?

- a. `barplot(table(c(6,3,12,12,7,7,7,7,6,3,2)), main="Sum of Numbers on Dice", col="red")`
- b. `barplot((6,3,12,12,7,7,7,7,6,3,2), title="Sum of Numbers on Dice", col="red")`
- c. `barplot(table(c(6,3,12,12,7,7,7,7,6,3,2)), main=Sum of Numbers on Dice, col=red)`
- d. `barplot(table(c(6,3,12,12,7,7,7,7,6,3,2)), title=Sum of Numbers on Dice, col=red)`

**MOOC Course - Foundations of R Software**

**Answers of Assignment 11**

1. d
2. a
3. a
4. c
5. a
6. c
7. d
8. b
9. d
10. a

# Week 12: Assignment 12

The due date for submitting this assignment has passed.

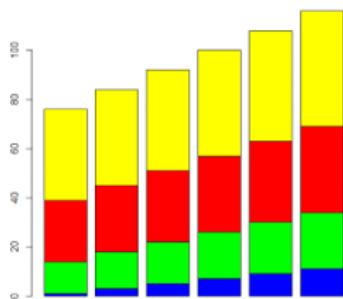
Due on 2022-10-19, 23:59

Assignment submitted on 2022-10-19, 21:30 IST

1 point

Question 1-4 are based on the following information:

The R command `barplot(*1*(nrow=*2*, ncol=*3*, data=seq(1,48,2), byrow=T), col =*4*)` is used to produce the following plot for a given data vector `x` with `*n*` indicating that some values are being used.



Q.1:

The value (or text) at `*1*` is

- character
- matrix
- num
- c

Yes, the answer is correct.

Score: 1

Accepted Answers:

`matrix`

1 point

The value at `*2*` is

- 6

24

16

4

Yes, the answer is correct.

Score: 1

Accepted Answers:

4

*1 point*

The value at \*3\* is

6

24

16

4

Yes, the answer is correct.

Score: 1

Accepted Answers:

6

*1 point*

The value (or text) at \*4\* is

c(blue, green, red, yellow)

c("yellow", "red", "green", "blue")

c("blue", "green", "red", "yellow")

c(yellow, red, green, blue)

Yes, the answer is correct.

Score: 1

Accepted Answers:

c("blue", "green", "red", "yellow")

*1 point*

Suppose two dice are rolled 20 times and the following are the sum of numbers on the upper face: 8, 8, 7, 12, 2, 3, 5, 3, 4, 6, 7, 6, 5, 6, 6, 7, 4, 3, 2, 9, 9. Which one of the following command is used to obtain the histogram of this data in R based on absolute frequencies?

```
histo(c(8,8,7,12,2,3,5,3,4,6,7,6,5,6,6,7,4, 3,2,9))
```

```
hist(c(8,8,7,12,2,3,5,3,4,6,7,6,5,6,6,7,4, 3,2,9))
```

```
hist(table(8,8,7,12,2,3,5,3,4,6,7,6,5,6,6,7,4, 3,2,9))
```

```
histogram(c(8,8,7,12,2,3,5,3,4,6,7,6,5,6,6,7,4, 3,2,9))
```

Yes, the answer is correct.

Score: 1

Accepted Answers:

```
hist(c(8,8,7,12,2,3,5,3,4,6,7,6,5,6,6,7,4, 3,2,9))
```

*1 point*

Suppose the number of persons visiting a shop for 20 consecutive days is recorded as follows: 520, 364, 251, 415, 763, 439, 854, 476, 634, 548, 754, 684, 577, 894, 447, 762, 868, 875, 582, 845. Which one of the following is the correct command to obtain the histogram of this data in R based on relative frequencies?

```
hist(c(520,364,251,415,763,439,854,476,634,548,754,684,577,894,  
447,762,868, 875,582,845) , freq=FALSE)
```

```
hist(c(520,364,251,415,763,439,854,476,634,548,754,684,577,894,  
447,762,868, 875,582,845) , freq=TRUE)
```

```
hist((520,364,251,415,763,439,854,476,634,548,754,684,577,894,  
447,762,868, 875,582,845) , freq=FALSE)
```

```
hist(table((520,364,251,415,763,439,854,476,634,548,754,684,577  
,894,447,762,868, 875,582,845)) ,freq=TRUE)
```

Yes, the answer is correct.

Score: 1

Accepted Answers:

```
hist(c(520,364,251,415,763,439,854,476,634,548,754,684,577,894,  
447,762,868, 875,582,845), freq=FALSE)
```

**1 point**

Suppose the division of students in the class – first (denoted as 1), second (denoted as 2) and third (denoted as 3) are recorded in the following data vector

```
division = c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2,1,3,  
3,3,2,2,2,2,1,2,2,1,1,1,3,2,2,1,2,3,2,2,1,2,3,3,2,1,2,2,3,1,1,2  
,1,2,3,2,3)
```

Which of the following is the correct command to obtain the pie diagram having the title as "Division of Students"?

```
pie(table(division), main=Division of Students)
```

```
pie(division, main="Division of Students")
```

```
pie(table(division), title="Division of Students")
```

```
pie(table(division), main="Division of Students")
```

Yes, the answer is correct.

Score: 1

Accepted Answers:

```
pie(table(division), main="Division of Students")
```

**1 point**

Which of the following is the correct matching of the commands in column A and the outputs in Column B about the nature of variables for the plots?

Column A		Column B	
(1)	Histogram	(i)	Quantitative variable
(2)	Barplot	(ii)	Qualitative variable
(3)	Pie diagram		

(1)-(i), (2)-(ii), (3)-(ii)

(1)-(ii), (2)-(i), (3)-(ii)

(1)-(i), (2)-(i) , (3)-(i)

(1)-(ii), (2)-(ii) , (3)-(ii)

Yes, the answer is correct.

Score: 1

Accepted Answers:

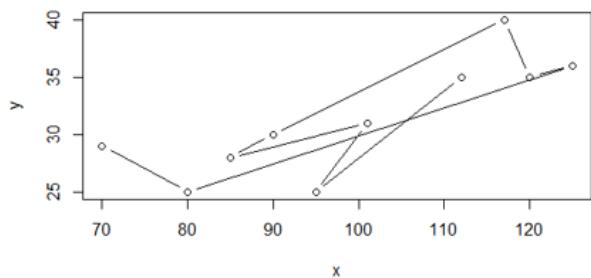
(1)-(i), (2)-(ii), (3)-(ii)

**1 point**

The paired data on height (**x**) in centimeters and weight (**y**) in kilograms of 10 children is obtained. Which one of the following is the correct command to obtain the scatter plot of the following type in R for the given data

**x = c(112,95,101,85,90,117,120,125,80,70)**

**y = c(35,25,31,28,30,40,35,36,25,29) ?**



- plot(x,y, "l")**
- plot(x,y, "s")**
- plot(x,y, "h")**
- plot(x,y, "b")**

Yes, the answer is correct.

Score: 1

Accepted Answers:

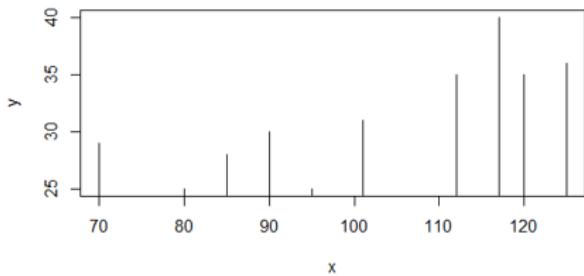
**plot(x,y, "b")**

**1 point**

The paired data on height (**x**) in centimeters and weight (**y**) in kilograms of 10 children is obtained. Which one of the following is the correct command to obtain the scatter plot of the following type in R for the given data

**x = c(112,95,101,85,90,117,120,125,80,70)**

**y = c(35,25,31,28,30,40,35,36,25,29) ?**



- plot(x,y, "l")**
- plot(x,y, "h")**
- plot(x,y, "s")**
- plot(x,y, "b")**

Yes, the answer is correct.

Score: 1

Accepted Answers:

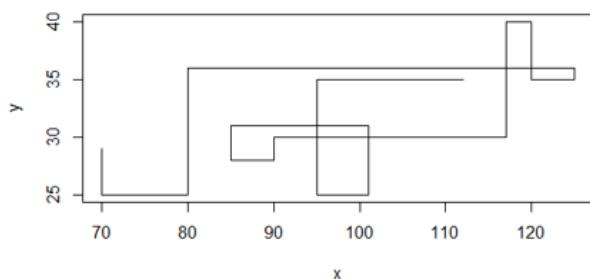
**plot(x,y, "h")**

**1 point**

The paired data on height (**x**) in centimeters and weight (**y**) in kilograms of 10 children is obtained. Which one of the following is the correct command to obtain the scatter plot of the following type in R for the given data

**x = c(112,95,101,85,90,117,120,125,80,70)**

**y = c(35,25,31,28,30,40,35,36,25,29) ?**



- `plot(x,y, "h")`
- `plot(x,y, "l")`
- `plot(x,y, "s")`
- `plot(x,y, "b")`

Yes, the answer is correct.

Score: 1

Accepted Answers:

```
plot(x,y, "s")
```

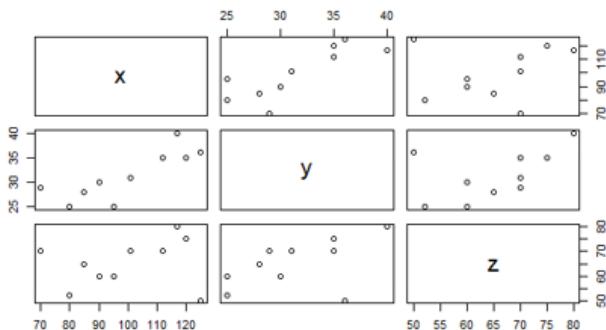
**1 point**

Which one of the following is the correct command to obtain the scatter plot of the following type in R for the given data

```
x = c(112,95,101,85,90,117,120,125,80,70)
```

```
y = c(35,25,31,28,30,40,35,36,25,29)
```

and `z = c(70,60,70,65,60,80,75,50,52)` ?



- `plots(cbind(x,y,z))`
- `plot(c(x,y,z))`
- `pairs(c(x,y,z))`
- `pairs(cbind(x,y,z))`

Yes, the answer is correct.

Score: 1

Accepted Answers:

```
pairs(cbind(x,y,z))
```

**1 point**

Which one of the following is the correct function to obtain the cube root of the sum of squares of three numbers?

```
○
sqrtcube <- function(x, y, z) {
  f1 <- x^2 + y^2 + z^2
  f2 <- (f1)^(1/3)
  print(paste("The value of sum of squares of", x, y, " and",
z, "is", f1, "and its cube root is", f2))
}
```

```
○
sqrtcube <- function(x, y, z) [
  f1 <- x^2 + y^2 + z^2
  f2 <- (f1)^(1/3)
  print(paste("The value of sum of squares of", x, y, " and",
z, "is", f1, "and its cube root is", f2))
]
```

```
○
sqrtcube <- function(x, y, z) {
  f1 <- x^2 + y^2 + z^2
  f2 <- (f1)^(1/3)
  print(paste("The value of sum of squares of", x, y, " and",
z, "is", f1, "and its cube root is", f2))
}
```

```
○
sqrtcube <- function(x, y, z) {
  f1 <- x^2 + y^2 + z^2
  (f2)^3 <- f1
  print(paste("The value of sum of squares of", x, y, " and",
z, "is", f1, "and its cube root is", f2))
}
```

Yes, the answer is correct.

Score: 1

Accepted Answers:

```

sqrtcube <- function(x, y, z) {
  f1 <- x^2 + y^2 + z^2
  f2 <- (f1)^(1/3)
  print(paste("The value of sum of squares of", x, y, " and",
z, "is", f1, "and its cube root is", f2))
}

```

**1 point**

Which one of the following is the correct code of the function to obtain the value of the following function on the basis of given data values of

$x = (x_1, x_2, \dots, x_n)$  and  $y = (y_1, y_2, \dots, y_n)$ :

$$f(x, y) = \frac{\log \sum_{i=1}^n \left( \frac{x_i}{y_i} \right)^{3/4}}{\exp \left( \sqrt[3]{\sum_{i=1}^n \left( \frac{x_i}{y_i} \right)^{3/4}} \right) + \left( \left( \sum_{i=1}^n \left( \frac{x_i}{y_i} \right)^{3/4} \right)^{1/3} \right)}$$



```

x=c(x1,x2,...,xn)
y=c(y1,y2,...,yn)

g <- function(x,y)
{
  n <- length(x)
  z <- 0
  for (i in 1:n)
  {
    z[i] <- x[i]/y[i]
  }
  sum(z)
}

f<-function()
{
fxy <- log(g(x,y) ^ (3/4))/exp(g(x,y) ^ (3/4)) + abs(g(x,y) ^ (3/4))
fxy
}

```

```
x=c(x1,x2,...,xn)
y=c(y1,y2,...,yn)

g <- function(x,y)
{
  n <- length(x)
  z <- 0
  for (i in 1:n)
  {
    z[i] <- (x[i]/y[i])^(3/4)
  }
  sum(z)
}

f<-function()
{
  fxy <- log(g(x,y))/exp(g(x,y)^(1/3)) + abs(g(x,y)^(1/5))
  fxy
}
```

```
x=c(x1,x2,...,xn)
y=c(y1,y2,...,yn)

g <- function(x,y)
{
  n <- length(x)
  z <- 0
  for (i in 1:n)
  {
    z[i] <- (x[i]^3/y[i]^4)
  }
  sum(z)
}

f<-function()
{
  fxy <- log(g(x,y))/exp(g(x,y)) + abs(g(x,y))
  fxy
}
```

```
x=c(x1,x2,...,xn)
y=c(y1,y2,...,yn)

g <- function(x,y)
{
  n <- length(x)
  z <- 0
  for (i in 1:n)
  {
    z[i] <- x[i]/y[i]
  }
  sum(z)
}

f<-function()
{
  fxy <- (log(g(x,y))^(3/4) / (exp(g(x,y))^(3/4) + 
  (abs(g(x,y)))^(3/4))
  fxy
}
```

Yes, the answer is correct.

Score: 1

Accepted Answers:

```
x=c(x1,x2,...,xn)
y=c(y1,y2,...,yn)

g <- function(x,y)
{
  n <- length(x)
  z <- 0
  for (i in 1:n)
  {
    z[i] <- (x[i]/y[i])^(3/4)
  }
  sum(z)
}

f<-function()
{
  fxy <- log(g(x,y))/exp(g(x,y)^(1/3)) + abs(g(x,y)^(1/5))
  fxy
}
```

# **Foundations of R Software**

**Lecture 0**

## **How to Learn and Follow the Course**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Basics

Blue Colour Courier New Font means a Command or Syntax in R.

Black colour Calibri font means usual expression.

Example:

Statement: The assignment operators are the left arrow with dash `<-` and equal sign `=`

`x <- 20` assigns the value 20 to `x` .

`x = 20` assigns the value 20 to `x` .

# Basics

**Outcome over R console:**

```
> x = 20
```

```
> x
```

```
[1] 20
```

# Basics

Syntax:

`y = x * 2` assigns the value `2*x` to `y`.

`z = x + y` assigns the value `x + y` to `z`.

Outcome over R console:

```
> y = x * 2
```

```
> y
```

```
[1] 40
```

```
> z = x + y
```

```
> z
```

```
[1] 60
```

# Basics

## Screenshot of outcome over R console



```
> x=20
> x
[1] 20
>
> y = x * 2
> y
[1] 40
> z = x + y
> z
[1] 60
```

# **Foundations of R Software**

## **Lecture 1 Introduction**

::::

## **Why R and Installation Procedure**

Shalabh

Department of Mathematics and Statistics

Indian Institute of Technology Kanpur

# What is R?

R is an environment for data manipulation, statistical computing, graphics display and data analysis.

Effective data handling and storage of outputs is possible.

Simple as well as complicated calculations are possible.

# **What is R?**

**Graphical display on-screen and hardcopy are possible.**

**Programming language is effective which includes all possibilities just like any other good programming language.**

**The R language is very similar in appearance to "S" - language based on a software S-Plus.**

# The R Foundation

The R Foundation is a **non-profit organization** working in the public interest.

It has been founded by the members of the **R Development Core Team** in order to provide support for the R project and other innovations in statistical computing.

Hold and administer the copyright of R software and documentation.

# The R Development

Professors Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, began an alternative implementation of the basic S language, completely independent of S-PLUS in 1991.

It was named partly after the first names of the first two R authors and partly as a play on the name of S.



Ross Ihaka and Robert Gentleman

( Source: <https://snipcademy.com/r-introduction>)

# The R Development

The first official release of R came in 1995.

The Comprehensive R Archive Network (CRAN) was officially announced 23 April 1997 with 3 mirrors and 12 contributed packages.

The first official "stable beta" version (v1.0) was released 29 February 2000.

As of November 2020, more than 16,000 packages are available.

# **Switching to R**

**Many people, research and design offices, analytical firms have started using R.**

**R is free.**

**Many statistical packages are freely available through the CRAN family of Internet sites covering a very wide range of modern statistics.**

**So you may also consider switching to R.**

# **Switching to R**

- **R has a statistical computing environment.**
- **It is free (open source) software and therefore is not a black box.**
- **It has a computer language which is convenient to use for statistical and graphical applications.**
- **The commands can be saved, run and stored in script files.**

## **Switching to R**

- **R is available for Windows, Unix, Linux and Macintosh platforms.**
- **R was developed to compete with S-Plus.**
- **Built in and contributed packages are available, and users are provided tools to make packages. It is possible to contribute own packages.**

## **Switching to R**

- This language provides the logical control of branching and looping, and modular programming using functions.
- Error messages are provided while executing a programme or a function which are helpful in finding the errors.

## **Switching to R**

- **R is an interpreted computer language (Not compiler).**
- **When we use the command line interface, each command or expression to be evaluated is typed at the command prompt, and immediately evaluated when the Enter key is pressed at the end of a syntactically complete statement.**

# Switching to R

- **Tips**
  - ✓ Press the up-arrow key to recall commands and edit them.
  - ✓ Use the Esc (Escape) key to cancel a command.
- **Graphics can be directly saved in a Postscript or PDF format.**

# Installing R

You may install R in a windows or Apple computer by downloading from <https://www.r-project.org>

Click on [download R](#)



## The R Project for Statistical Computing

### Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

### News

- [R version 4.1.2 \(Bird Hippie\)](#) has been released on 2021-11-01.
- [R version 4.0.5 \(Shake and Throw\)](#) was released on 2021-03-31.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- You can support the R Foundation with a renewable subscription as a [supporting member](#)

# Installing R

## Choose any mirror and click on the link

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here:

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud

<https://cloud.r-project.org/>

Algeria

<https://cran.usthb.dz/>

Argentina

<http://mirror.fcaglp.unlp.edu.ar/CRAN/>

Australia

<https://cran.csiro.au/>

<https://mirror.aarnet.edu.au/pub/CRAN/>

<https://cran.ms.unimelb.edu.au/>

<https://cran.curtin.edu.au/>

Austria

<https://cran.wu.ac.at/>

Belgium

<https://www.freestatistics.org/cran/>

<https://ftp.belnet.be/mirror/CRAN/>

Brazil

<https://nbcgib.uesc.br/mirrors/cran/>

<https://cran-r.c3sl.ufpr.br/>

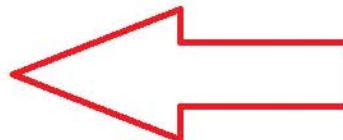
<https://cran.fiocruz.br/>

<https://vps.fmvz.usp.br/CRAN/>

<https://brieger.esalq.usp.br/CRAN/>

Bulgaria

<https://ftp.uni-sofia.bg/CRAN/>



Automatic redirection to servers worldwide, currently sponsored by Rstudio

University of Science and Technology Houari Boumediene

Universidad Nacional de La Plata

CSIRO

AARNET

School of Mathematics and Statistics, University of Melbourne

Curtin University

Wirtschaftsuniversität Wien

Patrick Wessa

Belnet, the Belgian research and education network

Computational Biology Center at Universidade Estadual de Santa Cruz

Universidade Federal do Parana

Oswaldo Cruz Foundation, Rio de Janeiro

University of Sao Paulo, Sao Paulo

University of Sao Paulo, Piracicaba

Sofia University

# Installing R

Else, download it from the Comprehensive R Archive Network (CRAN) website: <http://cran.r-project.org/>

The screenshot shows a web browser window with the following details:

- Address Bar:** https://cran.csiro.au (highlighted with a red oval).
- Toolbar:** Includes back, forward, search, and other standard browser icons.
- Header:** The Comprehensive R Archive Network
- Left Sidebar (CRAN Links):**
  - CRAN Mirrors
  - What's new?
  - Task Views
  - Search
  - About R
  - R Homepage
  - The R Journal
  - Software
  - R Sources
  - R Binaries
  - Packages
  - Other
  - Documentation
  - Manuals
  - FAQs
  - Contributed
- Main Content Area:**
  - Download and Install R:** Precompiled binary distributions of the base system and contributed packages. It lists options for Windows and Mac users:
    - Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
    - Download R for macOS
    - Download R for Windows
  - Source Code for all Platforms:** Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!
    - The latest release (2021-11-01, Bird Hippie) [R-4.1.2.tar.gz](#), read [what's new](#) in the latest version.
    - Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
    - Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
    - Source code of older versions of R is [available here](#).
    - Contributed extension [packages](#)

# **Foundations of R Software**

## **Lecture 2 Introduction**

::::

## **Help, Demonstration, and Examples in R**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Starting with R

To start R, double click on the icon .



Then we get the following Gui (Graphic user interface) window screen

A screenshot of the RGui (64-bit) window. The title bar says 'RGui (64-bit)'. The menu bar includes File, Edit, View, Misc, Packages, Windows, and Help. Below the menu is a toolbar with various icons. The main area is titled 'R Console'. It displays the R startup message, which includes information about the version (4.1.2), copyright (2021), platform (x86\_64-w64-mingw32/x64), and license details. It also mentions natural language support and collaborative project contributors. At the bottom, it says '[Previously saved workspace restored]' and has a red cursor at '> |'.

```
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 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.

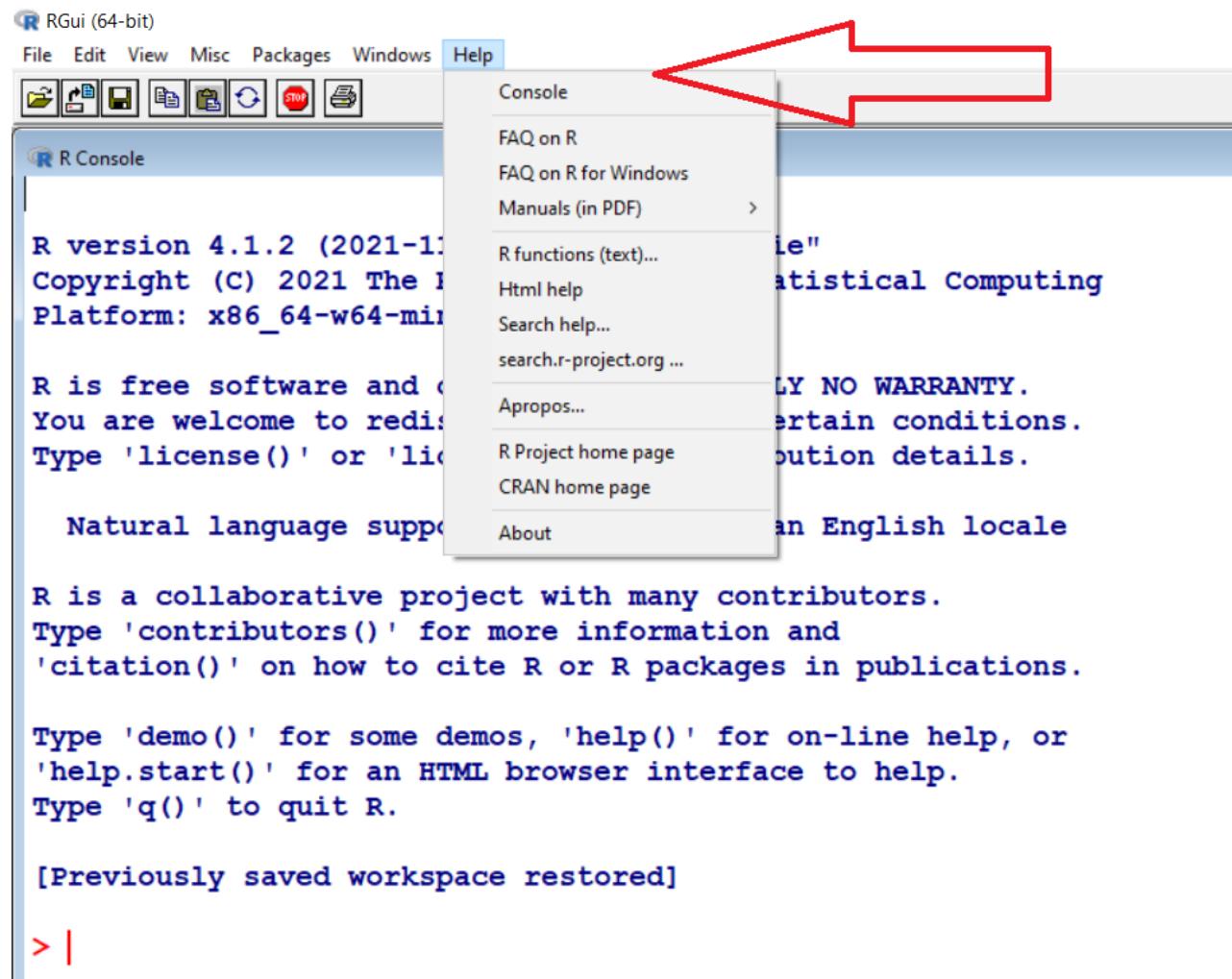
[Previously saved workspace restored]

> |
```

# Getting Help in R

This can be done in one of the following ways:

- 1) Start R software and click the help button in the toolbar of the R Gui (Graphic user interface) window.



# Getting Help in R

2. Search for help in Google [www.google.com](http://www.google.com)
3. If you need help with a function, then type question mark followed by the name of the function. For example, `?read.table` to get help for function `read.table`.

```
R Console

R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 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.

[Previously saved workspace restored]

> ?read.table
starting httpd help server ... done
> |
```

## read.table {utils}

### Data Input

#### Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

#### Usage

```
read.table(file, header = FALSE, sep = "", quote = "\'",  
          dec = ".", row.names, col.names,  
          as.is = !stringsAsFactors,  
          na.strings = "NA", colClasses = NA, nrows = -1,  
          skip = 0, check.names = TRUE, fill = !blank.lines.skip,  
          strip.white = FALSE, blank.lines.skip = TRUE,  
          comment.char = "#",  
          allowEscapes = FALSE, flush = FALSE,  
          stringsAsFactors = default.stringsAsFactors(),  
          fileEncoding = "", encoding = "unknown", text)  
  
read.csv(file, header = TRUE, sep = ",", quote = "\'",  
        dec = ".", fill = TRUE, comment.char = "", ...)  
  
read.csv2(file, header = TRUE, sep = ";", quote = "\'",  
          dec = ",", fill = TRUE, comment.char = "", ...)  
  
read.delim(file, header = TRUE, sep = "\t", quote = "\'",  
          dec = ".", fill = TRUE, comment.char = "", ...)  
  
read.delim2(file, header = TRUE, sep = "\t", quote = "\'",  
            dec = ",", fill = TRUE, comment.char = "", ...)
```

#### Arguments

⋮

# ...continued

## Arguments

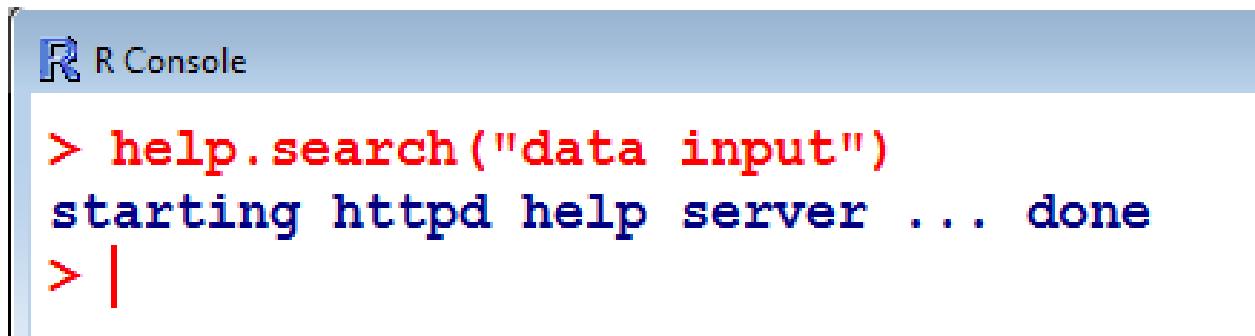
file	the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an <i>absolute</i> path, the file name is <i>relative</i> to the current working directory, <a href="#">getwd()</a> . Tilde-expansion is performed where supported. This can be a compressed file (see <a href="#">file</a> ).  Alternatively, <code>file</code> can be a readable text-mode <a href="#">connection</a> (which will be opened for reading if necessary, and if so <a href="#">closed</a> (and hence destroyed) at the end of the function call). (If <a href="#">stdin()</a> is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, <code>Ctrl-D</code> on Unix and <code>Ctrl-Z</code> on Windows. Any pushback on <code>stdin()</code> will be cleared before return.)
header	<code>file</code> can also be a complete URL. (For the supported URL schemes, see the ‘URLs’ section of the help for <a href="#">url</a> .)
sep	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: <code>header</code> is set to <code>TRUE</code> if and only if the first row contains one fewer field than the number of columns.
quote	the field separator character. Values on each line of the file are separated by this character. If <code>sep = ""</code> (the default for <code>read.table</code> ) the separator is ‘white space’, that is one or more spaces, tabs, newlines or carriage returns.
dec	the set of quoting characters. To disable quoting altogether, use <code>quote = ""</code> . See <a href="#">scan</a> for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless <code>colClasses</code> is specified.
row.names	the character used in the file for decimal points.  a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names.  If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if <code>row.names</code> is missing, the rows are numbered.  Using <code>row.names = NULL</code> forces row numbering. Missing or <code>NULL</code> <code>row.names</code> generate row names that are considered to be ‘automatic’ (and not preserved by <a href="#">as.matrix</a> ). a vector of optional names for the variables. The default is to use “v” followed by the column number. the default behavior of <code>read.table</code> is to convert character variables (which are not converted to logical, numeric or complex) to factors. The variable <code>as.is</code> controls the conversion of columns not otherwise specified by <code>colClasses</code> . Its value is either a vector of logicals (values are recycled if necessary), or a vector of numeric or character indices which specify which columns should not be converted to factors.
col.names	
as.is	

Note: to suppress all conversions including those of numeric columns, set `colClasses = "character"`.

**All minor details and explanations of all arguments are given.**

## Getting Help in R

4. Sometimes, you want to search by the subject on which we want help (e.g. data input). In such a case, type `help.search("data input")`



R Console

```
> help.search("data input")
starting httpd help server ... done
> |
```

Then we get....

# Then we get....

The screenshot shows a web browser window with the URL 127.0.0.1:24981/doc/html/Search?pattern=data input. The page title is "Search Results" and features a large R logo. The search string was "data input". Under "Help pages:", there are two links: [utils::read.DIF](#) (Data Input from Spreadsheet) and [utils::read.table](#) (Data Input). A red arrow points upwards from the text "Clicking over the link give required information" to the link [utils::read.table](#).

← | 127.0.0.1:24981/doc/html/Search?pattern=data input

## Search Results

R

The search string was "**data input**"

---

Help pages:

<a href="#">utils::read.DIF</a>	Data Input from Spreadsheet
<a href="#">utils::read.table</a>	Data Input

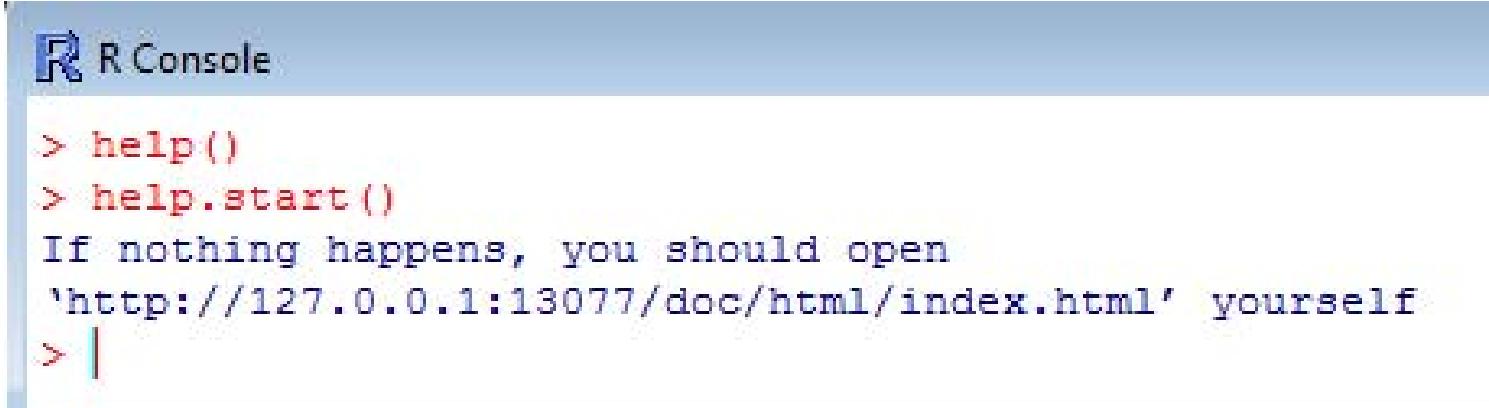
---

Clicking over the link give required information

# Getting Help in R

4. '`help()`' for on-line help,

or '`help.start()`' for an HTML browser interface to help.



R Console

```
> help()
> help.start()
If nothing happens, you should open
'<a href="http://127.0.0.1:13077/doc/html/index.html">http://127.0.0.1:13077/doc/html/index.html</a>' yourself
>
```

Then we get....



## Statistical Data Analysis



### Manuals

[An Introduction to R](#)  
[Writing R Extensions](#)  
[R Data Import/Export](#)

[The R Language Definition](#)  
[R Installation and Administration](#)  
[R Internals](#)

### Reference

[Packages](#)

[Search Engine & Keywords](#)

### Miscellaneous Material

[About R](#)  
[License](#)  
[NEWS](#)

[Authors](#)  
[Frequently Asked Questions](#)  
[User Manuals](#)

[Resources](#)  
[Thanks](#)  
[Technical papers](#)

### Material specific to the Windows port

[CHANGES up to R 2.15.0](#)

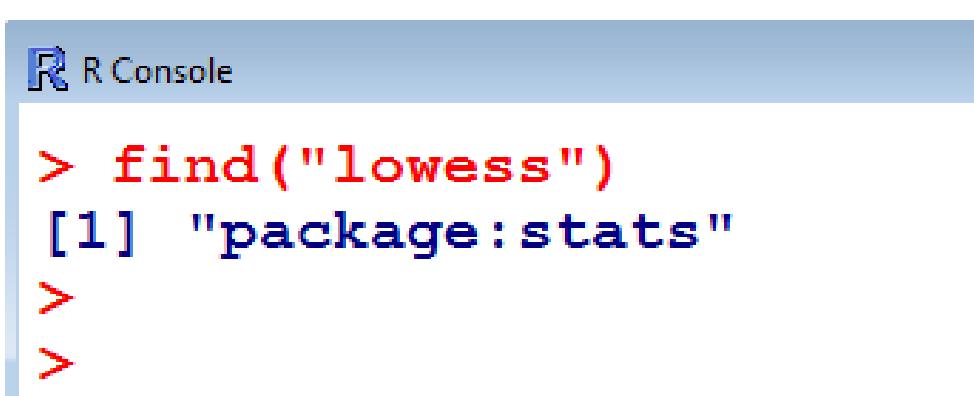
[Windows FAQ](#)

# Getting Help in R

- 5) Other useful functions are `find` and `apropos`.
- 6) The `find` function tells us what package something is in.

For example

```
> find("lowess") returns  
[1] "package:stats"
```



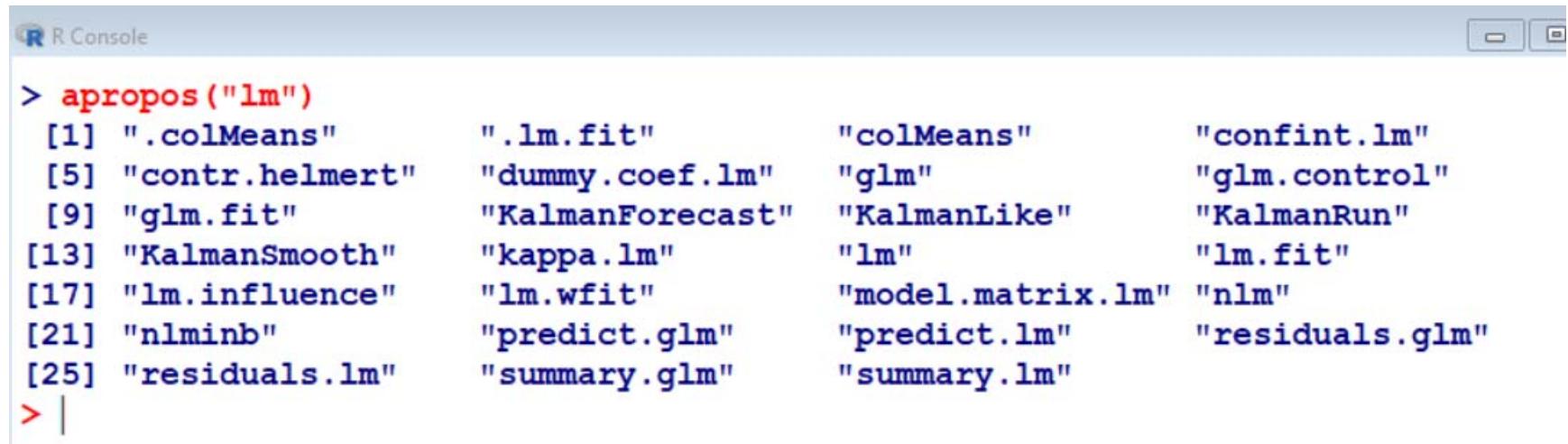
The image shows a screenshot of an R console window. The title bar says "R Console". The main area contains the following text:

```
> find("lowess")  
[1] "package:stats"  
>  
>
```

# Getting Help in R

7) The apropos returns a character vector giving the names of all objects in the search list that match your enquiry.

`apropos("lm")` returns



The screenshot shows an R console window titled "R Console". The command `> apropos("lm")` is entered, and the output is a character vector listing various R functions and objects related to linear models. The output is as follows:

```
> apropos("lm")
[1] ".colMeans"      ".lm.fit"        "colMeans"       "confint.lm"
[5] "contr.helmert"  "dummy.coef.lm"  "glm"           "glm.control"
[9] "glm.fit"         "KalmanForecast" "KalmanLike"     "KalmanRun"
[13] "KalmanSmooth"   "kappa.lm"       "lm"            "lm.fit"
[17] "lm.influence"   "lm.wfit"        "model.matrix.lm" "nlm"
[21] "nlminb"          "predict.glm"    "predict.lm"     "residuals.glm"
[25] "residuals.lm"   "summary.glm"   "summary.lm"
> |
```

# Worked Examples of Functions

To see a worked **example** just type the function name, e.g., **lm** for linear models:

```
example(lm)
```

and we see the printed and graphical output produced by the **lm** function.

R Console

```
Call:
lm(formula = weight ~ group - 1)

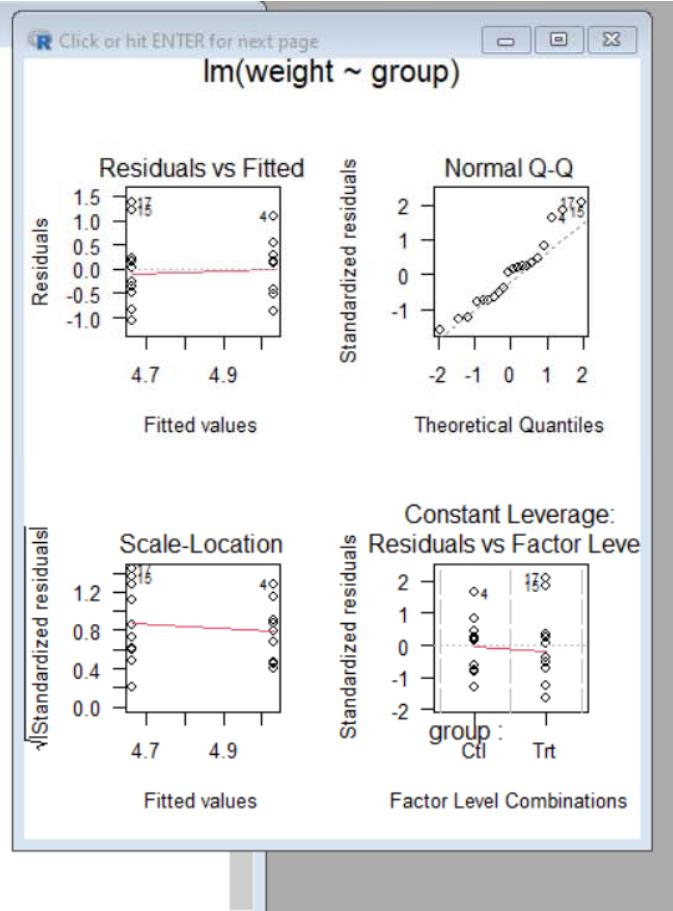
Residuals:
    Min      1Q  Median      3Q     Max 
-1.0710 -0.4938  0.0685  0.2462  1.3690 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
groupCtl   5.0320     0.2202  22.85 9.55e-15 ***
groupTrt   4.6610     0.2202  21.16 3.62e-14 ***  
---
Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1 

Residual standard error: 0.6964 on 18 degrees of freedom
Multiple R-squared:  0.9818, Adjusted R-squared:  0.9798 
F-statistic: 485.1 on 2 and 18 DF,  p-value: < 2.2e-16

lm> ## End(No test)
lm> opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))

lm> plot(lm.D9, las = 1)      # Residuals, Fitted, ...
Waiting to confirm page change...
```



...and other details follow further

# Demonstration of R Functions

This can be useful for seeing the type of things that R can do.

`demo(persp)` [persp is a command for 3d surface plots]

```
R R Console
> demo(persp)

demo(persp)
---- ~~~~~

Type <Return> to start :

> ### Demos for persp() plots -- things not in example(persp)
> #### -----
>
> require(datasets)

> require(grDevices); require(graphics)

> ## (1) The Obligatory Mathematical surface.
> ##      Rotated sinc function.
>
> x <- seq(-10, 10, length.out = 50)

> y <- x

> rotsinc <- function(x,y)
+ {
+   sinc <- function(x) { y <- sin(x)/x ; y[is.na(y)] <- 1; y }
+   10 * sinc( sqrt(x^2+y^2) )
+ }

> sinc.exp <- expression(z == Sinc(sqrt(x^2 + y^2)))

> z <- outer(x, y, rotsinc)

> oldpar <- par(bg = "white")

> persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
Waiting to confirm page change...
```

Click or hit ENTER for next page

$z = \text{Sinc}(\sqrt{x^2 + y^2})$

...and it continues

# Demonstration of R Functions

This can be useful for seeing the type of things that R can do.

`demo(graphics)`

```
R R Console
> demo(graphics)

  demo(graphics)
  ---- ~~~~~~

Type <Return> to start :

> # Copyright (C) 1997-2009 The R Core Team
>
> require(datasets)

> require(grDevices); require(graphics)

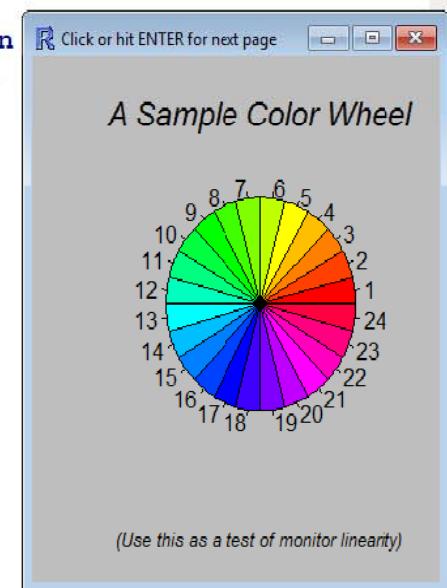
> ## A little color wheel.      This code just plots equally spaced hues in
> ## a pie chart.      If you have a cheap SVGA monitor (like me) you will
> ## probably find that numerically equispaced does not mean visually
> ## equispaced. On my display at home, these colors tend to cluster at
> ## the RGB primaries. On the other hand on the SGI Indy at work the
> ## effect is near perfect.
>
> par(bg = "gray")

> pie(rep(1,24), col = rainbow(24), radius = 0.9)
Waiting to confirm page change...

> title(main = "A Sample Color Wheel", cex.main = 1.4, font.main = 3)

> title(xlab = "(Use this as a test of monitor linearity)",
+       cex.lab = 0.8, font.lab = 3)

> ## We have already confessed to having these. This is just showing off X11
> ## color names (and the example (from the postscript manual) is pretty "cute".
>
```



# **Foundations of R Software**

## **Lecture 3** **Introduction**

::::

### **Packages and Libraries in R**

Shalabh

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# **What is a Library?**

**A library is a collection of books or media that are easily accessible for use and not just for display purposes only.**



- **Persons goes to library and search for suitable book.**
- **Get it issued and start using it.**
- **When done, return the book back to the library.**

# Libraries in R

R packages are the collections of programmes and data sets developed by different users and scientists. This increases the functionality and applications of R.

Commonly used important functions are added to base R functionalities.

If all the functionalities are added, the base package will become too big and every person may not like to use all the functionalities.

# **Libraries in R**

**Suppose someone is interested in cluster analysis.**

**It is not included in the base R package.**

**A package `cluster` is developed by the experts.**

**Whenever required, anyone can download it and use it for  
free!**

# Libraries in R

R provides many functions and one can also write own.

Functions and datasets are organised into libraries

To use a library, simply type the **library** function with the name of the library in brackets.

**library( . )**

For example, to load the **spatial** library type:

**library(spatial)**

# **Libraries in R**

**There are two type of libraries in R.**

**Some libraries are the part of base package.**

**Other libraries are user dependent and need based.**

# Libraries in R

Examples of libraries that come as a part of base package in R.

**MASS** : package associated with Venables and Ripley's book entitled *Modern Applied Statistics using S-Plus*.

**mgcv** : generalized additive models.

# Libraries in R

Examples of libraries that do not come as a part of base package in R.

**spatial** : Used in Spatial analysis in statistics.

**boot** : Used in Bootstrapping analysis in statistics.

# Libraries in R

**library( )** is the command used to load a package, and it refers to the place where the package is contained.

A package is the collection of functions bundled conveniently.

# Description of Libraries

The command `packageDescription()` provides the description file of a package.

Here is how we find the description of the `spatial` library:

`packageDescription("spatial")` returns

```
R Console
> packageDescription("spatial")
Package: spatial
Priority: recommended
Version: 7.3-14
Date: 2021-04-17
Depends: R (>= 3.0.0), graphics, stats, utils
Suggests: MASS
Authors@R: c(person("Brian", "Ripley", role = c("aut", "cre", "cph"),
  email = "ripley@stats.ox.ac.uk"), person("Roger", "Bivand", role =
  "ctb"), person("William", "Venables", role = "cph"))
Description: Functions for kriging and point pattern analysis.
Title: Functions for Kriging and Point Pattern Analysis
LazyLoad: yes
ByteCompile: yes
License: GPL-2 | GPL-3
```

followed by all the other details.

# Contents of Libraries

It is easy to use the `help` function to discover the contents of library packages.

Here is how we find out about the contents of the `spatial` library:

`library(help=spatial)` returns

Information on package 'spatial'

Description:

```
Package:           spatial
Priority:         recommended
Version:          7.3-14
Date:             2021-04-17
Depends:          R (>= 3.0.0), graphics, stats, utils
Suggests:         MASS
Authors@R:        c(person("Brian", "Ripley", role = c("aut", "cre",
                                                    "cph"), email = "ripley@stats.ox.ac.uk"),
                      person("Roger", "Bivand", role = "ctb"))
```

followed by a list of all the functions and data sets.

Then we get....

```
R Console
> library(help=spatial) |
> |
  R Documentation for package 'spatial'

Information on package 'spatial'

Description:
  This package provides functions for kriging and point pattern analysis.

  It includes functions for spatial prediction (kriging), spatial
  data structures, spatial statistics, and spatial point patterns.

  The package depends on R (>= 3.0.0), graphics, stats, and utils.

  It suggests MASS.

  Authors@R:
    Brian D. Ripley [aut, cre, cph]
    Roger S. Bivand [ctb]
    William N. Venables [cph]

  Description: Functions for kriging and point pattern analysis.
  Title: Functions for Kriging and Point Pattern Analysis
  LazyLoad: yes
  ByteCompile: yes
  License: GPL-2 | GPL-3
```

# **Installing Packages and Libraries**

**The base R package contains programs for basic operations.**

**It does not contain some of the libraries necessary for advanced statistical work.**

**Specific requirements are met by special packages.**

**They are downloaded and their downloading is very simple.**

# Installing Packages and Libraries

To install any package,

- run the R program,
- then on the command line, use the `install.packages` function to download the libraries we want.

# Installing Packages and Libraries

Examples :

- The package **boot** contains the statistical tools for bootstrap method.
- The package **cluster** contains statistical tools for clustering.

The packages **boot** or **cluster** can be installed by

```
install.packages( "boot" )
```

```
install.packages( "cluster" )
```

Then we get<sup>15</sup>...

```
install.packages("boot")
```

R Console

```
> install.packages("boot")
Installing package into 'C:/Users/Shalabh/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
```

Secure CRAN mirrors

- 0-Cloud [https]
- Australia (Canberra) [https]
- Australia (Melbourne 1) [https]
- Australia (Melbourne 2) [https]
- Australia (Perth) [https]**
- Austria [https]
- Belgium (Brussels) [https]
- Brazil (BA) [https]
- Brazil (PR) [https]
- Brazil (RJ) [https]
- Brazil (SP 1) [https]
- Brazil (SP 2) [https]
- Bulgaria [https]
- Canada (MB) [https]
- Canada (ON 2) [https]
- Canada (ON 3) [https]
- Chile (Santiago) [https]
- China (Beijing 2) [https]

Choose any country

```
R Console

> install.packages("boot")
Installing package into 'C:/Users/Shalabh/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
trying URL 'https://cran.curtin.edu.au/bin/windows/contrib/4.1/boot_1.3-28.zip'
Content type 'application/zip' length 641355 bytes (626 KB)

61% downloaded
URL: ... //cran.curtin.edu.au/bin/windows/contrib/4.1/boot_1.3-28.zip
```

```
R Console

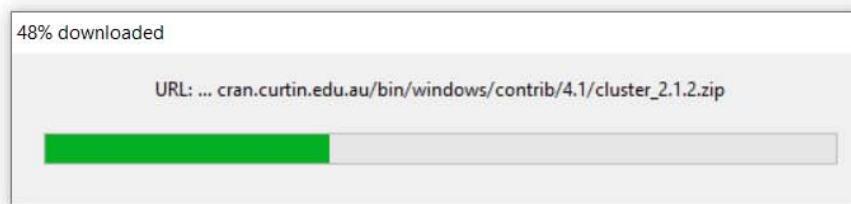
> install.packages("boot")
Installing package into 'C:/Users/Shalabh/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cran.curtin.edu.au/bin/windows/contrib/4.1/boot_1.3-28.zip'
Content type 'application/zip' length 641355 bytes (626 KB)
downloaded 626 KB

package 'boot' successfully unpacked and MD5 sums checked
The downloaded binary packages are in
  C:\Users\Shalabh\AppData\Local\Temp\RtmpuAOPUm\downloaded_packages
> |
```

```
install.packages("cluster")
```

```
R Console
```

```
> install.packages("cluster")
Installing package into 'C:/Users/Shalabh/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
trying URL 'https://cran.curtin.edu.au/bin/windows/contrib/4.1/cluster_2.1.2.zip'
Content type 'application/zip' length 617983 bytes (603 KB)
```



```
R Console
```

```
> install.packages("cluster")
Installing package into 'C:/Users/Shalabh/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
trying URL 'https://cran.curtin.edu.au/bin/windows/contrib/4.1/cluster_2.1.2.zip'
Content type 'application/zip' length 617983 bytes (603 KB)
downloaded 603 KB
```

```
package 'cluster' successfully unpacked and MD5 sums checked
```

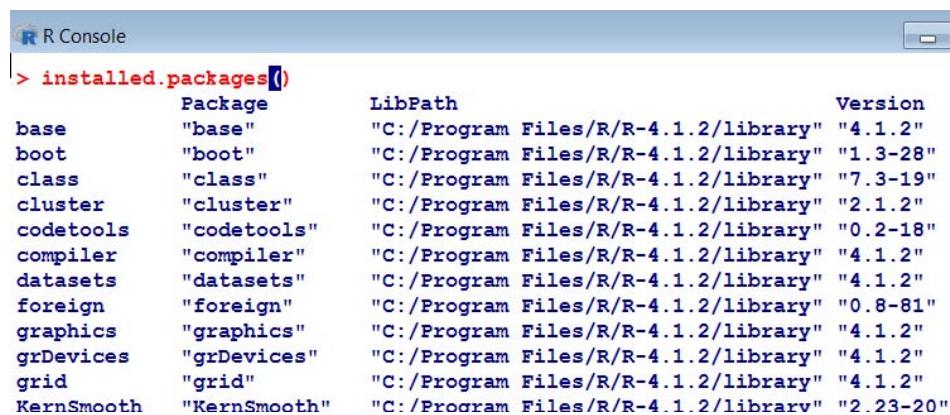
```
The downloaded binary packages are in
      C:\Users\Shalabh\AppData\Local\Temp\RtmpuAOPUm\downloaded_packages
> |
```

# Which packages are installed on my computer

The command `installed.packages()` is used to see the installed packages on the computer.

```
> installed.packages()  
> installed.packages()
```

Version	Package	LibPath
base	"base"	"C:/Program Files/R/R-4.1.2/library" "4.1.2"
boot	"boot"	"C:/Program Files/R/R-4.1.2/library" "1.3-28"
class	"class"	"C:/Program Files/R/R-4.1.2/library" "7.3-19"
cluster	"cluster"	"C:/Program Files/R/R-4.1.2/library" "2.1.2"
codetools	"codetools"	"C:/Program Files/R/R-4.1.2/library" "0.2-18"
compiler	"compiler"	"C:/Program Files/R/R-4.1.2/library" "4.1.2" ...



## Uninstalling packages installed on my computer

The command `remove.packages("package")` is used to see remove the installed packages on the computer.

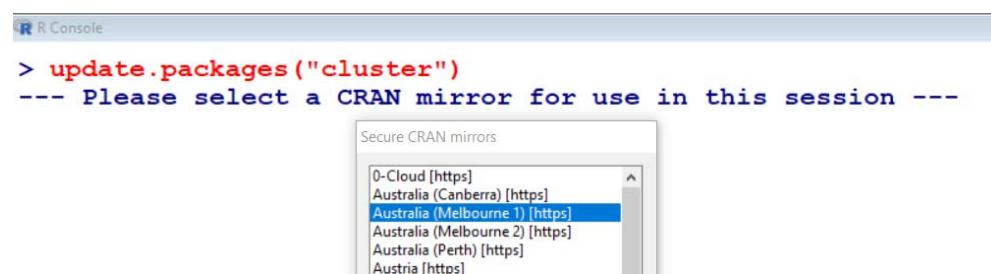
For example, if we want to remove the package `cluster` from the computer. Then use

```
remove.packages("cluster")
```

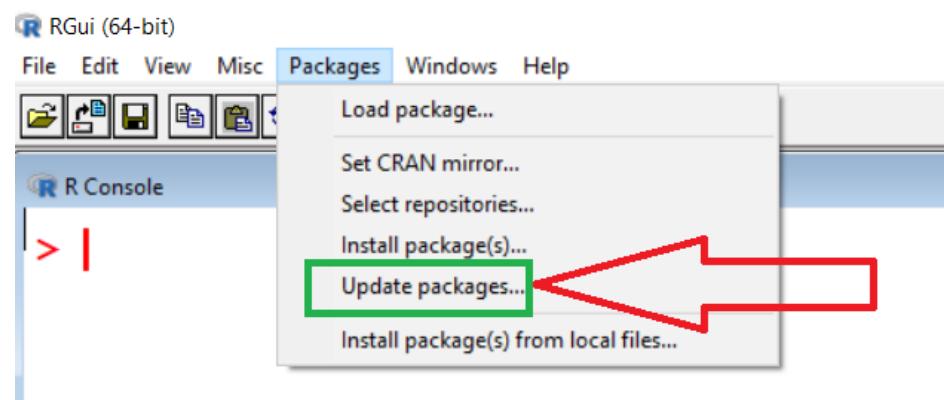
# Updating packages installed on my computer

The command `update.packages()` is used to see update the installed packages on the computer.

`update.packages("cluster")`



Alternatively, go to R console

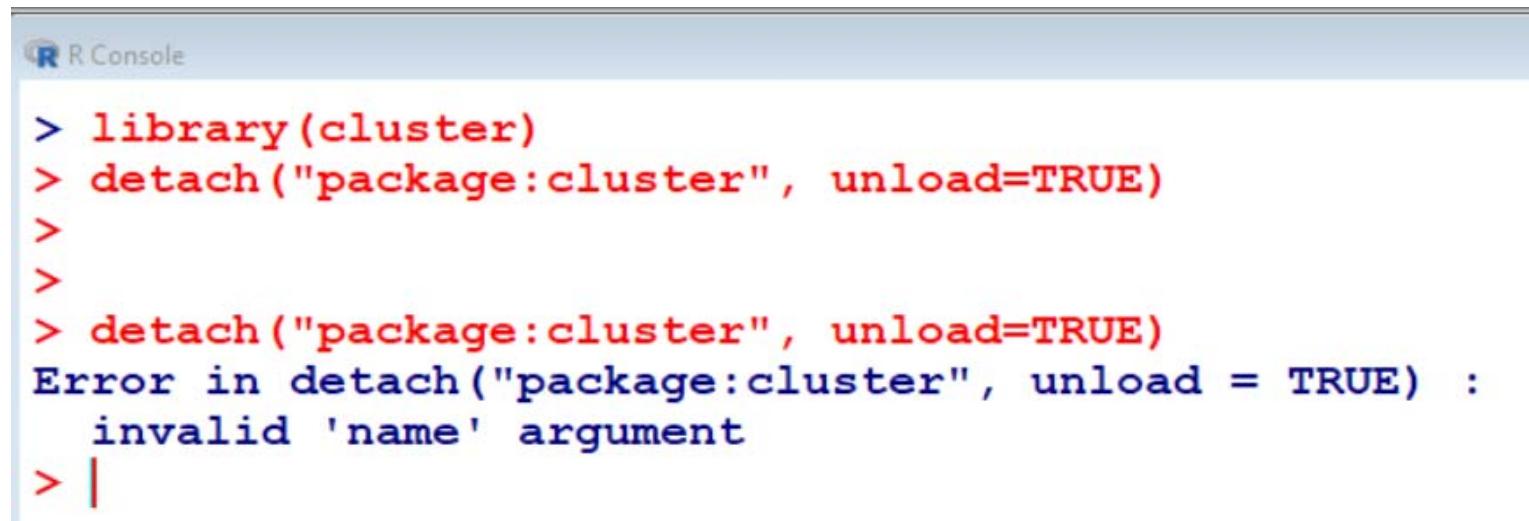


# Unloading packages installed on my computer

The command

```
detach("package:cluster", unload=TRUE)
```

is used to unload the installed packages on the computer.



A screenshot of an R console window titled "R Console". The console displays the following R session:

```
> library(cluster)
> detach("package:cluster", unload=TRUE)
>
>
> detach("package:cluster", unload=TRUE)
Error in detach("package:cluster", unload = TRUE) :
  invalid 'name' argument
> |
```

The console shows that the first attempt to detach the package succeeds, but the second attempt fails with an error message: "invalid 'name' argument".

# **Foundations of R Software**

## **Lecture 4** **Introduction**

::::

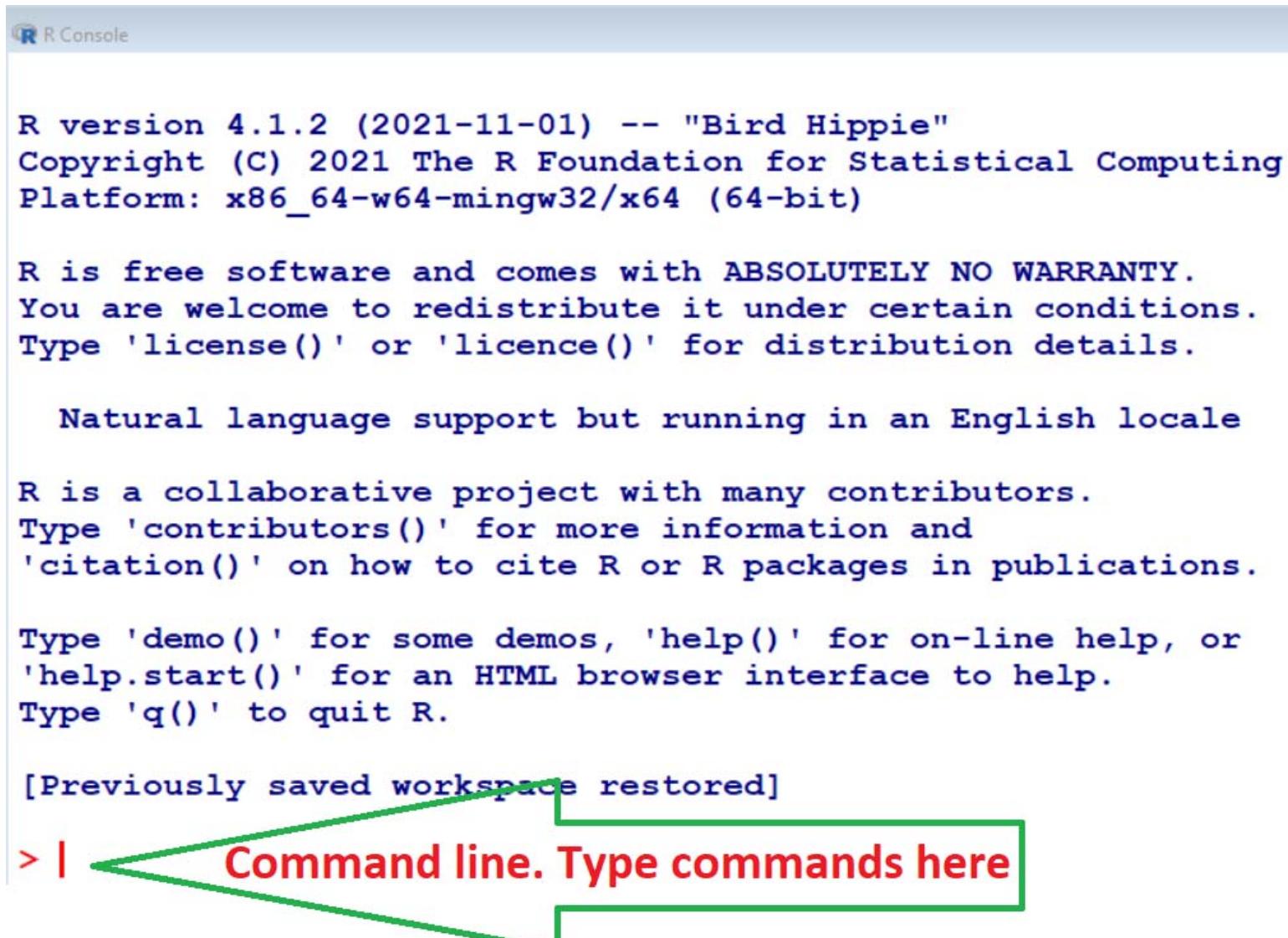
## **Command Line and Data Editor**

Shalabh

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Command Line versus Scripts

## What is command line?



R version 4.1.2 (2021-11-01) -- "Bird Hippie"  
Copyright (C) 2021 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.

[Previously saved workspace restored]

> | **Command line. Type commands here**

A green callout box highlights the text "Command line. Type commands here" in red, which is located at the bottom of the screenshot. A green arrow points from the left towards this text, and another green arrow points upwards from the bottom towards the word "Command line".

# **Command Line versus Scripts**

**Execution of commands in R is not menu driven.**

**(Not like Clicking over buttons to get outcome)**

**We need to type the commands.**

**Single line and multi line commands are possible to write.**

**When writing multi-line programs, it is useful to use a text editor rather than execute everything directly at the command line.**

# Command Line versus Scripts

## Option 1:

One may use R's own built-in editor.

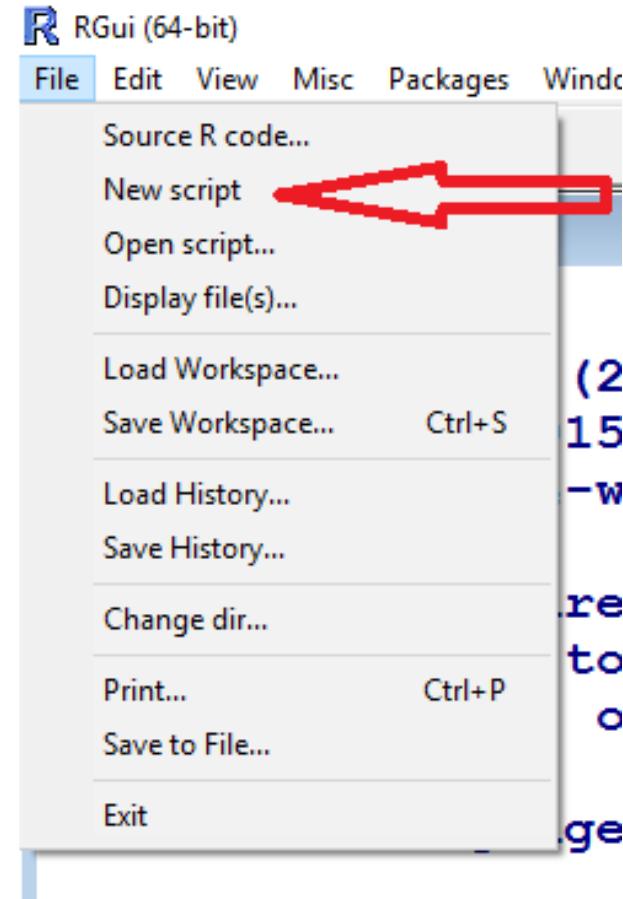
It is accessible from the **RGui** menu bar.

Click **File** and then click on **New script**.

# Command Line versus Scripts

At this point R will  
open a window entitled  
**Untitled-R Editor.**

We may type and edit in this.



If we want to execute a line or a group of lines, just highlight them and press **Ctrl+R**.

# **Command Line versus Scripts**

**Option 2:**

**Use other editor software which help running of R.**

**Different editors are available- R studio, Tinn R etc.**

**They are free software.**

**R Studio is an editor which helps working in R.**

**Tinn R is another editor available at**

**(<https://sourceforge.net/projects/tinn-r>)**

# **Installing R Studio**

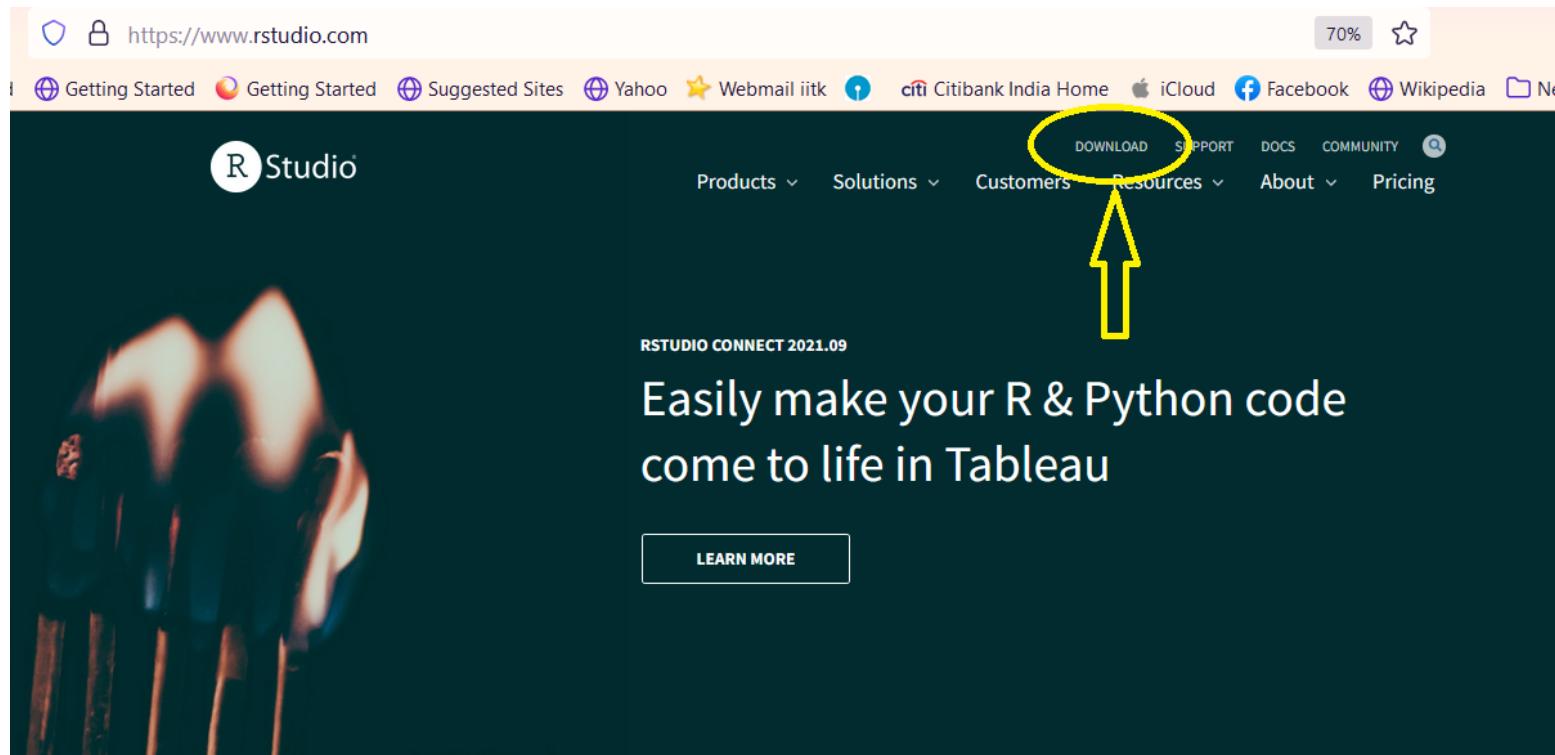
**R Studio is a software which helps in running the R software.**

**R Studio is written in C++ programming language.**

**R Studio is a free and open-source integrated development environment (IDE) for R.**

**Download R Studio software from website  
<https://www.rstudio.com/>**

# Installing R Studio



Click on Download.

# Installing R Studio

The screenshot shows the RStudio download page. At the top, there's a navigation bar with links like 'Getting Started', 'Suggested Sites', 'Yahoo', 'Webmail iitk', 'Citibank India Home', 'iCloud', 'Facebook', and 'Wikipedia'. Below the navigation is the RStudio logo and a menu bar with 'Products', 'Solutions', 'Customers', 'Resources', 'About', and 'Pricing'. A search icon is also present.

The main banner features a blue background with hexagonal patterns containing various R packages like 'tidyverse', 'Shiny', 'dplyr', etc., and the text 'Download the RStudio IDE'.

**Choose Your Version**

The RStudio IDE is a set of integrated tools designed to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace.

[LEARN MORE ABOUT THE RSTUDIO IDE](#)

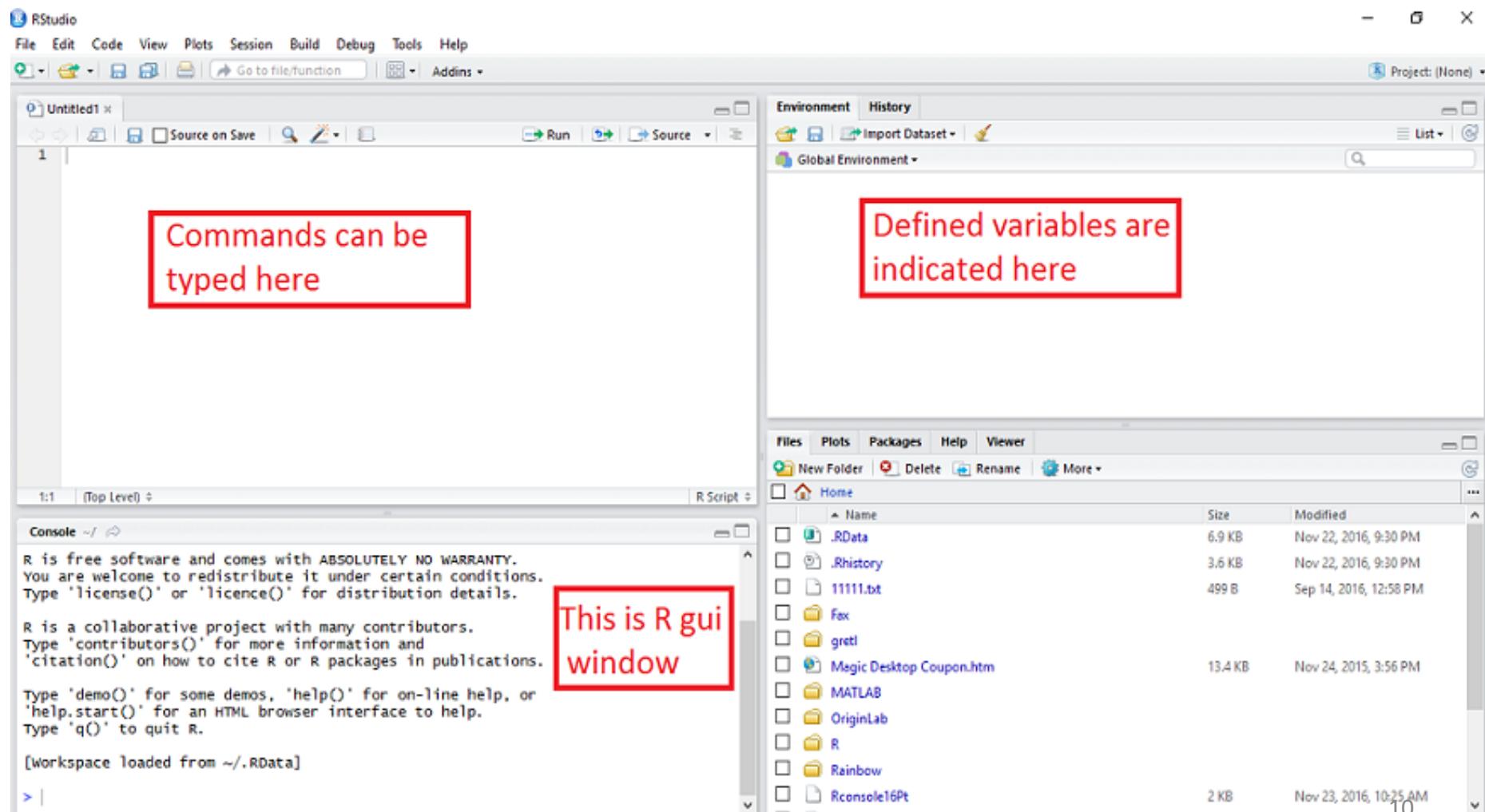
A red arrow points from the 'LEARN MORE' button down to the 'RStudio Desktop' option.

<b>RStudio Desktop</b> Open Source License <b>Free</b>	<b>RStudio Desktop Pro</b> Commercial License <b>\$995</b> /year	<b>RStudio Server</b> Open Source License <b>Free</b>	<b>RStudio Workbench</b> Commercial License <b>\$4,975</b> /year (5 Named Users)
--	---	---	--

**Download and double click on the downloaded file.**

# Command Line versus Scripts

Use R Studio software.



# **Foundations of R Software**

## **Lecture 5** **Introduction**

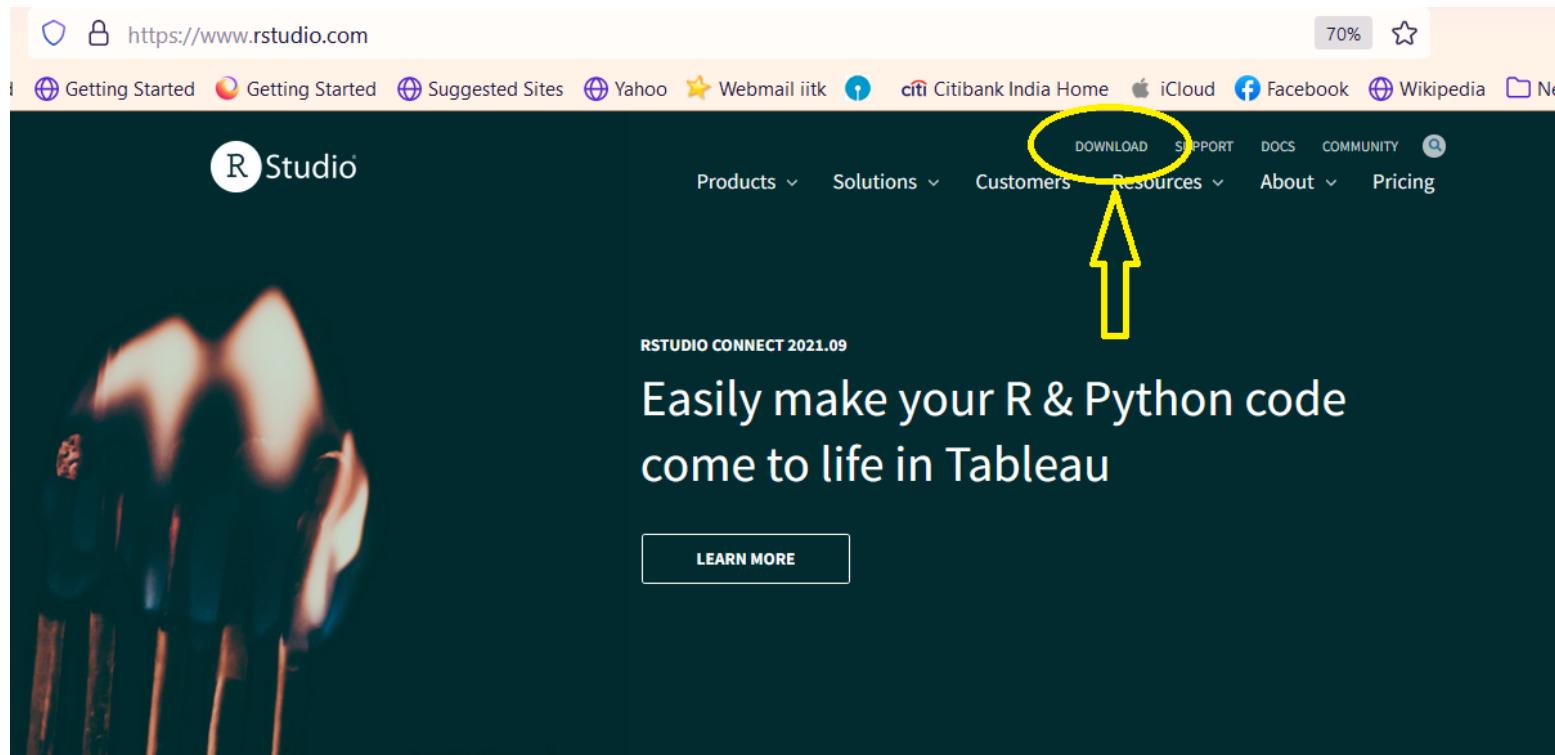
::::

## **Introduction to R Studio**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Installing R Studio



Click on Download.

# Installing R Studio

The screenshot shows the RStudio download page. At the top, there's a navigation bar with links like 'Getting Started', 'Suggested Sites', 'Yahoo', 'Webmail iitk', 'Citibank India Home', 'iCloud', 'Facebook', and 'Wikipedia'. Below the navigation is the RStudio logo and a menu bar with 'Products', 'Solutions', 'Customers', 'Resources', 'About', and 'Pricing'. A search icon is also present.

The main banner features a blue background with hexagonal patterns containing various R packages like 'tidyverse', 'Shiny', 'readr', 'dplyr', etc., and the text 'Download the RStudio IDE'.

**Choose Your Version**

The RStudio IDE is a set of integrated tools designed to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace.

[LEARN MORE ABOUT THE RSTUDIO IDE](#)

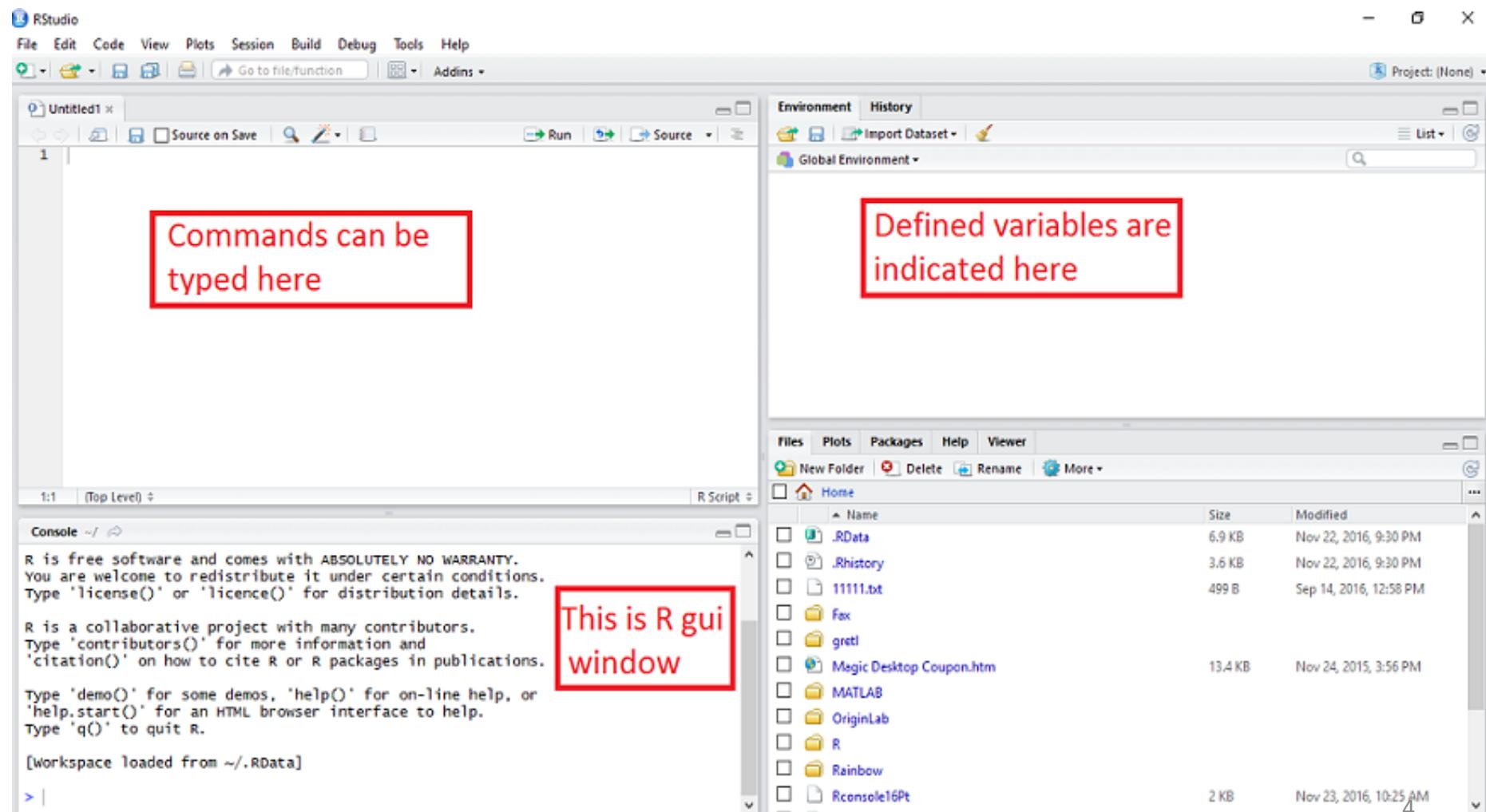
A red arrow points from the 'LEARN MORE' button down to the 'RStudio Desktop' option.

<b>RStudio Desktop</b> Open Source License <b>Free</b>	<b>RStudio Desktop Pro</b> Commercial License <b>\$995</b> /year	<b>RStudio Server</b> Open Source License <b>Free</b>	<b>RStudio Workbench</b> Commercial License <b>\$4,975</b> /year (5 Named Users)
--	---	---	--

**Download and double click on the downloaded file.**

# Command Line versus Scripts

Use R Studio software.



# Command Line versus Scripts

Suppose we want to use following three functions:

Type them.

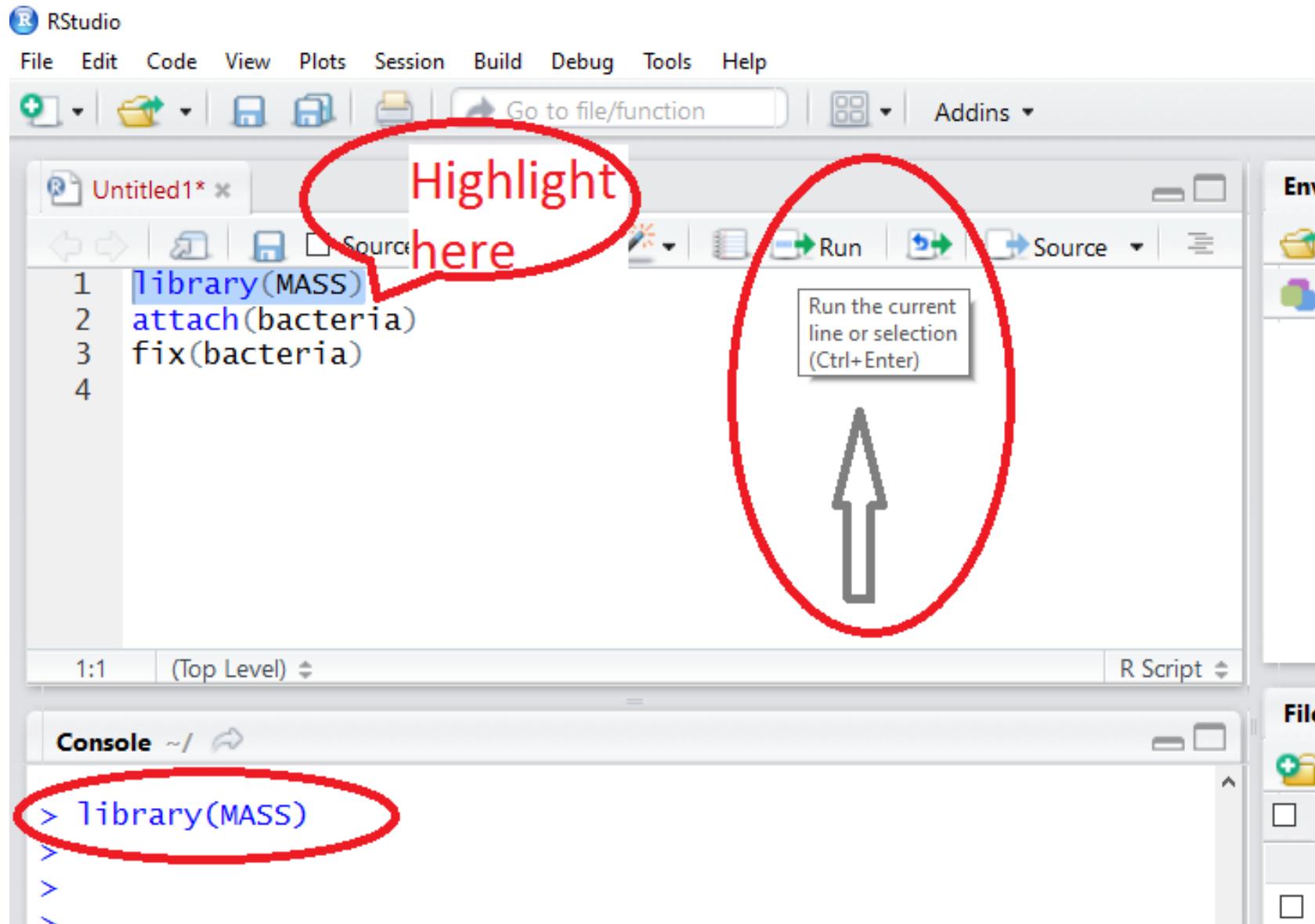
```
library(MASS)  
attach(bacteria)  
fix(bacteria)
```

Suppose we want to run only function: `library(MASS)`

Highlight it and click on `Run`

Then we get....

# Command Line versus Scripts



## Data Editor

There is a data editor within R that can be accessed from the menu bar by selecting Edit/Data editor.

Provide the name of the matrix or data frame that we want to edit and a **Data Editor** window appears.

Alternatively we can do this from the command line using the **fix** function.

### Example:

```
library(MASS)  
attach(bacteria)  
fix(bacteria)
```

# Data Editor

We can do it in R Studio as follows :

The screenshot shows the R Studio interface. In the top-left, there's a code editor window titled "Untitled1\*" containing R code:

```
library(MASS)
attach(bacteria)
fix(bacteria)
```

A red oval highlights the word "library" in the first line of code, with the text "Highlight here" written over it.

To the right of the code editor is a "Data Editor" window. It has a title bar "Data Editor" and a table with the following data:

	y	ap	hilo	week	ID	trt
1	y	p	hi	0	X01	placebo
2	y	p	hi	2	X01	placebo
3	y	p	hi	4	X01	placebo
4	y	p	hi	11	X01	placebo
5	y	a	hi	0	X02	drug+
6	y	a	hi	2	X02	drug+
7	n	a	hi	6	X02	drug+
8	y	a	hi	11	X02	drug+
9	y	a	lo	0	X03	drug
10	y	a	lo	2	X03	drug
11	y	a	lo	4	X03	drug
12	y	a	lo	6	X03	drug

A red house-shaped callout points to the "Run" button in the toolbar above the code editor, with the text "Click here". A large red arrow points from this callout to the "Data Editor" window, with the text "We get this window". Another large red arrow points from the "Data Editor" window to a text box below it, with the text "This is the data in MASS".

# **Introduction to R Studio**

**It is an interface between R and us.**

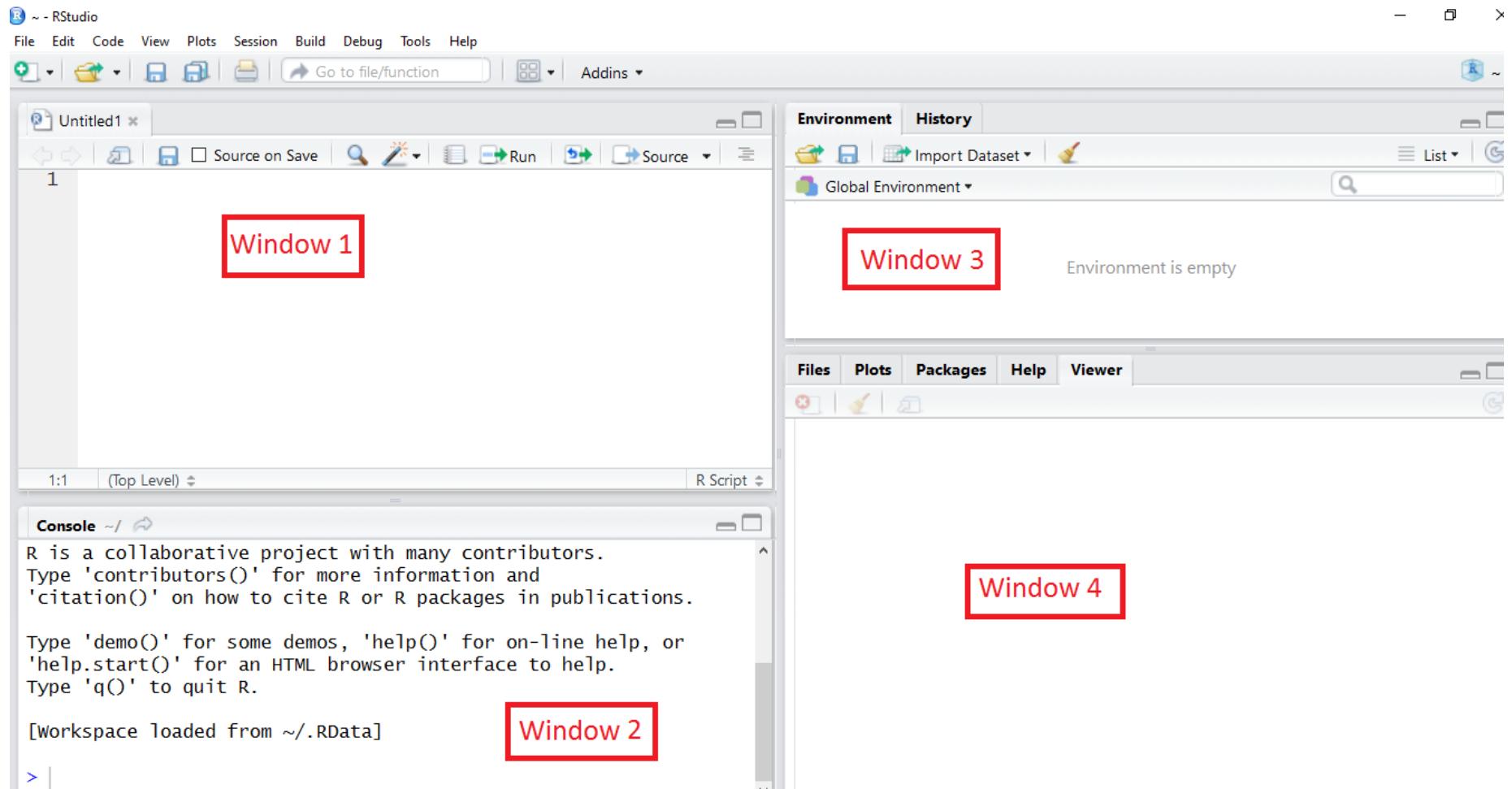
**More useful for beginners.**

**It makes coding easier.**

**When we start R studio, we see 4 windows**

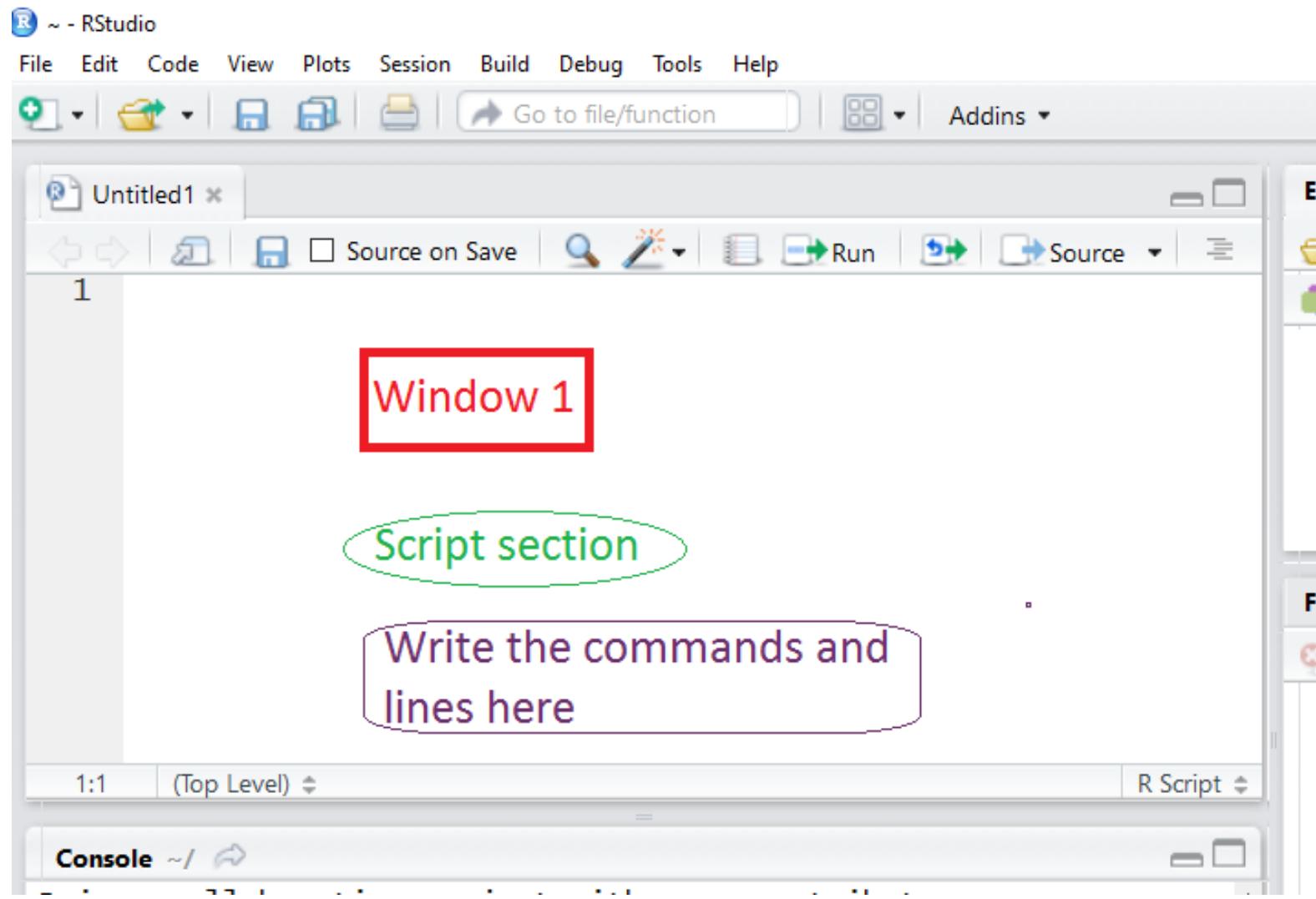
# Introduction to R Studio

First opening window of Rstudio is as follows having four windows.



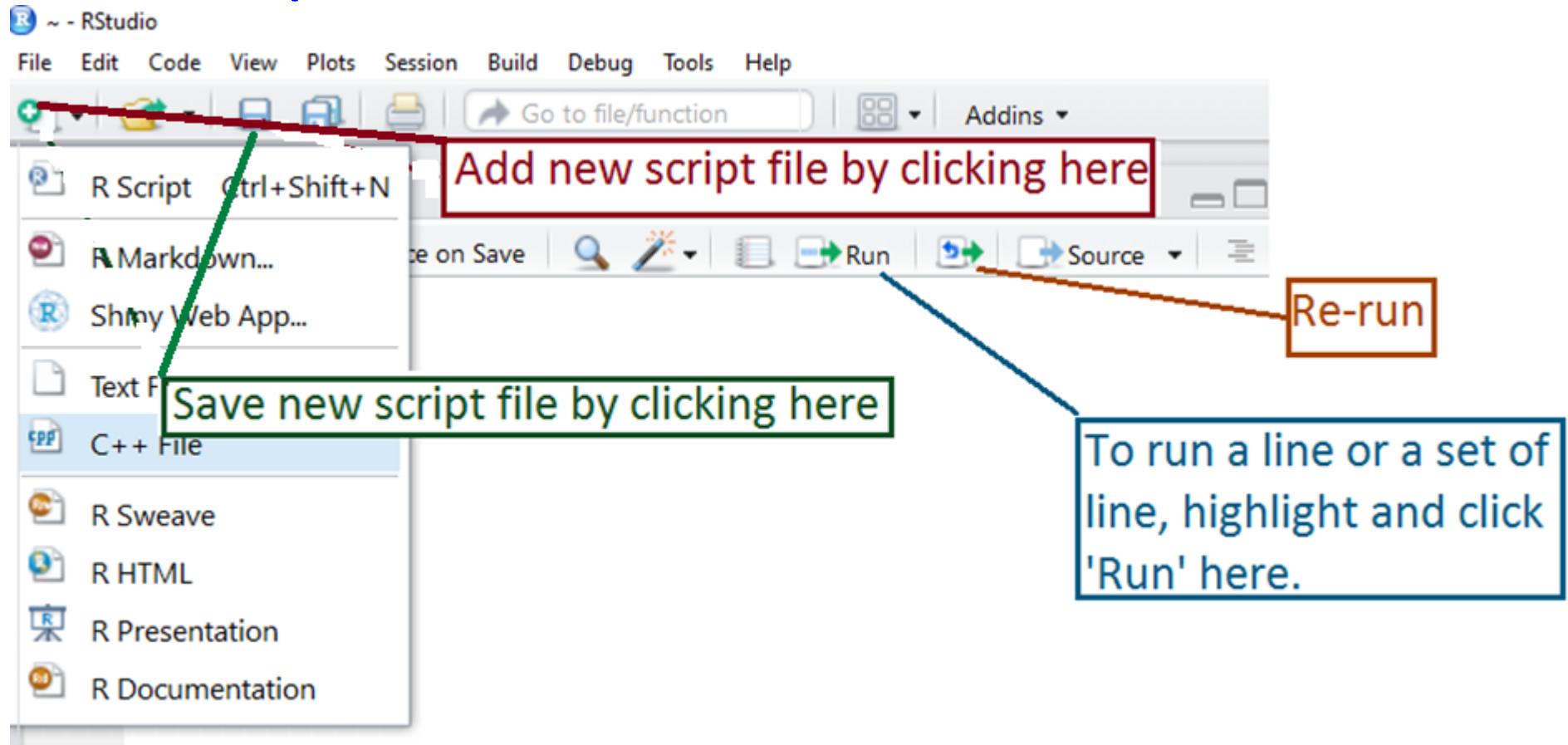
# Introduction to R Studio

## Description of Window 1



# Introduction to R Studio

## Description of Window 1



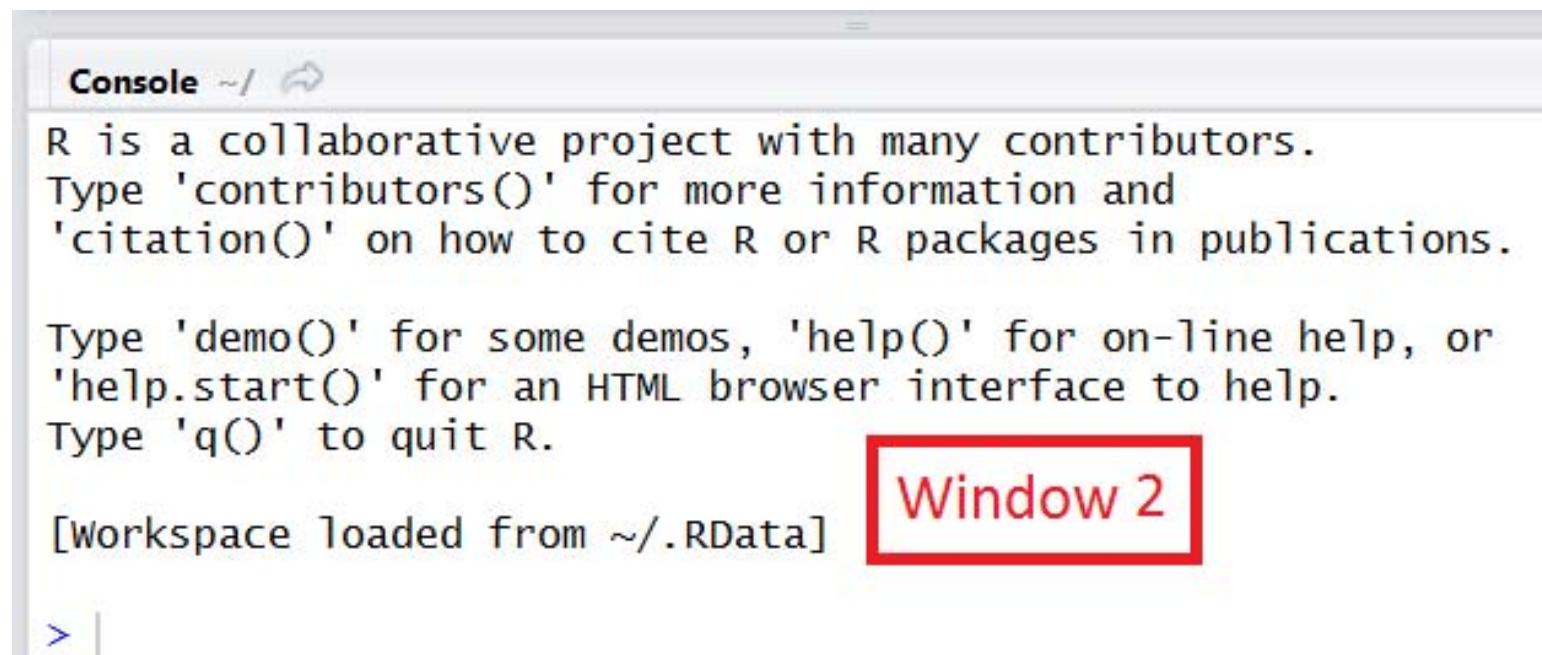
# Introduction to R Studio

## Description of Window 2 : Console

R program window appears here.

Calculations take place in console window.

One can write programmes in console also but it is hard to make corrections and experiments with the coding.



The screenshot shows the RStudio Console window. The title bar says "Console ~ /". The window contains the following text:

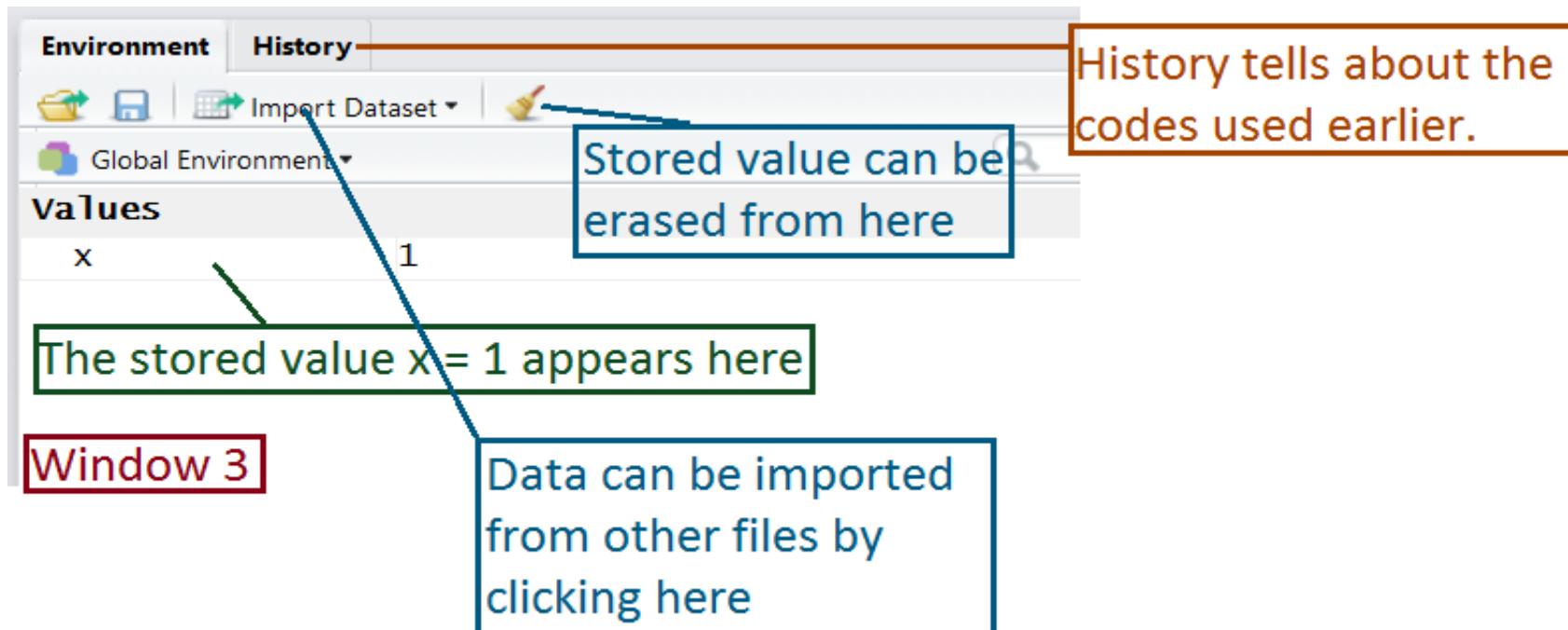
```
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
[workspace loaded from ~/.RData]
```

A red rectangular box highlights the word "Window" in the text "Window 2" located at the bottom right of the console area.

# Introduction to R Studio

## Description of Window 3 : Environment window

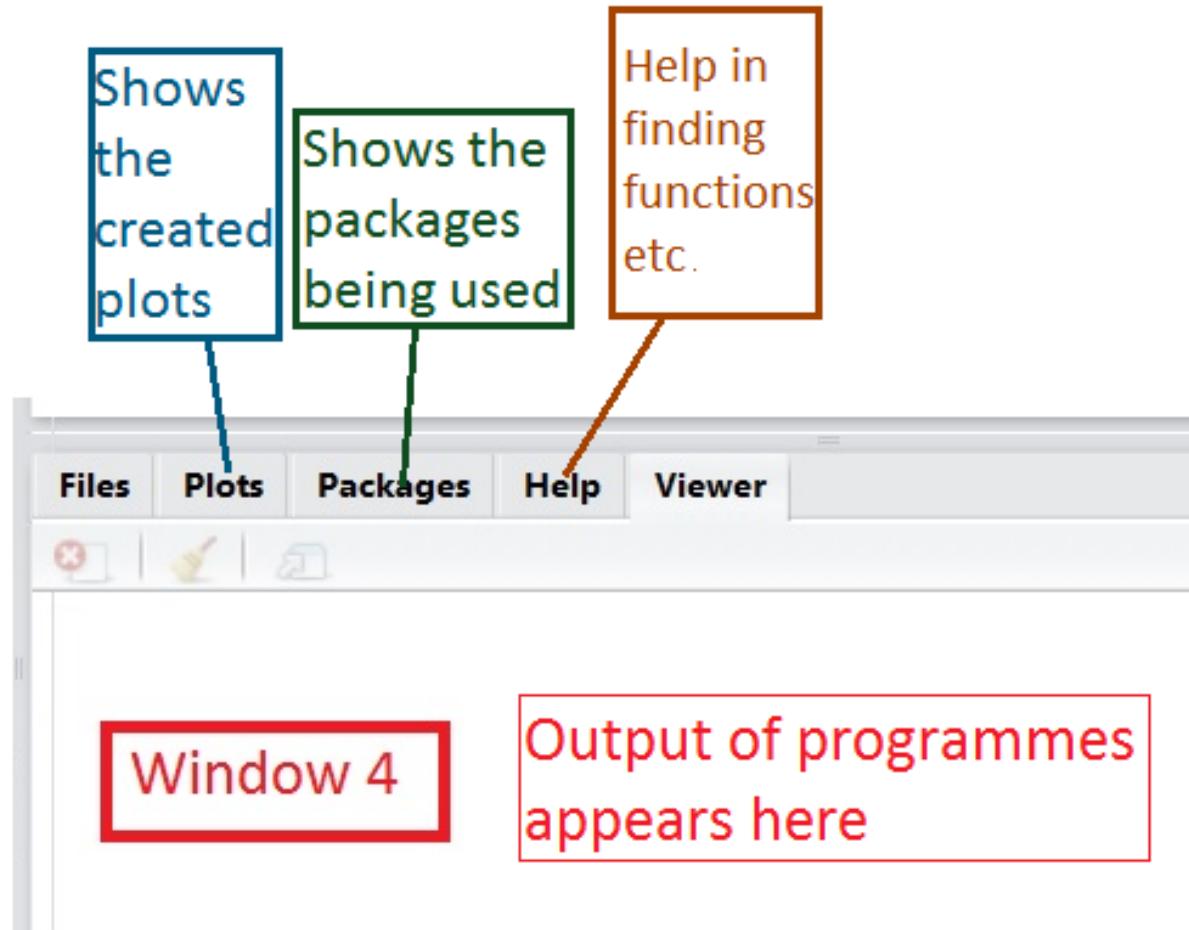
All the variables and objects used in the programme appear here.  
The nature and values of variables and objects also appear here.



# Introduction to R Studio

## Description of Window 4 : Output window

The output of programmes appears in this window.



# Introduction to R Studio

## Description of Window 4 : Output window

Packages:

All the packages being installed appear here.

Packages are not active.

Check mark in the boxes to activate them.

Mark check in boxes to activate the required packages

Name	Description	Version	Action
Packrat Library			
<input checked="" type="checkbox"/> <a href="#">Agreement</a>	Statistical Tools for Measuring Agreement	0.8-1	
<input checked="" type="checkbox"/> <a href="#">bayesm</a>	Bayesian Inference for Marketing/Micro-Econometrics	3.0-2	
<input type="checkbox"/> <a href="#">compositions</a>	Compositional Data Analysis	1.40-1	
<input type="checkbox"/> <a href="#">compute.es</a>	Compute Effect Sizes	0.2-4	
<input type="checkbox"/> <a href="#">DEoptimR</a>	Differential Evolution Optimization in Pure R	1.0-6	
<input type="checkbox"/> <a href="#">energy</a>	E-Statistics: Multivariate Inference via the Energy of Data	1.7-0	
<input type="checkbox"/> <a href="#">MAd</a>	Meta-Analysis with Mean Differences	0.8-2	
<input type="checkbox"/> <a href="#">meta</a>	General Package for Meta-Analysis	4.5-0	
<input type="checkbox"/> <a href="#">metafor</a>	Meta-Analysis Package for R	1.9-8	

# Introduction to R Studio

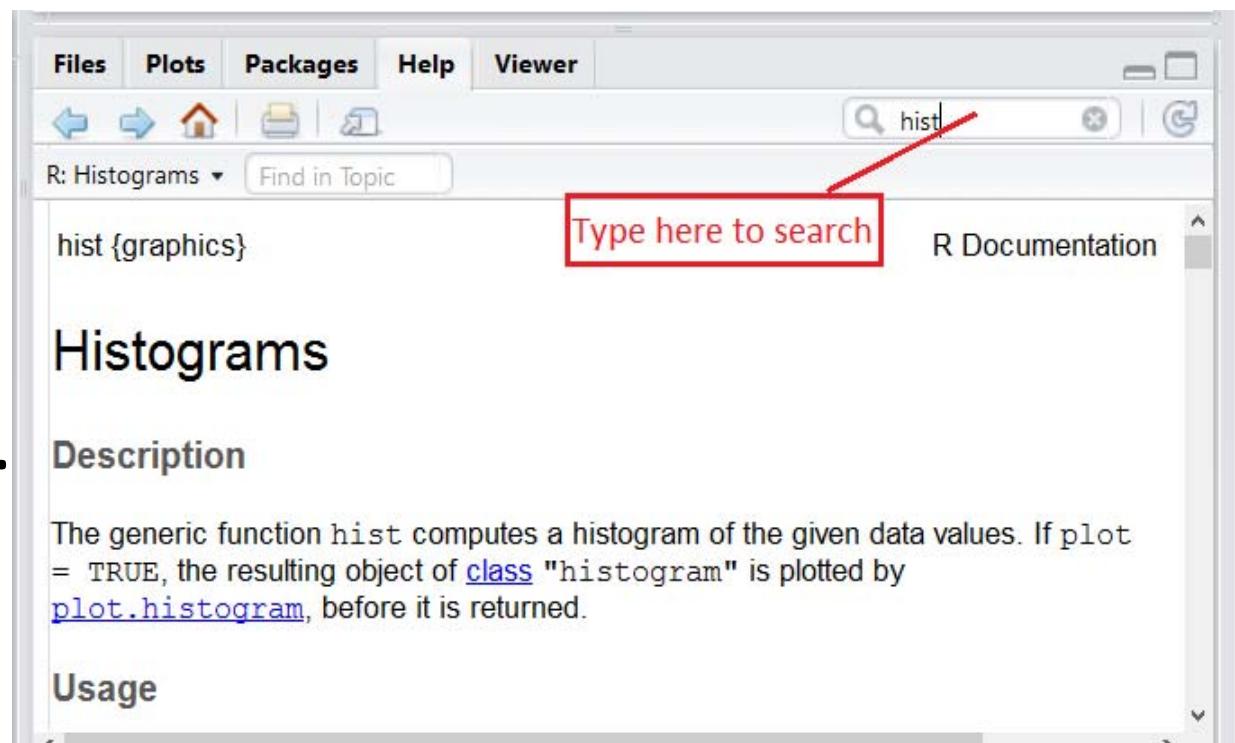
## Window 4 : Output window

Help:

Various types of help can be asked.

E.g., to know about histogram,  
type **hist**.

Information appears.



# Introduction to R Studio

**Example:**

**Bar diagram of values 1,2,1,1,2,3,1,2,3,1,2,2,3**

**R studio has following operation and output:**

# Introduction to R Studio

**Example:**

**Bar diagram of values**

```
x = c(1,2,1,1,2,3,1,2,3,1,2,2,3)  
barplot(table(x))
```

**R studio has following operation and output:**

# **Foundations of R Software**

## **Lecture 6**

## **Introduction**

::::

## **Basic Operations in R**

**Shalabh**

**Department of Mathematics and Statistics**  
**Indian Institute of Technology Kanpur**

# How to see the contents in the working directory in R

Type '`ls()`' to quit R.

Following are the contents in my computer's directory (not yours)

```
R Console
> ls()
[1] "corboott"
[3] "corrboot"
[5] "day"
[7] "fever"
[9] "hours"
[11] "marks"
[13] "marks_hours_data"
[15] "N"
[17] "N1"
[19] "N2"
[21] "N3"
[23] "oldpar"
[25] "power"
[27] "power_temp_data"
[29] "rashes"
[31] "rotsinc"
[43] "corr_power_temp"
[45] "covboot"
[47] "e"
[49] "gender"
[51] "mark_hrs"
[53] "marks_hours_boot"
[55] "n"
[57] "n1"
[59] "n2"
[61] "n3"
[63] "non_purulent_conjunctivitis"
[65] "p"
[67] "power_temp"
[69] "q"
[71] "rollno"
[73] "S1"
```

# How to see the location of the working directory in R

Use '`getwd()`' to find the current working directory in R.

Following is the current working directory in my computer (not yours).

```
R Console
> getwd()
[1] "C:/Users/Shalabh/Documents"
> |
```

# How to set the working directory in R

Use '`setwd()`' to change the working directory in R.

Suppose we want to set 'Rcourse' directory located on c drive.

Following is the command to change the working directory .

```
> setwd("c:/Rcourse")
> getwd()
[1] "c:/Rcourse"
```

```
R Console
> getwd()
[1] "C:/Users/Shalabh/Documents"
> setwd("c:/Rcourse")
> getwd()
[1] "c:/Rcourse"
> |
```

Be watchful in choosing the path command while working in Unix or Macintosh.

# **How to interrupt a running computation in R**

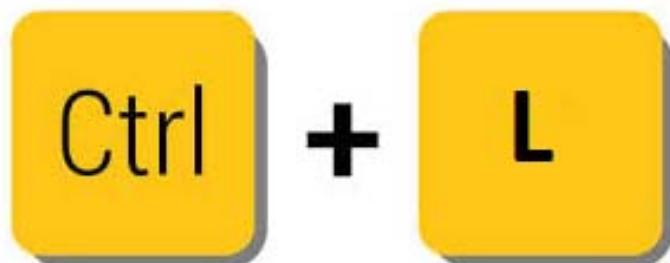
**To interrupt a long-running computation and return to the command prompt without exiting,**

**Press the Esc key on your keyboard**



# How to clean the GUI window in R

To clean the contents on the GUI window, press ctrl key and L key simultaneously.



# Search the web for information and answers regarding R.

To search the web for information and answers regarding R, use the **RSiteSearch** function to search by keyword or phrase.

This searches the phrase from <http://search.r-project.org>

For example, to know about 'mode' , type

**RSiteSearch( "mode" )**

Then we get....

R Console

```
> RSiteSearch("mode")
A search query has been submitted to http://search.r-project.org
The results page should open in your browser shortly
> |
```

# Search the web for information and answers regarding R.

The screenshot shows a web browser window with the URL <https://search.r-project.org/?P=mode&HITSPERPAGE=20&SORT=&DB=cran-help&DB=r-help&DB=cran-vignettes&DB=cran-vignettes>. The search bar contains the word "mode". Below the search bar, there are dropdown menus for "Sort by: relevance" and "Results per Page: 20". There are also checkboxes for "Search in: R Manuals", "Base Packages Help Pages | CRAN", "Task Views", "CRAN Packages", "General Info", "Help Pages", "News", "Readme", and "Vignettes".  
The search results are listed below, each with a red progress bar, a timestamp, a size, a title, a brief description, a link, and a "cran-help" matching note.

- Modified: 2021-11-11 Size: 2.7K [R: Sets the developer mode to help form check\\_arg\\_calls](#)  
...is pinpointed and the associated help is referred to. Usage setDreamerr\_dev.mode(dev.mode = FALSE) Arguments dev.mode A logical, default is FALSE. Details Since this mode ensures a detailed cheking...  
[/CRAN/refmans/dreamerr/html/setDreamerr\\_dev.mode.html](#)  
cran-help matching: mode, mode and mode
- Modified: 2021-11-11 Size: 1.2K [R: Check if the supplied 'mode' is a valid logging mode.](#)  
...mode is a valid logging mode. Description Check if the supplied mode is a valid logging mode. Usage check\_log\_mode(mode) Arguments mode mode how to log. Valid are tree, autodetect and compound...  
[/CRAN/refmans/beautier/html/check\\_log\\_mode.html](#)  
cran-help matching: mode, mode and mode
- Modified: 2021-11-11 Size: 1.8K [R: Create a one-mode projection of a two mode graph](#)  
project\_to\_one\_mode {birankr} R Documentation Create a one-mode projection of a two mode graph Description Create a one-mode projection of a two mode graph. Converts a rectangular matrix to a square...  
[/CRAN/refmans/birankr/html/project\\_to\\_one\\_mode.html](#)  
cran-help matching: mode, mode and mode
- Modified: 2021-11-11 Size: 2.2K [R: Collapse multiple modes into one mode.](#)  
...of lower order where the first mode indexes the modes indicated in m. Usage collapse\_mode(X, m) Arguments X An array whose modes we are collapsing. m A vector of integers giving the modes to collapse.  
[/CRAN/refmans/tensr/html/collapse\\_mode.html](#)  
cran-help matching: mode, mode and mode
- Modified: 2021-11-11 Size: 1.1K [R: Mode Allowing for Multi Modal Mode](#)  
...Mode Allowing for Multi Modal Mode Description Function calculating the mode, allowing for multiple modes in case of equal frequencies. Usage stats\_mode\_multi(x) Arguments x vector to get mode...  
[/CRAN/refmans/demvs/html/stats\\_mode\\_multi.html](#)

# What the numbers in [ ] present

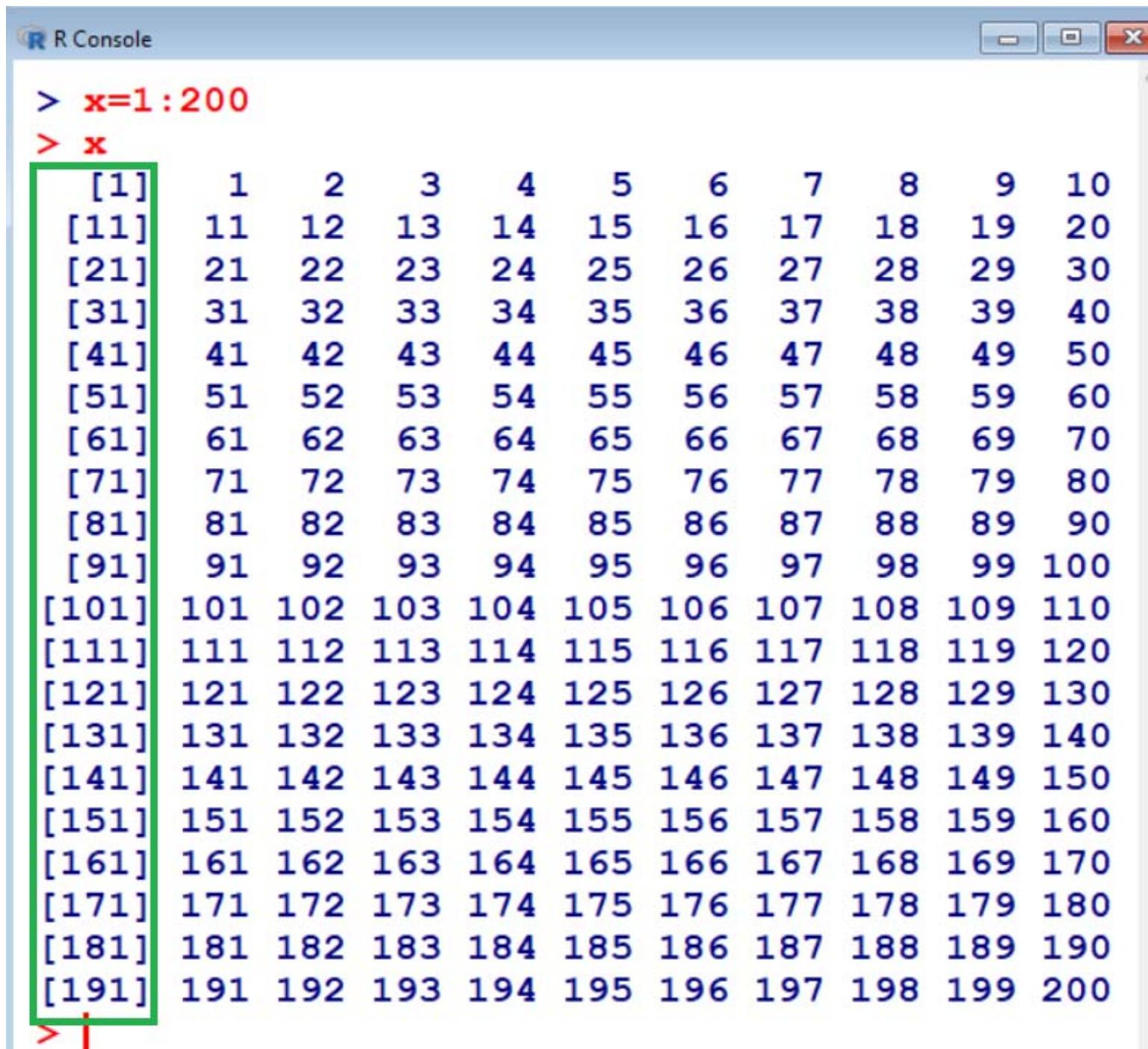
```
R Console
> x=1:200
> x
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
[163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
[181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
[199] 199 200
> |
```

The numbers in the first column inside[ ] present the position index.

They represent the count of the starting number.

This index depends upon the width of the GUI window.

# What the numbers in [ ] present



R Console window showing the output of the command `x=1:200`. The output is a matrix of 20 rows and 10 columns, representing the numbers 1 through 200 in a grid format. The first column is highlighted with a green border.

```
> x
```

[1]	1	2	3	4	5	6	7	8	9	10
[11]	11	12	13	14	15	16	17	18	19	20
[21]	21	22	23	24	25	26	27	28	29	30
[31]	31	32	33	34	35	36	37	38	39	40
[41]	41	42	43	44	45	46	47	48	49	50
[51]	51	52	53	54	55	56	57	58	59	60
[61]	61	62	63	64	65	66	67	68	69	70
[71]	71	72	73	74	75	76	77	78	79	80
[81]	81	82	83	84	85	86	87	88	89	90
[91]	91	92	93	94	95	96	97	98	99	100
[101]	101	102	103	104	105	106	107	108	109	110
[111]	111	112	113	114	115	116	117	118	119	120
[121]	121	122	123	124	125	126	127	128	129	130
[131]	131	132	133	134	135	136	137	138	139	140
[141]	141	142	143	144	145	146	147	148	149	150
[151]	151	152	153	154	155	156	157	158	159	160
[161]	161	162	163	164	165	166	167	168	169	170
[171]	171	172	173	174	175	176	177	178	179	180
[181]	181	182	183	184	185	186	187	188	189	190
[191]	191	192	193	194	195	196	197	198	199	200

```
>
```

# **Reloading packages when restarting R**

**If some packages are loaded through library and we exit from R.**

**When R is restarted, we need to reload the libraries.**

# Meaning of e in R

Any number like

**5.2345e+7** is  $5.2345 \times 10^7$

**5.2345e-7** is  $5.2345 \times 10^{-7}$

# Cleaning up the Windows

We assign names to variables when analyzing any data.

It is good practice to remove the variable names given to any data frame at the end each session in R.

This way, variables with same names but different properties will not get in each others way in subsequent work.

`rm( )` command removes variable names.

For example,

`rm(x,y,z)` removes the variables x, y and z.

# Cleaning up the Windows

`detach( )` command detaches objects from the Search Path

It removes it from the `search( )` path of available R objects.

Usually this is either a `data.frame` which has been attached or a package which was attached by `library`.

To get rid of everything, including data frames, type

```
rm(list=ls())
```

Then we get....

# Cleaning up the Windows

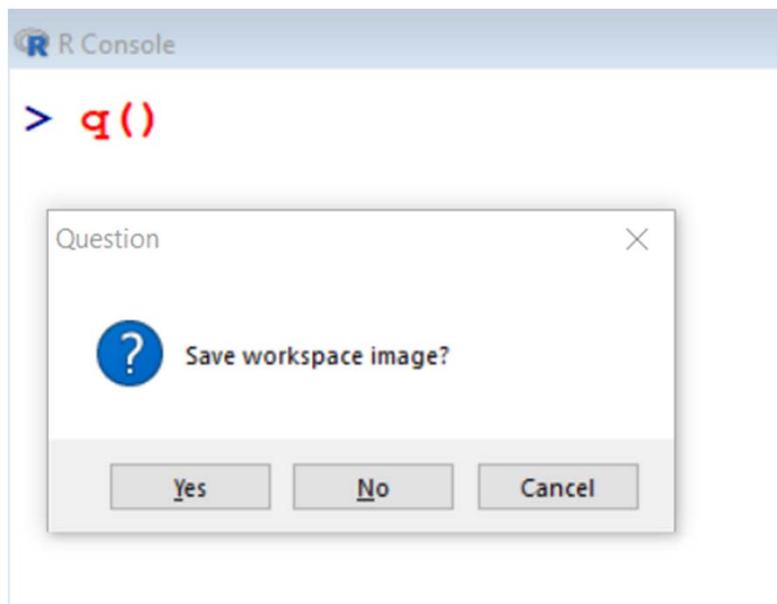
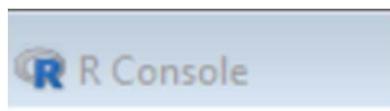
The screenshot shows the R Console interface with a toolbar at the top containing icons for file operations like Open, Save, Print, and Stop. The console window displays the following R session:

```
> library(cluster)
>
>
>
> detach(package: cluster)
> |
```

Two annotations are present: a green box labeled "Loads the cluster package" surrounds the first line of code, and another green box labeled "Detaches the cluster package" surrounds the fourth line of code.

# How to quit in R

Type '**q()**' to quit R.



# **Foundations of R Software**

## **Lecture 7 Introduction**

::::

## **Some more Basic Operations in R**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Assignment operator

- > is the prompt sign in R.
- The assignment operators are the left arrow with dash <- and equal sign =

> x <- 20 assigns the value 20 to x.

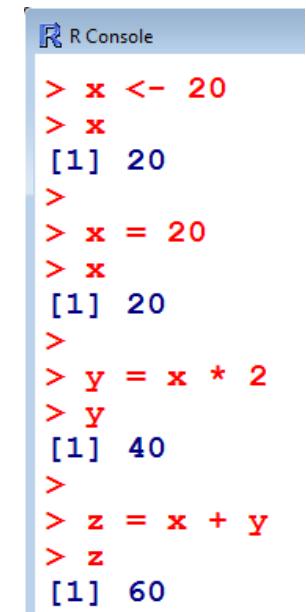
> x = 20 assigns the value 20 to x.

Initially only <- was available in R.

- > x = 20 assigns the value 20 to x.

> y = x \* 2 assigns the value 2\*x to y.

> z = x + y assigns the value x + y to z.



The image shows a screenshot of the R console window. The title bar says "R Console". The console area contains the following R session:

```
> x <- 20
> x
[1] 20
>
> x = 20
> x
[1] 20
>
> y = x * 2
> y
[1] 40
>
> z = x + y
> z
[1] 60
```

# Assignment of numbers and characters

- The numbers are assigned in the usual way

> `x <- 20` assigns the value 20 to `x`.

> `x = 20` assigns the value 20 to `x`.

- The characters assigned within double or single quotes

> `x = "apple"` assigns the apple to `x`.

> `x <- "apple1"` assigns the apple to `x`.

> `x = 'apple'` assigns the apple to `x`.

> `x <- 'apple'` assigns the apple to `x`.

# Assignment of numbers and characters

```
R Console

> x = 20
> x
[1] 20
> x <- 20
> x
[1] 20
> x=apple
Error: object 'apple' not found
> x = "apple"
> x
[1] "apple"
> x = 'apple'
> x
[1] "apple"
> |
```

# Knowing numbers and characters

To know if a value is a number, use `is.numeric()`

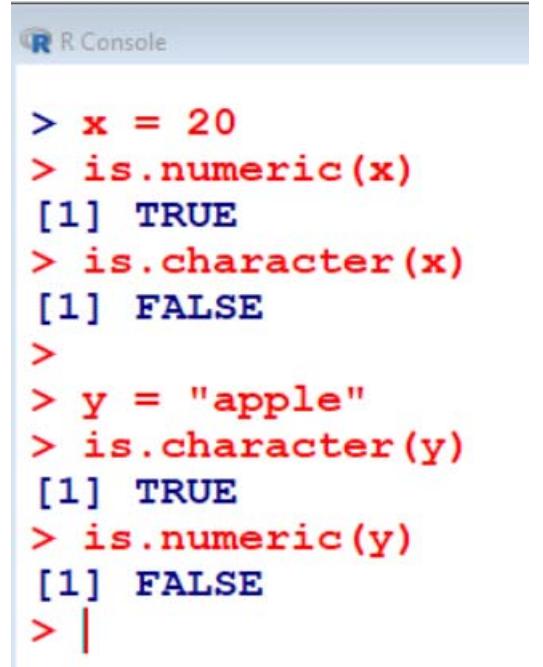
To know if a value is a character, use `is.character()`

```
x = 20
> is.numeric(x)
[1] TRUE

> is.character(x)
[1] FALSE

y = "apple"
> is.character(y)
[1] TRUE

> is.numeric(y)
[1] FALSE
```



A screenshot of an R console window titled "R Console". The window shows a series of R commands and their results. The commands are in red, and the results are in blue. The results for the first two lines are identical to the ones shown on the left. The third line shows that 'y' is a character vector. The fourth line shows that 'y' is not numeric. The fifth line is a blank line starting with '>'. The sixth line shows that 'y' is indeed numeric. The seventh line shows that 'y' is not numeric again, which is likely a mistake or a continuation of the previous command.

```
> x = 20
> is.numeric(x)
[1] TRUE
> is.character(x)
[1] FALSE
>
> y = "apple"
> is.character(y)
[1] TRUE
> is.numeric(y)
[1] FALSE
> |
```

# Converting numbers and characters

To convert a value as a number, use `as.numeric()`

To convert a value as a character, use `as.character()`

Converting a number into a character:

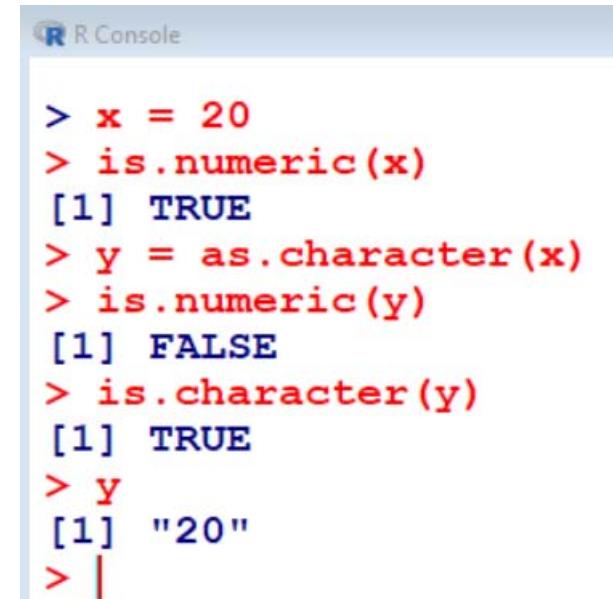
```
> x = 20
> is.numeric(x)
[1] TRUE

> y = as.character(x)
> is.numeric(y)
[1] FALSE

> is.character(y)
[1] TRUE

> y
[1] "20"

> y
[1] "20"
```



The screenshot shows an R console window titled "R Console". It displays the following R session:

```
R Console
> x = 20
> is.numeric(x)
[1] TRUE
> y = as.character(x)
> is.numeric(y)
[1] FALSE
> is.character(y)
[1] TRUE
> y
[1] "20"
> |
```

# Converting numbers and characters

Converting a character into a number:

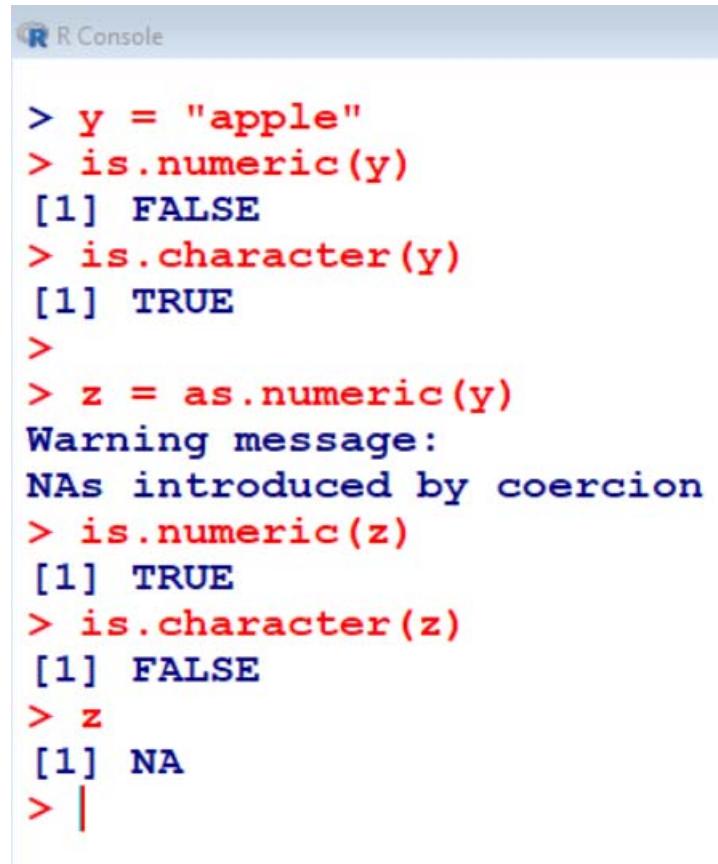
```
> y = "apple"
> is.numeric(y)
[1] FALSE

> is.character(y)
[1] TRUE

> z = as.numeric(y)
Warning message:
NAs introduced by coercion

> is.numeric(z)
[1] TRUE

> is.character(z)
[1] FALSE
> z
[1] NA
```



The screenshot shows the R console interface with a light blue header bar labeled "R Console". Below the header, there is a scrollable text area containing R code and its output. The code is identical to the one shown on the left, demonstrating the conversion of the character string "apple" to a numeric value z, which results in NA due to a warning message about NAs being introduced by coercion.

```
R Console
> y = "apple"
> is.numeric(y)
[1] FALSE
> is.character(y)
[1] TRUE
>
> z = as.numeric(y)
Warning message:
NAs introduced by coercion
> is.numeric(z)
[1] TRUE
> is.character(z)
[1] FALSE
> z
[1] NA
> |
```

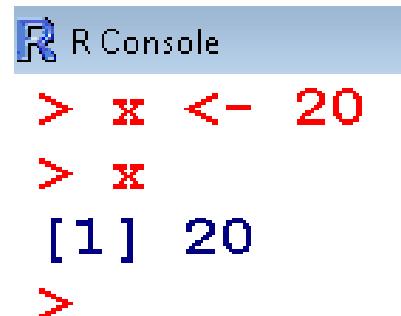
# Comment operator

# : The character # marks the beginning of a comment.

All characters until the end of the line are ignored.

```
> # mu is the mean
```

```
> # x <- 20 is treated as comment only
```



A screenshot of the R console window. The title bar says "R Console". The console area contains the following text:

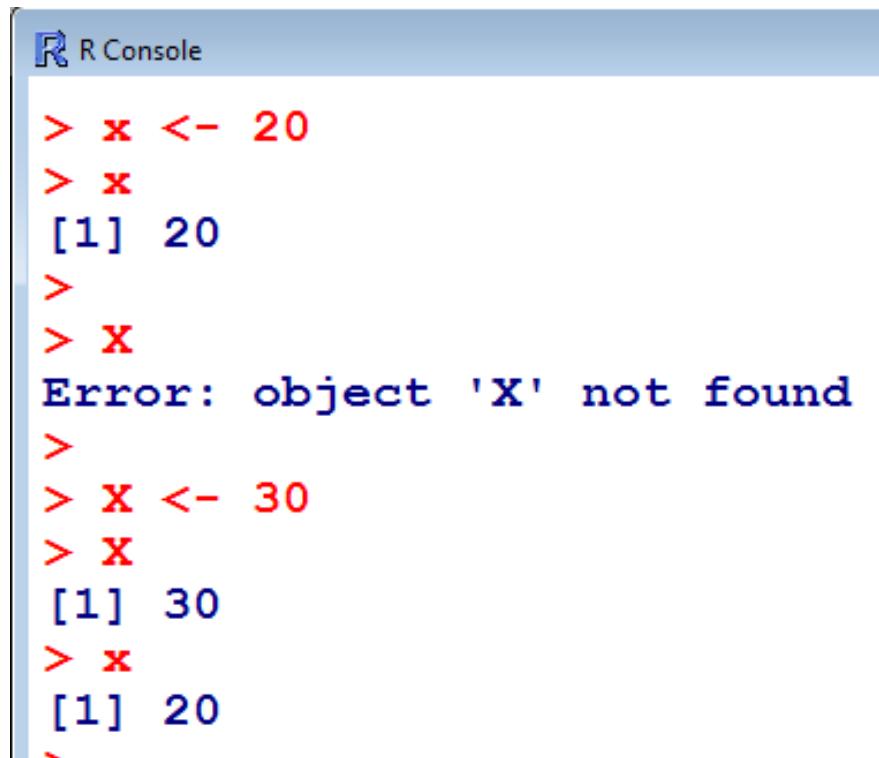
```
> x <- 20
> x
[1] 20
>
```

The text "x <- 20" and "x" are in red, indicating they are comments. The output "[1] 20" is in blue, indicating it is a regular R object.

# Case sensitivity in R

- Capital and small letters are different.

> **x** <- 20 and > **x** <- 20 are different

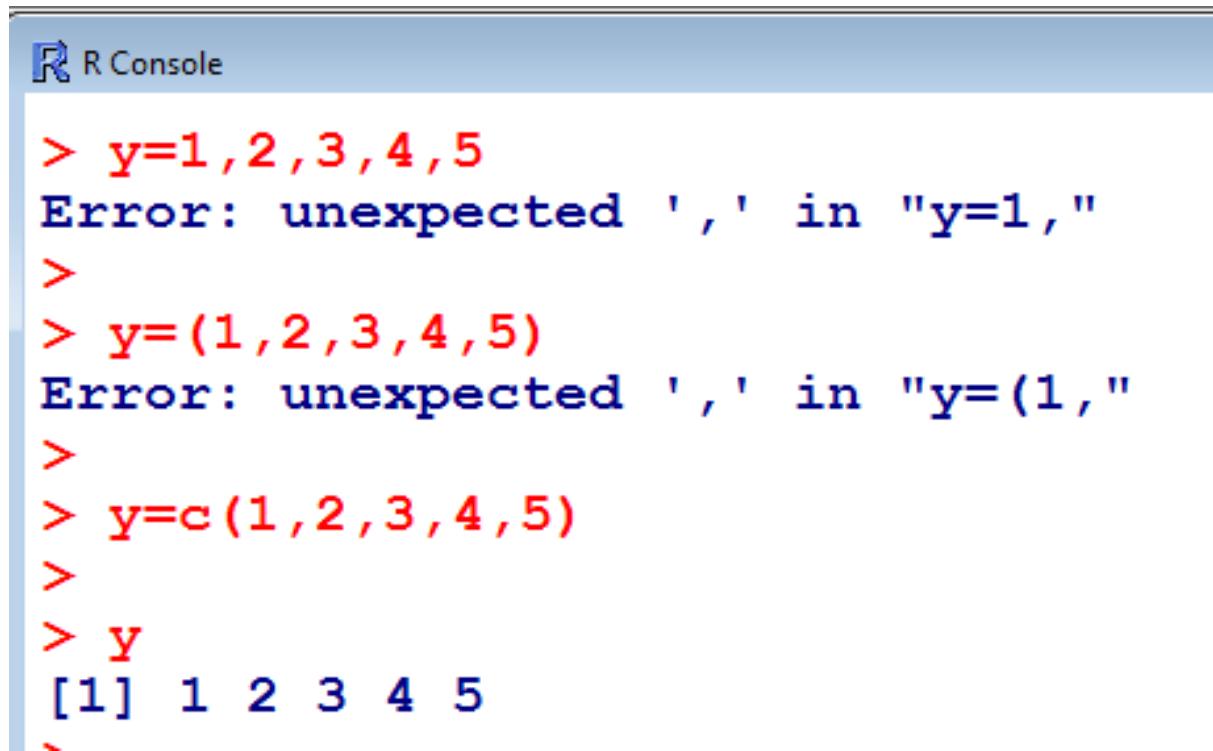


The screenshot shows an R console window titled "R Console". The session starts with the command "> **x** <- 20", followed by the output "[1] 20". Then, the command "> **x**" is entered, which results in the error message "Error: object 'X' not found". The session continues with the command "> **x** <- 30", followed by the output "[1] 30". Finally, the command "> **x**" is entered again, resulting in the output "[1] 20". This demonstrates that the variable "x" and "X" are treated as different objects in R.

```
> x <- 20
> x
[1] 20
>
> x
Error: object 'X' not found
>
> x <- 30
> x
[1] 30
> x
[1] 20
```

# Combining values in a data vector

- The command `c(1,2,3,4,5)` combines the numbers 1,2,3,4 and 5 to a vector.



R Console

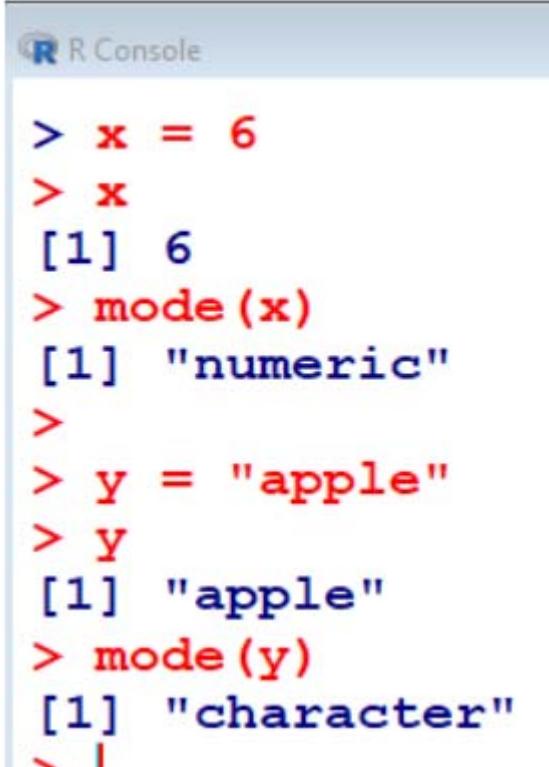
```
> y=1,2,3,4,5
Error: unexpected ',' in "y=1,"
>
> y=(1,2,3,4,5)
Error: unexpected ',' in "y=(1,"
>
> y=c(1,2,3,4,5)
>
> y
[1] 1 2 3 4 5
`
```

# Mode

Command `mode( )` explains the type or storage mode of an object.

[Be watchful: It's not like mean, median, mode]

```
> x = 6  
> x  
[1] 6  
> mode(x)  
[1] "numeric"  
  
> y = "apple"  
> y  
[1] "apple"  
> mode(y)  
[1] "character"
```



The screenshot shows the R console interface with a light blue header bar containing the R logo and the text "R Console". Below the header, the R command line is visible, showing the same sequence of commands and their results as the text above, demonstrating that the variable x is numeric and the variable y is a character string.

```
> x = 6  
> x  
[1] 6  
> mode(x)  
[1] "numeric"  
>  
> y = "apple"  
> y  
[1] "apple"  
> mode(y)  
[1] "character"  
|
```

# Mode

Following modes are available:

- "logical",
- "integer",
- "double",
- "complex",
- "raw",
- "character",
- "list",
- "expression",
- "name",
- "symbol" and
- "function".

# Mode

Command `storage.mode()` returns the storage mode of its argument.

It is generally used when calling functions written in another language, such as C or FORTRAN, to ensure that R objects have the data type expected by the routine being called.

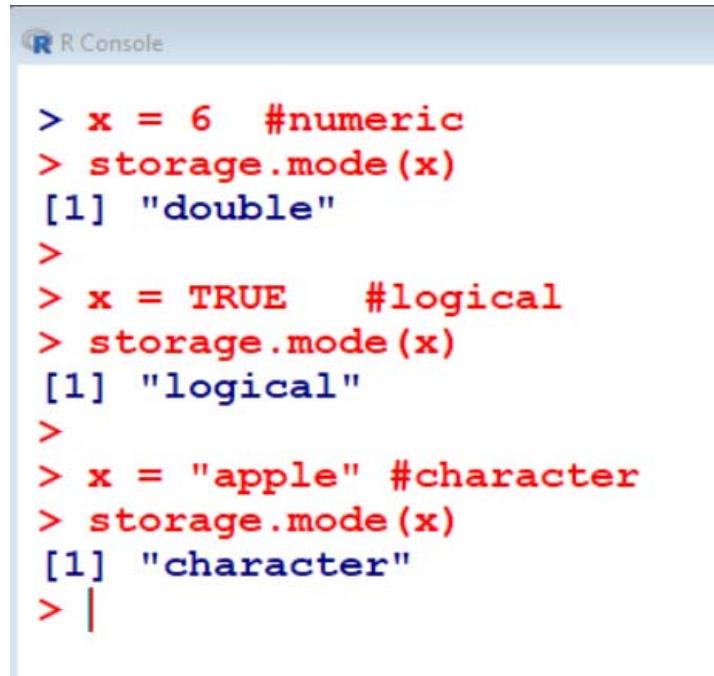
# Mode

<b>mode</b>	<b>storage.mode</b>
<b>logical</b>	<b>logical</b>
<b>numeric</b>	<b>integer</b>
<b>numeric</b>	<b>double</b>
<b>complex</b>	<b>complex</b>
<b>character</b>	<b>character</b>
<b>raw</b>	<b>raw</b>

# Mode

## Examples:

```
> x = 6    #numeric  
  
> storage.mode(x)  
[1] "double"  
  
  
> x = TRUE    #logical  
  
> storage.mode(x)  
[1] "logical"  
  
  
> x = "apple" #character  
  
> storage.mode(x)  
[1] "character"
```

A screenshot of an R console window titled "R Console". It shows three examples of determining the storage mode of variables. In each example, a numeric value or character string is assigned to 'x', followed by a comment indicating its type (e.g., "#numeric"). Then, 'storage.mode(x)' is run, and the output is "[1] <mode>".

```
R Console  
> x = 6    #numeric  
> storage.mode(x)  
[1] "double"  
>  
> x = TRUE    #logical  
> storage.mode(x)  
[1] "logical"  
>  
> x = "apple" #character  
> storage.mode(x)  
[1] "character"  
> |
```

# Infinity

Some calculations yield result as infinity ( $\infty$ ). For example,  $3/0$ . In R

`is.finite()` and `is.infinite()` are used to know if an outcome is finite or infinite, respectively.

```
> 3/0
[1] Inf

> 5+Inf
[1] Inf

> x = 5+Inf
> is.finite(x)
[1] FALSE

> is.infinite(x)
[1] TRUE
>
```



# **Foundations of R Software**

## **Lecture 8**

## **Basics of Calculations**

::::

**R as a Calculator with Scalars and Data Vectors :  
Addition, Subtraction, Multiplication & Division**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# R as a calculator

R can perform all standard calculations like addition, subtraction, multiplication, division etc.

R has built in functions for computations.

R can operator over scalars and data vectors.

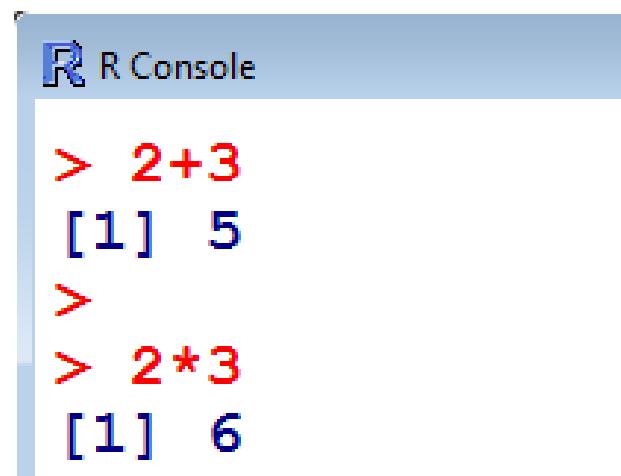
R computations can be carried out with

- Scalar versus scalar.
- Scalar versus data vectors.
- Data vectors versus data vectors.

# R as a calculator

```
> 2+3          # Command for addition  
[1] 5          # Output
```

```
> 2*3          # Command for multiplication  
[1] 6          # Output
```

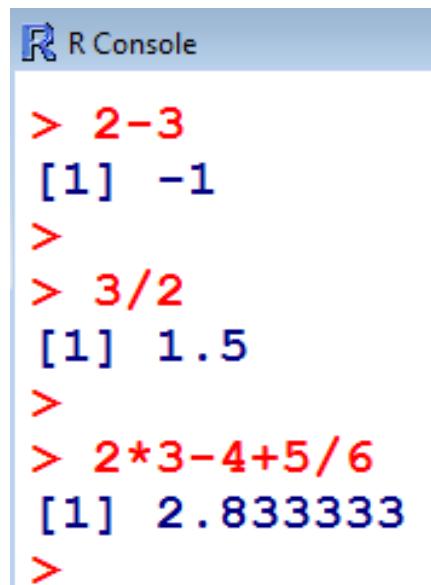


The screenshot shows the R Console interface. The title bar says "R R Console". The main area contains the following text:

```
> 2+3  
[1] 5  
>  
> 2*3  
[1] 6
```

# R as a calculator

```
> 2-3           # Command for subtraction  
[1] -1          # Output  
  
> 3/2           # Command for division  
[1] 1.5          # Output  
  
> 2*3-4+5/6    # Command  
[1] 2.8333      # Output
```



The screenshot shows the R Console window. The title bar says "R R Console". The console area displays the following R session:

```
> 2-3  
[1] -1  
>  
> 3/2  
[1] 1.5  
>  
> 2*3-4+5/6  
[1] 2.833333  
>
```

# R as a calculator

**BODMAS rule is applicable.**

**B-Brackets, O-Orders (powers), D-Division, M-Multiplication,  
A-Addition, S-Subtraction.**

**The mathematical expressions with multiple operators are solved from left to right in this order.**

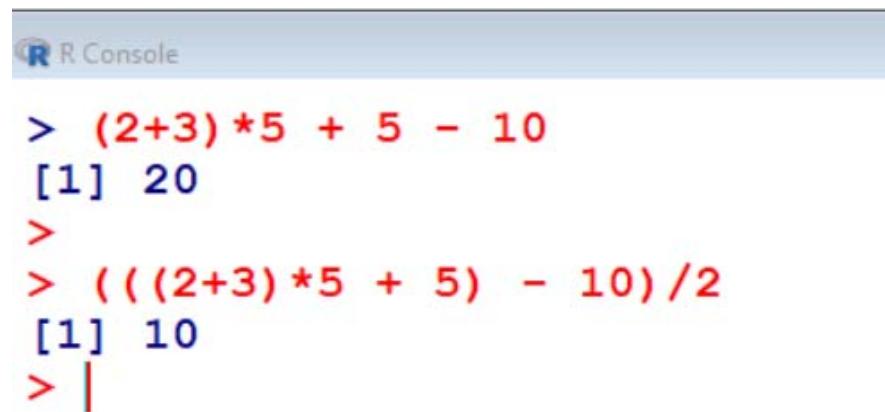
**Only ( ) brackets are used for BODMAS.**

**No brackets { } and [ ] are used in BODMAS.**

# R as a calculator

```
> (2+3)*5 + 5 - 10 # Command for BODMAS  
[1] 20 # Output
```

```
> (((2+3)*5 + 5) - 10)/2 # Command for BODMAS  
[1] 10 # Output
```

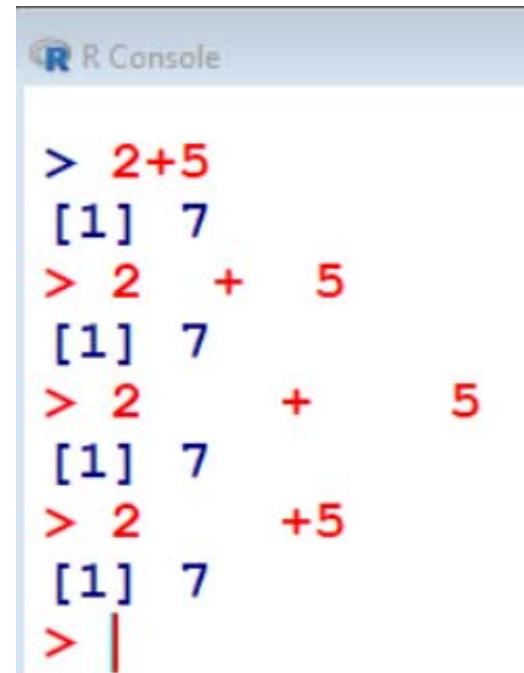


```
R Console  
> (2+3)*5 + 5 - 10  
[1] 20  
>  
> (((2+3)*5 + 5) - 10)/2  
[1] 10  
> |
```

# R as a calculator

Blank space has no role in calculations

```
> 2+5  
[1] 7  
  
> 2 + 5  
[1] 7  
  
> 2 + 5  
[1] 7  
  
> 2 +5  
[1] 7
```



The image shows a screenshot of the R Console window. It displays several R commands and their results. The commands are color-coded: the first command and its result are in blue, while subsequent attempts are in red. The console output is as follows:

```
R R Console  
> 2+5  
[1] 7  
> 2 + 5  
[1] 7  
> 2 + 5  
[1] 7  
> 2 +5  
[1] 7  
> |
```

# Addition with scalar

```
> c(2,3,5,7) + 10  
[1] 12 13 15 17
```

$2+10, 3+10, 5+10, 7+10$

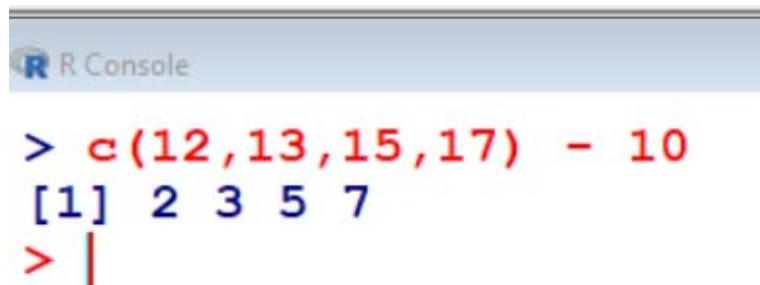
R R Console

```
> c(2,3,5,7) + 10  
[1] 12 13 15 17
```

# Subtraction with scalar

```
> c(12,13,15,17) - 10  
[1] 2 3 5 7
```

$$12-10, 13-10, 15-10, 17-10$$



```
R Console  
> c(12,13,15,17) - 10  
[1] 2 3 5 7  
> |
```

# Multiplication with scalar

```
> c(2,3,5,7) * 10  
[1] 20 30 50 70
```

$2 \times 10, 3 \times 10, 5 \times 10, 7 \times 10$

R Console

```
> c(2,3,5,7) * 10  
[1] 20 30 50 70  
> |
```

# Division with scalar

```
> c(12,13,15,17) / 10  
[1] 1.2 1.3 1.5 1.7
```

$$12 \div 10, \quad 13 \div 10, \quad 15 \div 10, \quad 17 \div 10$$



```
> c(12,13,15,17) / 10  
[1] 1.2 1.3 1.5 1.7  
> |
```

# **Foundations of R Software**

## **Lecture 9**

## **Basics of Calculations**

**::::**

## **Calculations with Data Vectors : Addition, Subtraction, Multiplication & Division**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Addition with data vectors

```
> c(2,3,5,7) + c(-2,-3, -5, 8)  
[1] 0 0 0 15
```

$$2 + (-2), \quad 3 + (-3), \quad 5 + (-5), \quad 7 + 8$$

# **Foundations of R Software**

## **Lecture 10**

## **Basics of Calculations**

::::

**R as a Calculator with Scalars and Data Vectors :  
Power operations, Integer and Modulo divisions**

Shalabh

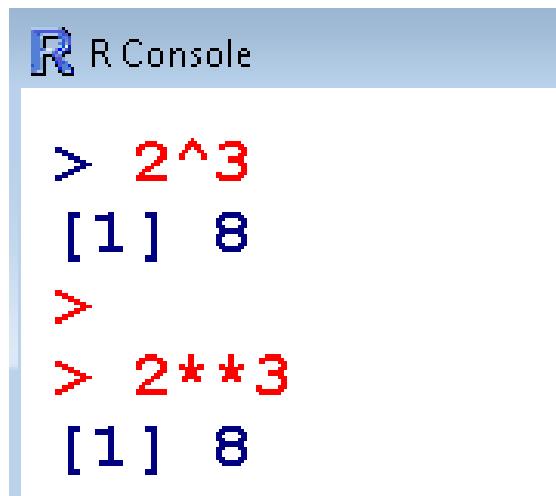
Department of Mathematics and Statistics

Indian Institute of Technology Kanpur

# R as a calculator

```
> 2^3          # Command for power operator  
[1] 8          # Output
```

```
> 2**3         # Command for power operator  
[1] 8          # Output
```

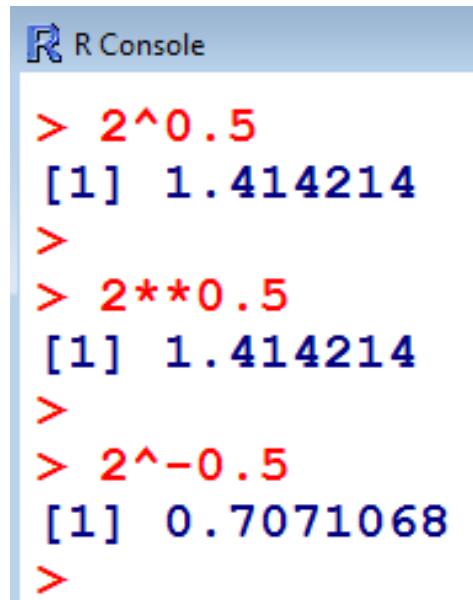


A screenshot of the R console window. The title bar says "R Console". The main area contains two sets of code and output. The first set shows the command `> 2^3` in red and its output [1] 8 in blue. The second set shows the command `> 2**3` in red and its output [1] 8 in blue. The console has a light blue background and a dark blue sidebar.

```
> 2^3  
[1] 8  
>  
> 2**3  
[1] 8
```

# R as a calculator

```
> 2^0.5          # Command for power operator  
[1] 1.414214   # Output  
  
> 2**0.5        # Command for power operator  
[1] 1.414214   # Output  
  
> 2^-0.5        # Command for power operator  
[1]  
0.7071068     # Output
```



The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R code and its corresponding output. The first command is `> 2^0.5`, followed by the output `[1] 1.414214`. A new line starts with `>`. The second command is `> 2**0.5`, followed by the output `[1] 1.414214`. Another new line starts with `>`. The third command is `> 2^-0.5`, followed by the output `[1] 0.7071068`. A final new line starts with `>`.

```
> 2^0.5  
[1] 1.414214  
>  
> 2**0.5  
[1] 1.414214  
>  
> 2^-0.5  
[1] 0.7071068  
>
```

# Power operator with scalar

```
> c(2,3,5,7)^2      # command: application to a  
                      vector  
[1] 4 9 25 49       # output
```

$$2^2, 3^2, 5^2, 7^2$$

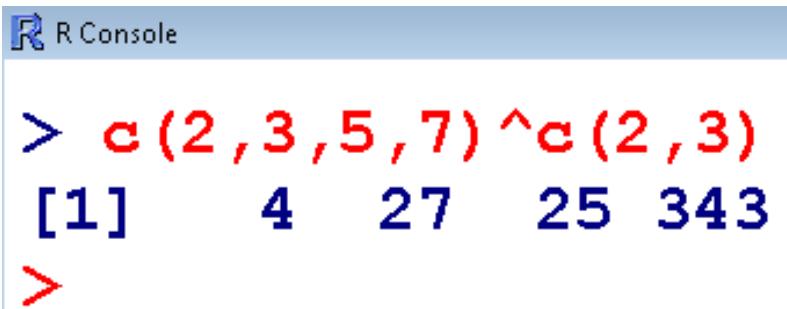
R Console

```
> c(2,3,5,7)^2  
[1] 4 9 25 49  
>
```

# Power operation with vector

```
> c(2,3,5,7)^c(2,3) # !!ATTENTION! Observe the  
# operation  
[1] 4 27 25 343 # output
```

$$2^2, 3^3, 5^2, 7^3$$



R Console

```
> c(2,3,5,7)^c(2,3)
[1] 4 27 25 343
>
```

# Power operation with vector

```
> c(1,2,3,4,5,6)^c(2,3,4) # command: application  
                           to a vector with vector  
[1]  1  8  81 16 125 1296  # output
```

$$1^2, 2^3, 3^4, 4^2, 5^3, 6^4$$

R Console

```
> c(1,2,3,4,5,6)^c(2,3,4)  
[1]  1    8   81  16  125 1296  
>
```

# Power operation with vector

```
> c(2,3,5,7)^c(2,3,4)      #error message  
[1]  4 27 625 49           # output
```

Warning message:

longer object length is not a multiple of  
shorter object length in: c(2,3,5,7)^c(2,3,4)

$$2^2, 3^3, 5^4, 7^2$$



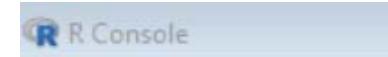
```
R Console  
> c(2,3,5,7)^c(2,3,4)  
[1]  4 27 625 49
```

Warning message:

In c(2, 3, 5, 7)^c(2, 3, 4) :  
 longer object length is not a multiple of shorter object length  
>

# Integer division with scalar

**Integer Division:** Division in which the fractional part (remainder) is discarded



**Operator :** `%/%`

```
> 2 %/% 2  
[1] 1
```

```
> 5 %/% 2  
[1] 2
```

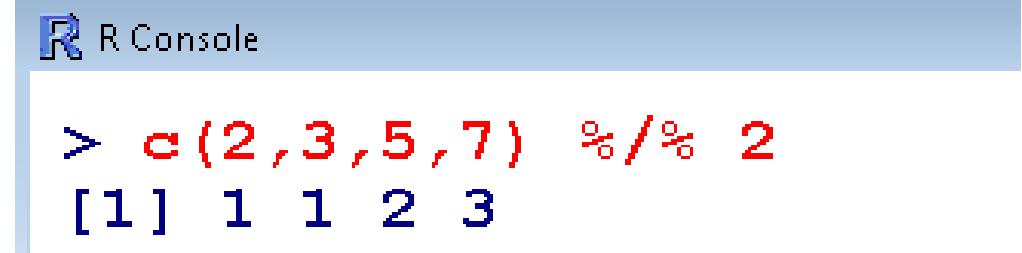
```
> 7 %/% 3  
[1] 2
```

```
> 2 %/% 2  
[1] 1  
> 5 %/% 2  
[1] 2  
> 7 %/% 3  
[1] 2  
> |
```

## Integer division with scalar

```
> c(2,3,5,7) %/% 2  
[1] 1 1 2 3
```

$2 \% / \% 2, \ 3 \% / \% 2, \ 5 \% / \% 2, \ 7 \% / \% 2$

A screenshot of an R console window titled "R Console". The title bar is blue with white text. The main area of the window shows the R code and its output.

```
> c(2,3,5,7) %/% 2  
[1] 1 1 2 3
```

## Integer division with data vector

```
> c(2,3,5,7) %/% c(2,3)  
[1] 1 1 2 2
```

2%/%2, 3%/%3, 5%/%2, 7%/%3

R R Console

```
> c(2,3,5,7) %/% c(2,3)  
[1] 1 1 2 2
```

# Integer division with data vector

```
> c(2,3,5) %/% c(2,3)
```

```
[1] 1 1 2
```

Warning message:

In c(2, 3, 5)%/%c(2, 3) :

longer object length is not a multiple of  
shorter object length

```
2%/%2, 3%/%3, 5%/%2
```

R Console

```
> c(2,3,5) %/% c(2,3)
```

```
[1] 1 1 2
```

Warning message:

In c(2, 3, 5)%/%c(2, 3) :

longer object length is not a multiple of shorter object length

```
> |
```

# Modulo Division (x mod y) with scalars

**Modulo Division:** modulo operation finds the remainder after division of one number by another.

Operator : `%%`

```
> 2 %% 2
```

```
[1] 0
```

```
> 3 %% 2
```

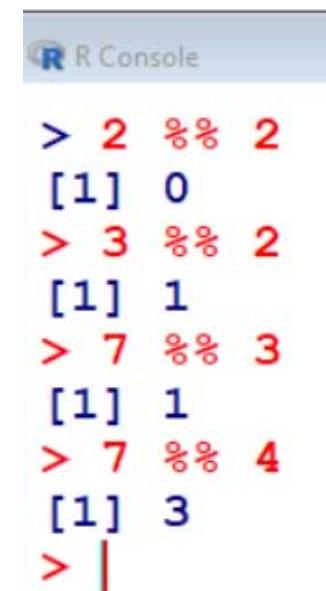
```
[1] 1
```

```
> 7 %% 3
```

```
[1] 1
```

```
> 7 %% 4
```

```
[1] 3
```



A screenshot of an R console window titled "R Console". The window shows several lines of R code and their corresponding outputs. The code consists of modulo operations using the operator `%%`. The first four lines show `2 %% 2`, `3 %% 2`, `7 %% 3`, and `7 %% 4` with results 0, 1, 1, and 3 respectively. The last line is a blank line starting with a greater than sign.

```
> 2 %% 2
[1] 0
> 3 %% 2
[1] 1
> 7 %% 3
[1] 1
> 7 %% 4
[1] 3
> |
```

## Modulo Division (x mod y) with scalars

```
> c(2,3,5,7) %% 2  
[1] 0 1 1 1
```

`2%%2, 3%%2, 5%%2, 7%%2`

R R Console

```
> c(2,3,5,7) %% 2  
[1] 0 1 1 1
```

## Modulo Division ( $x \bmod y$ ) with data vectors

```
> c(2,3,5,7) %% c(2,3)
[1] 0 0 1 1
```

```
2%%2, 3%%3, 5%%2, 7%%3
```



```
> c(2,3,5,7) %% c(2,3)
[1] 0 0 1 1
```

# Modulo Division ( $x \bmod y$ ) with data vectors

```
> c(2,3,5) %% c(2,3)
[1] 0 0 1
```

Warning message:

```
In c(2, 3, 5)%%c(2, 3) :
  longer object length is not a multiple of
  shorter object length
```

2%/%2, 3%/%3, 5%/%2

R R Console

```
> c(2,3,5) %% c(2,3)
[1] 0 0 1
Warning message:
In c(2, 3, 5)%%c(2, 3) :
  longer object length is not a multiple of shorter object length
> |
```

# **Foundations of R Software**

## **Lecture 11**

## **Basics of Calculations**

::::

## **Built in Functions and Assignments**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# **R as a calculator**

**R can perform all standard calculations like addition, subtraction, multiplication, division etc.**

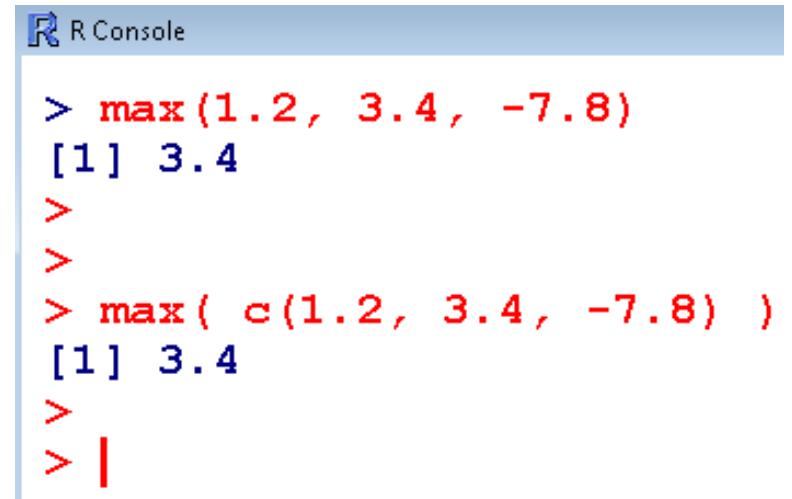
**R has built in functions for computations.**

# Maximum

Operator: **max**

```
> max(1.2, 3.4, -7.8)
[1] 3.4
```

```
> max( c(1.2, 3.4, -7.8) )
[1] 3.4
```



R Console

```
> max(1.2, 3.4, -7.8)
[1] 3.4
>
>
> max( c(1.2, 3.4, -7.8) )
[1] 3.4
>
> |
```

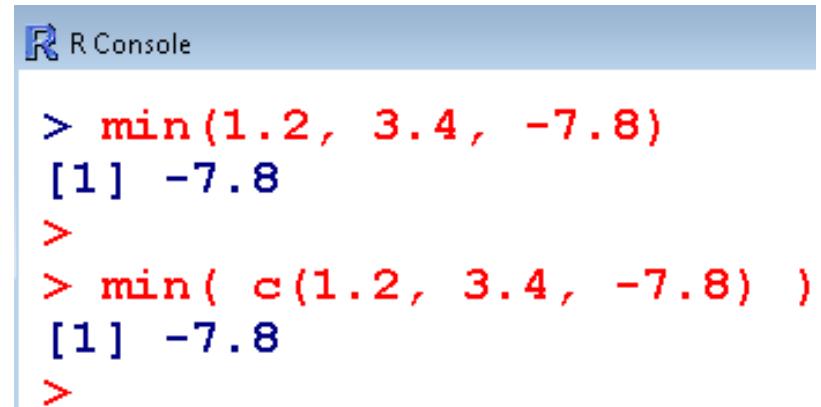
See the difference in use of **c** command.

# Minimum

Operator: `min`

```
> min(1.2, 3.4, -7.8)
[1] -7.8
```

```
> min( c(1.2, 3.4, -7.8) )
[1] -7.8
```



The screenshot shows the R console interface with a blue header bar containing the R logo and the text "R Console". Below the header, there are two examples of the `min` function being used.

```
R Console
> min(1.2, 3.4, -7.8)
[1] -7.8
>
> min( c(1.2, 3.4, -7.8) )
[1] -7.8
>
```

See the difference in use of `c` command.

# Arithmetic mean

Operator: `mean`

```
> mean(2, 3, 4)
```

$$\text{mean} = \frac{2+3+4}{3} = 3$$

```
[1] 2
```

```
> mean(c(2, 3, 4))  
[1] 3
```

```
R Console  
> mean(2, 3, 4)  
[1] 2  
> mean(c(2, 3, 4))  
[1] 3  
> |
```

See the difference in use of `c` command.

Suggestion: Always use `c` command to input more than one data value.

# Overview Over Further Functions

<code>abs()</code>	Absolute value
<code>sqrt()</code>	Square root
<code>round()</code> , <code>floor()</code> , <code>ceiling()</code>	Rounding, up and down
<code>sum()</code> , <code>prod()</code>	Sum and product
<code>log()</code> , <code>log10()</code> , <code>log2()</code>	Logarithms
<code>exp()</code>	Exponential function
<code>sin()</code> , <code>cos()</code> , <code>tan()</code> , <code>asin()</code> , <code>acos()</code> , <code>atan()</code>	Trigonometric functions
<code>sinh()</code> , <code>cosh()</code> , <code>tanh()</code> , <code>asinh()</code> , <code>acosh()</code> , <code>atanh()</code>	Hyperbolic functions

# Examples

Operator: **abs** for finding the absolute values

```
> abs(-4)
```

```
[1] 4
```

```
> abs(c(-1,-2,-3,4,5))
```

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

R R Console

```
> abs(-4)
```

```
[1] 4
```

```
>
```

```
> abs(c(-1,-2,-3,4,5))
```

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

```
>
```

# Examples

Operator: `sqrt` for finding the square root of values

```
> sqrt(4)
```

```
[1] 2
```

```
> sqrt(c(4,9,16,25))
```

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

R Console

```
> sqrt(4)
```

```
[1] 2
```

```
>
```

```
> sqrt(c(4,9,16,25))
```

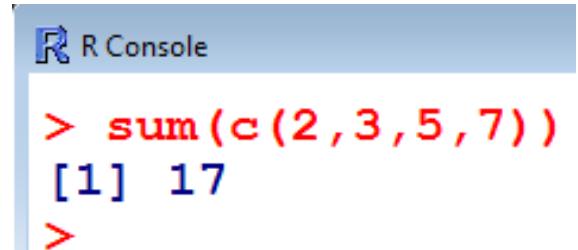
```
[1] 2 3 4 5
```

```
>
```

# Examples

Operator: **sum** for finding the sum of values

```
> sum(c(2,3,5,7))  
[1] 17
```

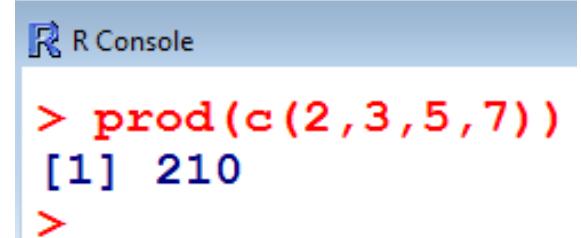


R Console

```
> sum(c(2,3,5,7))  
[1] 17  
>
```

Operator: **prod** for finding the product of values

```
> prod(c(2,3,5,7))  
[1] 210
```



R Console

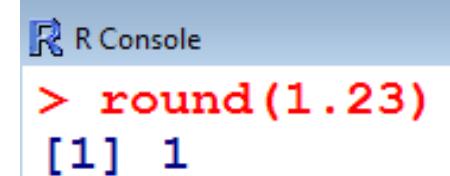
```
> prod(c(2,3,5,7))  
[1] 210  
>
```

# Examples

Operator: **round** for finding the round off values

```
> round(1.23)
```

```
[1] 1
```

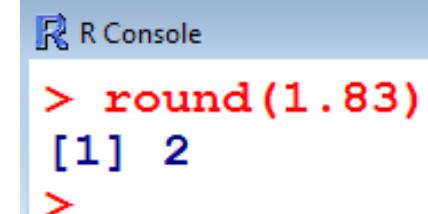


R Console  
> round(1.23)  
[1] 1

A screenshot of an R console window. The title bar says "R Console". The main area contains the command "round(1.23)" in red and its output "[1] 1" in blue.

```
> round(1.83)
```

```
[1] 2
```



R Console  
> round(1.83)  
[1] 2  
>

A screenshot of an R console window. The title bar says "R Console". The main area contains the command "round(1.83)" in red and its output "[1] 2" in blue. A new line indicator ">" is visible at the bottom.

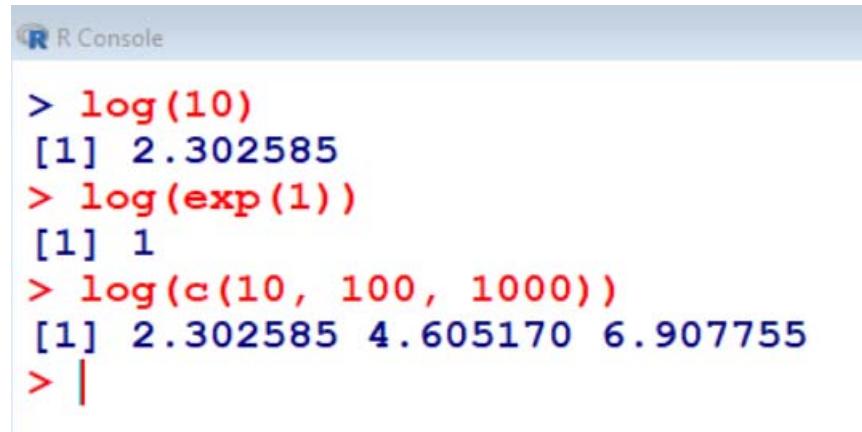
# Examples

Operator: `log` for finding the natural log (`ln`) of values

```
> log(10)
[1] 2.302585
```

```
> log(exp(1))
[1] 1
```

```
> log(c(10, 100, 1000))
[1] 2.302585 4.605170 6.907755
```



A screenshot of the R Console window. The title bar says "R Console". The console area contains the following R code and output:

```
> log(10)
[1] 2.302585
> log(exp(1))
[1] 1
> log(c(10, 100, 1000))
[1] 2.302585 4.605170 6.907755
> |
```

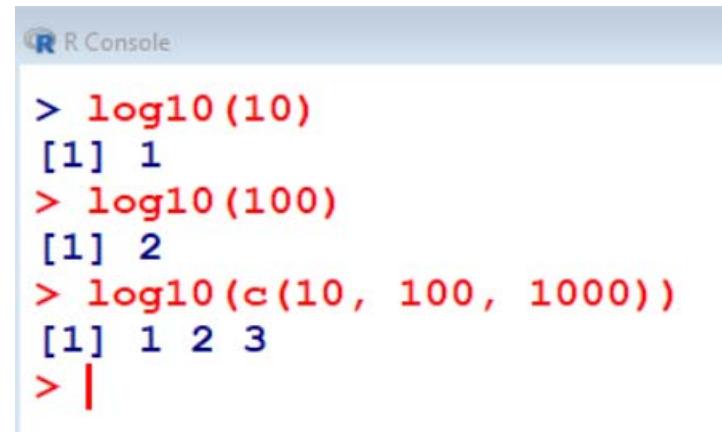
# Examples

Operator: `log10` for finding the log (with base 10) of values

```
> log10(10)
[1] 1
```

```
> log10(100)
[1] 2
```

```
> log10(c(10, 100, 1000))
[1] 1 2 3
```



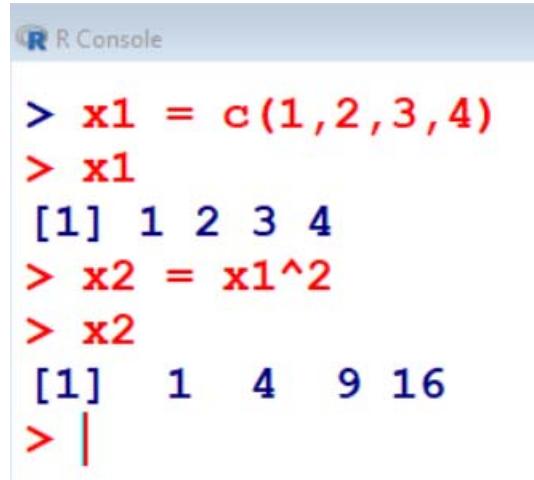
A screenshot of an R console window titled "R Console". It shows the following R code and its output:

```
R Console
> log10(10)
[1] 1
> log10(100)
[1] 2
> log10(c(10, 100, 1000))
[1] 1 2 3
> |
```

# Assignments

An assignment can also be used to save values in variables:

```
> x1 = c(1,2,3,4)
> x1
[1] 1 2 3 4
> x2 = x1^2
> x2
[1] 1 4 9 16
```



A screenshot of an R console window titled "R Console". The window shows the following R session:

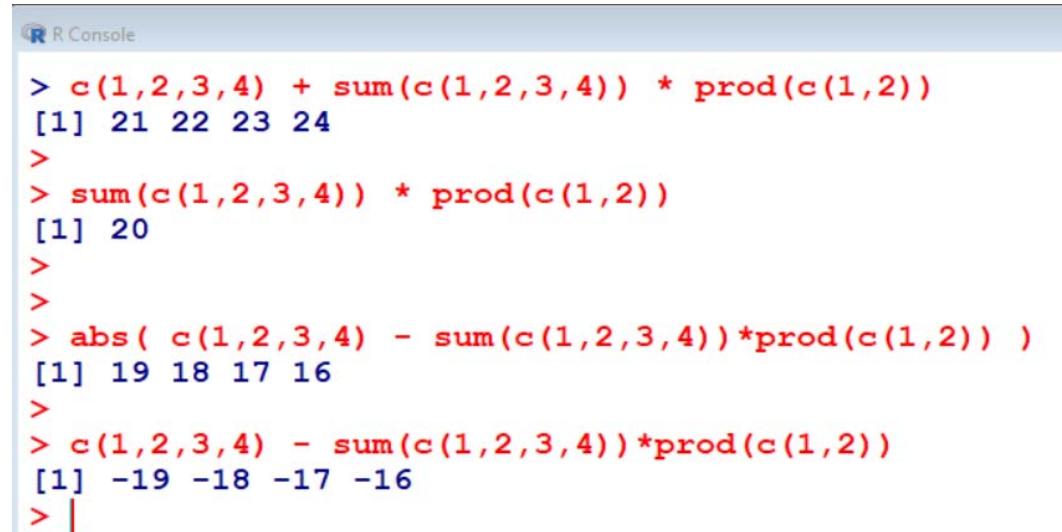
```
> x1 = c(1,2,3,4)
> x1
[1] 1 2 3 4
> x2 = x1^2
> x2
[1] 1 4 9 16
> |
```

# Assignments

Calculations with scalar, data vector and built in functions also work.

```
> c(1,2,3,4) + sum(c(1,2,3,4)) * prod(c(1,2))
[1] 21 22 23 24
```

```
> abs( c(1,2,3,4) - sum(c(1,2,3,4))*prod(c(1,2)) )
[1] 19 18 17 16
```



The screenshot shows the R Console interface with a light blue header bar containing the R logo and the word "Console". Below the header, there is a scrollable text area displaying R code and its output. The code consists of several lines of R commands, each starting with a red ">" prompt followed by the command in black text. The output is shown in blue text below each command. The first two lines are identical to the ones above. The third line starts with a red ">" prompt, followed by the command, and the output [1] 20 is in blue. The fourth line starts with a red ">" prompt, followed by the command, and the output [1] 19 18 17 16 is in blue. The fifth line starts with a red ">" prompt, followed by the command, and the output [1] -19 -18 -17 -16 is in blue. The sixth line starts with a red ">" prompt, followed by a vertical line | indicating the end of the session.

```
> c(1,2,3,4) + sum(c(1,2,3,4)) * prod(c(1,2))
[1] 21 22 23 24
>
> sum(c(1,2,3,4)) * prod(c(1,2))
[1] 20
>
>
> abs( c(1,2,3,4) - sum(c(1,2,3,4))*prod(c(1,2)) )
[1] 19 18 17 16
>
> c(1,2,3,4) - sum(c(1,2,3,4))*prod(c(1,2))
[1] -19 -18 -17 -16
> |
```

# **Foundations of R Software**

## **Lecture 12**

## **Basics of Calculations**

::::

## **Matrices**

Shalabh

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Matrix

Matrices are important objects in any calculation.

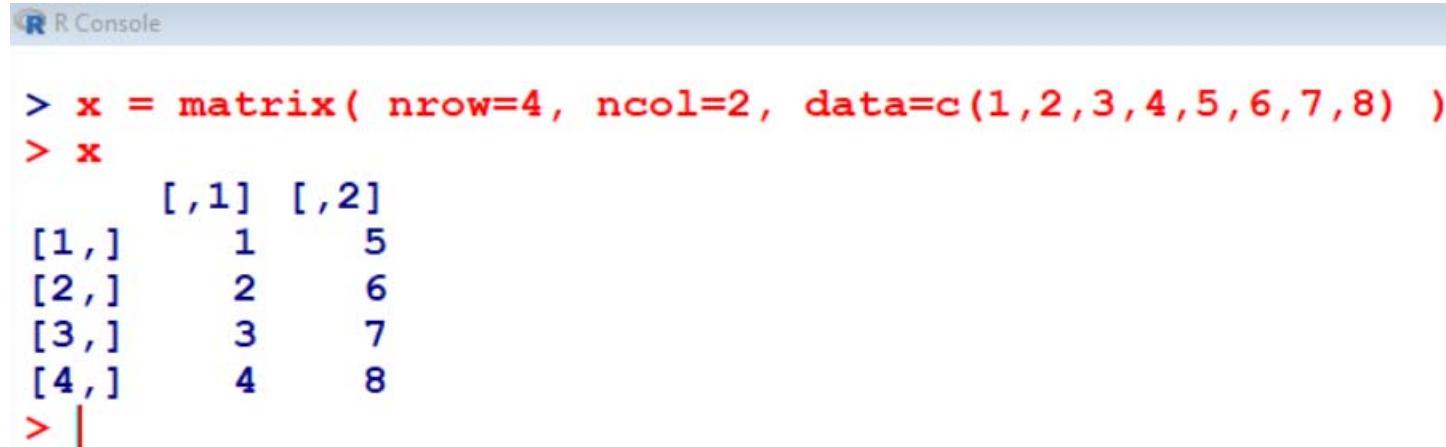
A matrix is a rectangular array with  $p$  rows and  $n$  columns.

An element in the  $i$ -th row and  $j$ -th column is denoted by  $X_{ij}$  (book version) or  $X[i, j]$  ("program version"),  $i = 1, 2, \dots, n, j = 1, 2, \dots, p$ .

An element of a matrix can also be an object, for example a string.  
However, in mathematics, we are mostly interested in numerical matrices, whose elements are generally real numbers

In R, a  $4 \times 2$ -matrix  $X$  can be created with a following command:

```
> x = matrix( nrow=4, ncol=2,
               data=c(1,2,3,4,5,6,7,8) )
> x
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```



R Console

```
> x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8) )
> x
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> |
```

# Matrix parameters

We see:

The parameter `nrow` defines the row number of a matrix.

The parameter `ncol` defines the column number of a matrix.

The parameter `data` assigns specified values to the matrix elements.

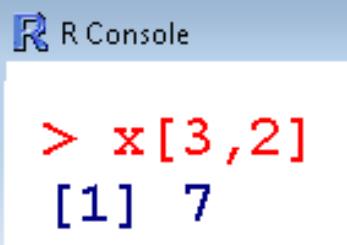
The data values from the parameters are written column-wise in matrix.

# Accessing any element of a matrix

```
> x
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

One can access a single element of a matrix with `x[i, j]`:

```
> x[3,2]
[1] 7
```



R Console

```
> x[3,2]
[1] 7
```

# Entering data column wise in a matrix

In case, the data has to be entered column wise, then a  $4 \times 2$ -matrix  $X$  can be created with

```
> x = matrix( nrow=4, ncol=2,
               data=c(1,2,3,4,5,6,7,8))
```

OR

```
x = matrix( nrow=4, ncol=2,
               data=c(1,2,3,4,5,6,7,8), byrow = FALSE)
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

# Entering data row wise in a matrix

In case, the data has to be entered row wise, then a  $4 \times 2$ -matrix  $X$  can be created with

```
> y = matrix( nrow=4, ncol=2,
              data=c(1,2,3,4,5,6,7,8), byrow = TRUE)
> y
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```

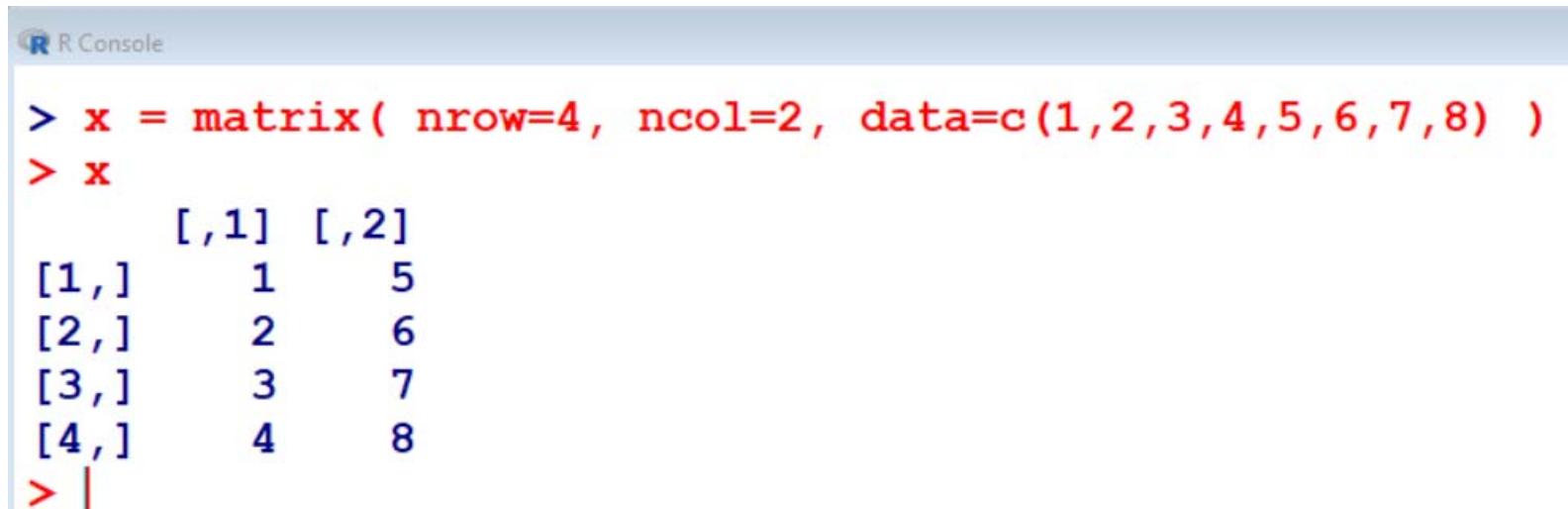
# Entering data row wise and column wise in a matrix

```
R Console

> x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8) )
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> y = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8), byrow = TRUE )
> y
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> |
```

In R, a  $4 \times 2$ -matrix  $X$  can be created with a following command:

```
> x = matrix( nrow=4, ncol=2,
               data=c(1,2,3,4,5,6,7,8) )
> x
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```



The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R code and its output. The code is identical to the one shown above, creating a matrix 'x' with 4 rows and 2 columns from the vector 'c(1,2,3,4,5,6,7,8)'. The output shows the matrix structure with four rows and two columns of values.

```
> x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8) )
> x
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> |
```

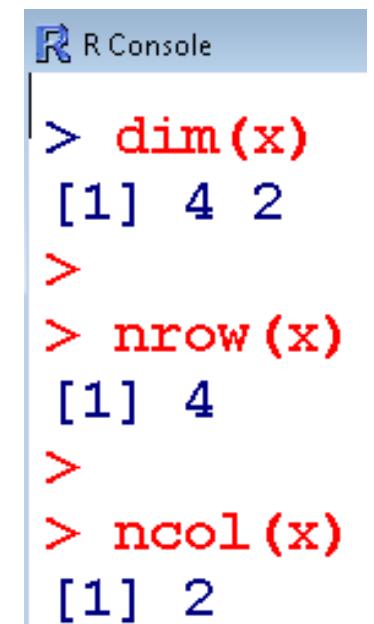
# Properties of a Matrix

We can get specific *properties* of a matrix:

```
> dim(x) # tells the dimension of matrix  
[1] 4 2
```

```
> nrow(x) # tells the number of rows  
[1] 4
```

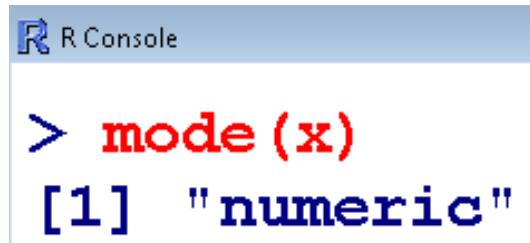
```
> ncol(x) # tells the number of columns  
[1] 2
```



The image shows a screenshot of an R console window titled "R Console". The window contains the following text:  
> dim(x)  
[1] 4 2  
>  
> nrow(x)  
[1] 4  
>  
> ncol(x)  
[1] 2

# Properties of a Matrix

```
> mode(x)      # Informs the type or storage  
mode of an object, e.g.,  
numerical, logical etc.  
[1] "numeric"
```

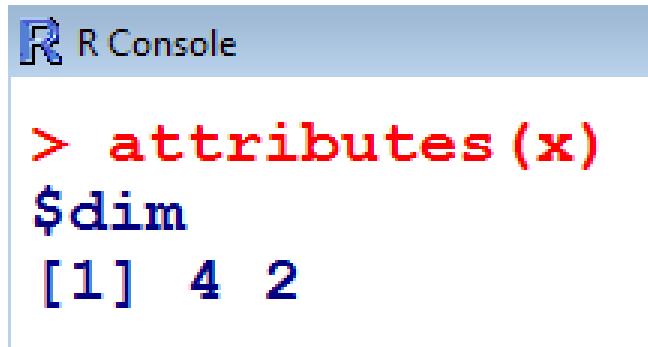


R Console

```
> mode(x)  
[1] "numeric"
```

**attributes** provides all the attributes of an object

```
> attributes(x) #Informs the dimension of matrix  
$dim [1] 4 2
```



R Console

```
> attributes(x)  
$dim  
[1] 4 2
```

## Help on the Object "Matrix"

To know more about these important objects, we use R-help on "matrix".

```
> help("matrix")
matrix      package:base      R Documentation
```

### Matrices

#### Description:

'**matrix**' creates a matrix from the given set of values.

'**as.matrix**' attempts to turn its argument into a matrix.

'**is.matrix**' tests if its argument is a (strict) matrix. It is generic: you can write methods to handle specific classes of objects, see Internal Methods.

Then we get an overview on how a matrix can be created and what parameters are available:

**Usage:**

```
matrix(data [= NA, nrow = 1, ncol = 1, byrow = FALSE,  
           dimnames = NULL)  
as.matrix(x)  
is.matrix(x)
```

**Arguments:**

**data:** an optional data vector.

**nrow:** the desired number of rows

**ncol:** the desired number of columns

**byrow:** logical. If '**FALSE**'(the default) the matrix is filled by columns, otherwise the matrix is filled by rows.

**dimnames:** A '**dimnames**' attribute for the matrix: a '**list**' of length 2.

**x:** an R object.

Then, the meaning of each parameter is explained:

**Details:**

If either of '`nrow`' or '`ncol`' is not given, an attempt is made to infer it from the length of '`data`' and the other parameter.

If there are too few elements in '`data`' to fill the array, then the elements in '`data`' are recycled. If '`data`' has length zero, '`NA`' of an appropriate type is used for atomic vectors and '`NULL`' for lists.

'`is.matrix`' returns '`TRUE`' if '`x`' is a matrix (i.e., it is not a '`data.frame`' and has a '`dim`' attribute of length 2) and '`FALSE`' otherwise.

'`as.matrix`' is a generic function. The method for data frames will convert any non-numeric/complex column into a character vector using '`format`' and so return a character matrix, except that all-logical data frames will be coerced to a logical matrix.

**Finally, references and cross-references are displayed...**

**References:**

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

**See Also:**

**'data.matrix'**, which attempts to convert to a numeric matrix.

.. as well as an example:

**Examples:**

```
is.matrix(as.matrix(1:10))
data(warpbreaks)
!is.matrix(warpbreaks) # data.frame, NOT matrix!
warpbreaks[1:10,]
as.matrix(warpbreaks[1:10,]) #using
as.matrix.data.frame(.) method
```

# **Foundations of R Software**

## **Lecture 13**

## **Basics of Calculations**

::::

## **Matrix Operations – Row, Column & Other Operations**

Shalabh

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Matrix Operations

## Renaming the row and column names

```
> x = matrix( nrow=4, ncol=3, data=c(1:12) )  
> x  
     [,1] [,2] [,3]  
[1,]    1    5    9  
[2,]    2    6   10  
[3,]    3    7   11  
[4,]    4    8   12
```

**rownames(x)** Renames the row names

**colnames(x)** Renames the column names

## Matrix Operations

### Renaming the row and column names

```
> rownames(x) = c("r1", "r2", "r3", "r4")
```

```
> x
```

	[,1]	[,2]	[,3]
r1	1	5	9
r2	2	6	10
r3	3	7	11
r4	4	8	12

```
> colnames(x) = c("c1", "c2", "c3")
```

```
> x
```

	c1	c2	c3
r1	1	5	9
r2	2	6	10
r3	3	7	11
r4	4	8	12

# Matrix Operations

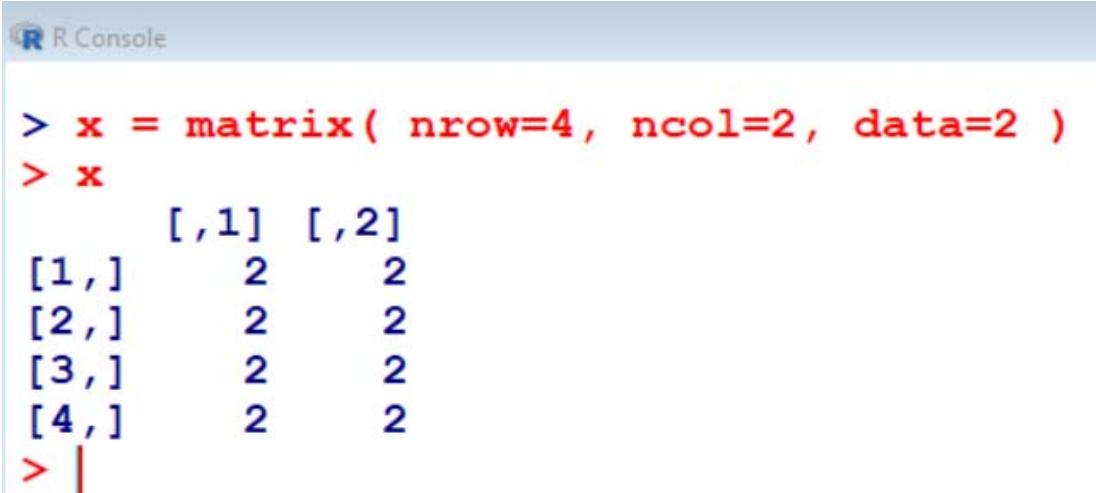
```
R Console

> x = matrix( nrow=4, ncol=3, data=c(1:12) )
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> rownames(x) = c("r1", "r2", "r3", "r4")
> x
     [,1] [,2] [,3]
r1    1    5    9
r2    2    6   10
r3    3    7   11
r4    4    8   12
> colnames(x) = c("c1", "c2", "c3")
> x
   c1 c2 c3
r1  1  5  9
r2  2  6 10
r3  3  7 11
r4  4  8 12
> |
```

# Matrix Operations

Assigning a specified number to all matrix elements:

```
> x = matrix( nrow=4, ncol=2, data=2 )  
> x  
      [,1] [,2]  
[1,]    2    2  
[2,]    2    2  
[3,]    2    2  
[4,]    2    2
```



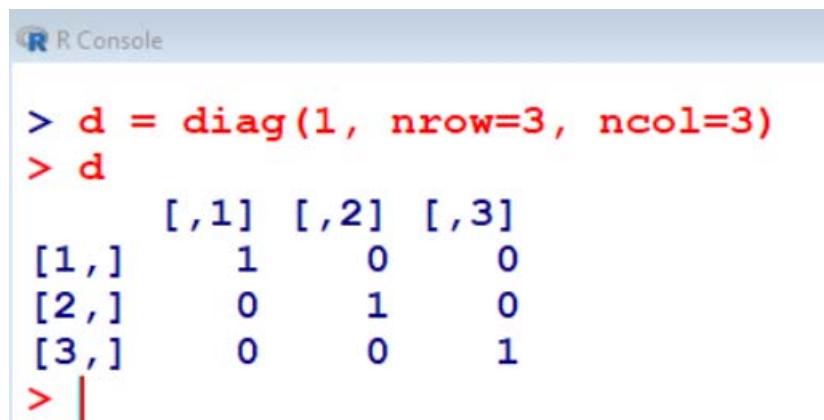
The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R code and its output. The code is identical to the one shown above, creating a 4x2 matrix 'x' with all elements set to 2. The output shows the matrix structure with four rows and two columns, each containing the value 2.

```
> x = matrix( nrow=4, ncol=2, data=2 )  
> x  
      [,1] [,2]  
[1,]    2    2  
[2,]    2    2  
[3,]    2    2  
[4,]    2    2  
> |
```

## Matrix Operations: Diagonal matrix

Construction of a diagonal matrix, here the identity matrix of a dimension 3:

```
> d = diag(1, nrow=3, ncol=3)
> d
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```



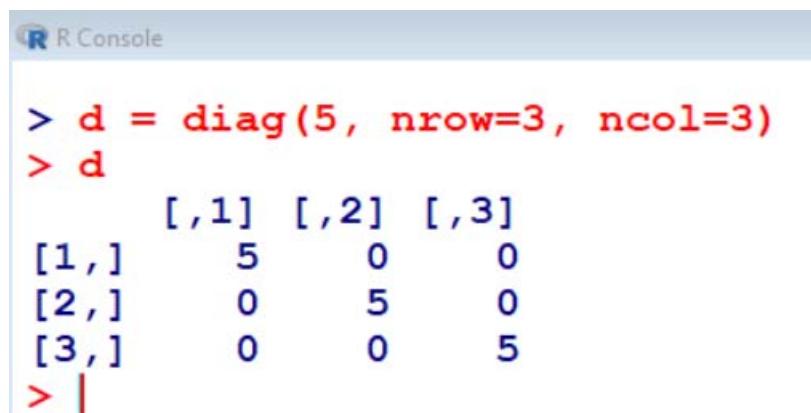
The screenshot shows the R console interface. The title bar says "R Console". The main area contains the following R code and its output:

```
> d = diag(1, nrow=3, ncol=3)
> d
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> |
```

## Matrix Operations: Diagonal matrix

Construction of a diagonal matrix with all numbers as 5 of a dimension 3:

```
> d = diag(5, nrow=3, ncol=3)
> d
      [,1] [,2] [,3]
[1,]     5     0     0
[2,]     0     5     0
[3,]     0     0     5
```



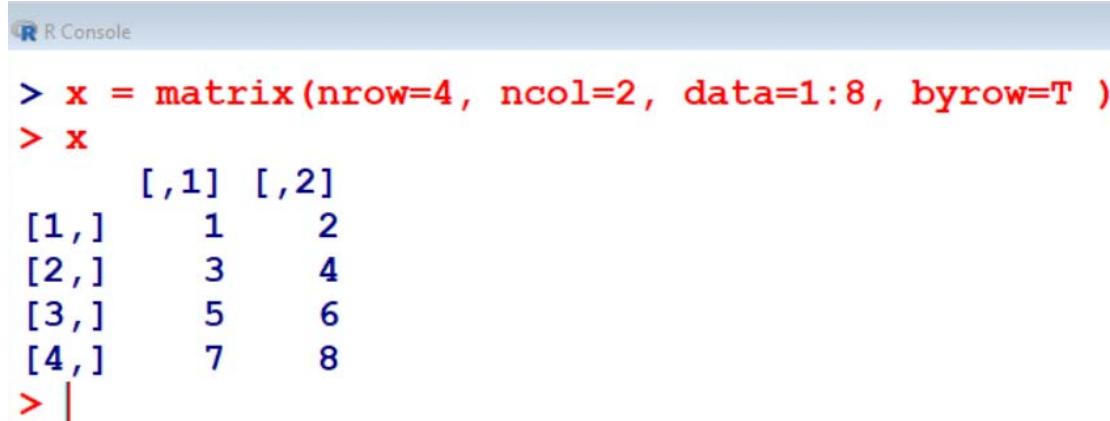
The screenshot shows the R Console window. The title bar says "R Console". The code entered is identical to the one above, resulting in the same 3x3 diagonal matrix output.

```
> d = diag(5, nrow=3, ncol=3)
> d
      [,1] [,2] [,3]
[1,]     5     0     0
[2,]     0     5     0
[3,]     0     0     5
> |
```

# Matrix Operations: Transpose

Transpose of a matrix X:  $X'$

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T )  
> x  
      [,1]    [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8
```



R Console

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T )  
> x  
      [,1]    [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
> |
```

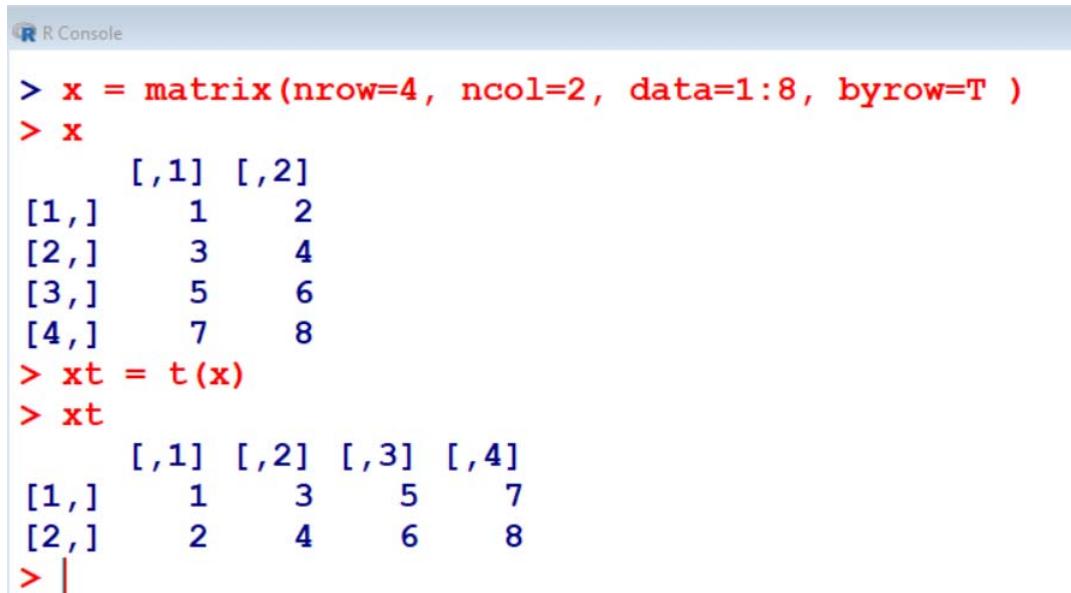
# Matrix Operations: Transpose

Transpose of a matrix X:  $X'$

```
> xt = t(x)
```

```
> xt
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	3	5	7
[2,]	2	4	6	8



R Console

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T )
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> xt = t(x)
> xt
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> |
```

## Matrix Operations: Row and column sums

Finding the row and column sums

```
> x = matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))  
> x  
     [,1] [,2]  
[1,]    1    5  
[2,]    2    6  
[3,]    3    7  
[4,]    4    8
```

**rowSums(x)** Finds the sum of numbers in rows

**colSums(x)** Finds the sum of numbers in columns

## Matrix Operations: Row and column means

```
> x
```

	[ ,1 ]	[ ,2 ]
[1, ]	1	5 → $(1+5) = 6$
[2, ]	2	6 → $(2+6) = 8$
[3, ]	3	7 → $(3+7) = 10$
[4, ]	4	8 → $(4+8) = 12$
	$(1+2+3+4)$ = 10	$(5+6+7+8)$ = 26

```
> rowSums(x)
```

```
[1] 6 8 10 12
```

```
> colSums(x)
```

```
[1] 10 26
```

```
R Console
> x = matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))
> x
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> rowSums(x)
[1] 6 8 10 12
> colSums(x)
[1] 10 26
>
```

## Matrix Operations: Row and column means

Finding the row and column means

```
> x = matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))  
> x  
     [,1] [,2]  
[1,]    1    5  
[2,]    2    6  
[3,]    3    7  
[4,]    4    8
```

**rowMeans(x)** Finds the means of rows

**colMeans(x)** Finds the means of columns

# Matrix Operations: Row and column means

```
> x
```

	[ ,1 ]	[ ,2 ]
[1, ]	1	5 → $(1+5)/2 = 3$
[2, ]	2	6 → $(2+6)/2 = 4$
[3, ]	3	7 → $(3+7)/2 = 5$
[4, ]	4	8 → $(4+8)/2 = 6$
	$(1+2+3+4)/4$ = 2.5	$(5+6+7+8)/4$ = 6.5

```
> rowMeans(x)
```

```
[1] 3 4 5 6
```

```
> colMeans(x)
```

```
[1] 2.5 6.5
```

```
R Console
> x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8) )
> x
[,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> rowMeans(x)
[1] 3 4 5 6
> colMeans(x)
[1] 2.5 6.5
> |
```

# **Foundations of R Software**

## **Lecture 14**

## **Basics of Calculations**

::::

## **Matrix Operations – Access and Mathematical Operations**

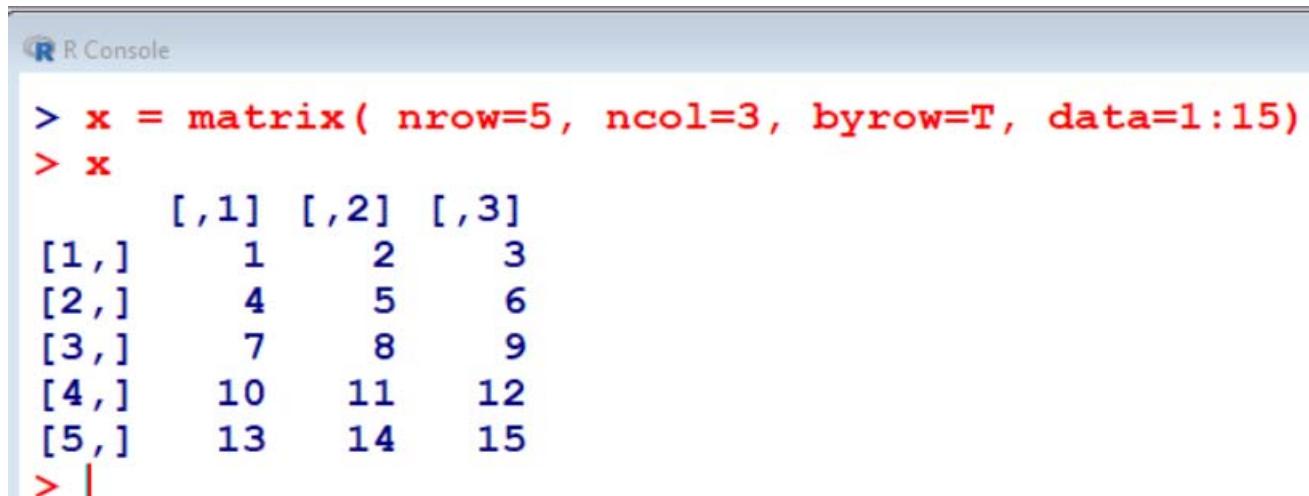
**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Access to rows, columns or submatrices

```
x = matrix( nrow=5, ncol=3, byrow=T, data=1:15)  
x  
     [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9  
[4,]   10   11   12  
[5,]   13   14   15
```



The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R code and its output. The code creates a 5x3 matrix 'x' with data from 1 to 15, row-major order. The output shows the matrix structure with columns labeled [,1], [,2], [,3] and rows labeled [1,] through [5,].

```
> x = matrix( nrow=5, ncol=3, byrow=T, data=1:15)  
> x  
     [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9  
[4,]   10   11   12  
[5,]   13   14   15  
> |
```

# Access to rows, columns or submatrices

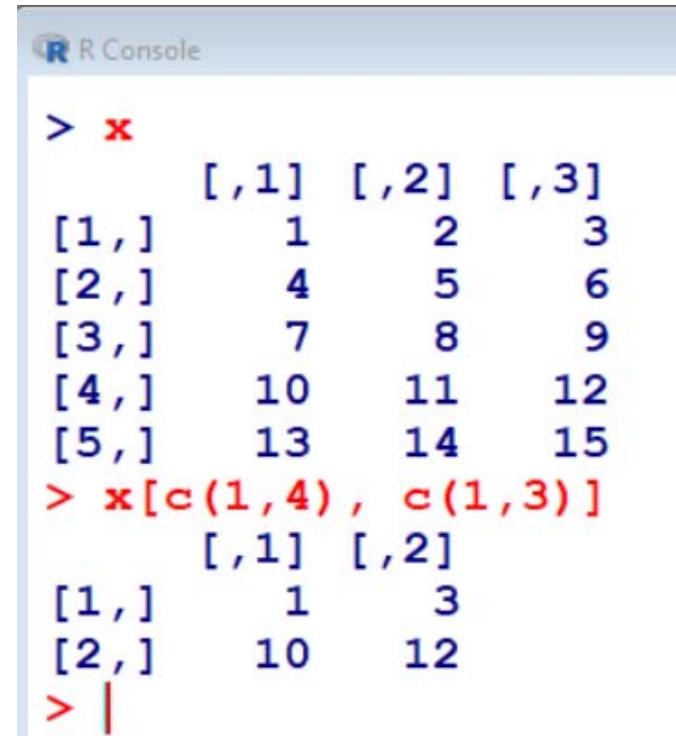
```
> x[3,]  
[1] 7 8 9  
  
> x[,2]  
[1] 2 5 8 11 14  
  
> x[4:5, 2:3]  
     [,1] [,2]  
[1,]    11   12  
[2,]    14   15
```

R Console

```
> x[3,]  
[1] 7 8 9  
>  
> x[,2]  
[1] 2 5 8 11 14  
>  
> x[4:5, 2:3]  
     [,1] [,2]  
[1,]    11   12  
[2,]    14   15
```

# Access to rows, columns or submatrices

```
> x[c(1,4), c(1,3)]  
      [,1] [,2]  
[1,]    1    3  
[2,]   10   12
```



The screenshot shows an R console window titled "R Console". It displays a 5x3 matrix named "x" and a subset of it. The matrix "x" has columns labeled [,1], [,2], and [,3]. The subset selected by the command `x[c(1,4), c(1,3)]` includes the first and fourth rows and the first and third columns, resulting in a 2x2 matrix with columns [,1] and [,2].

```
> x  
      [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9  
[4,]   10   11   12  
[5,]   13   14   15  
> x[c(1,4), c(1,3)]  
      [,1] [,2]  
[1,]    1    3  
[2,]   10   12  
> |
```

# Access to rows, columns or submatrices

```
> x  
      [,1] [,2] [,3]  
[1,]    1    2    3  
[2,]    4    5    6  
[3,]    7    8    9  
[4,]   10   11   12  
[5,]   13   14   15
```

Look at the outcome

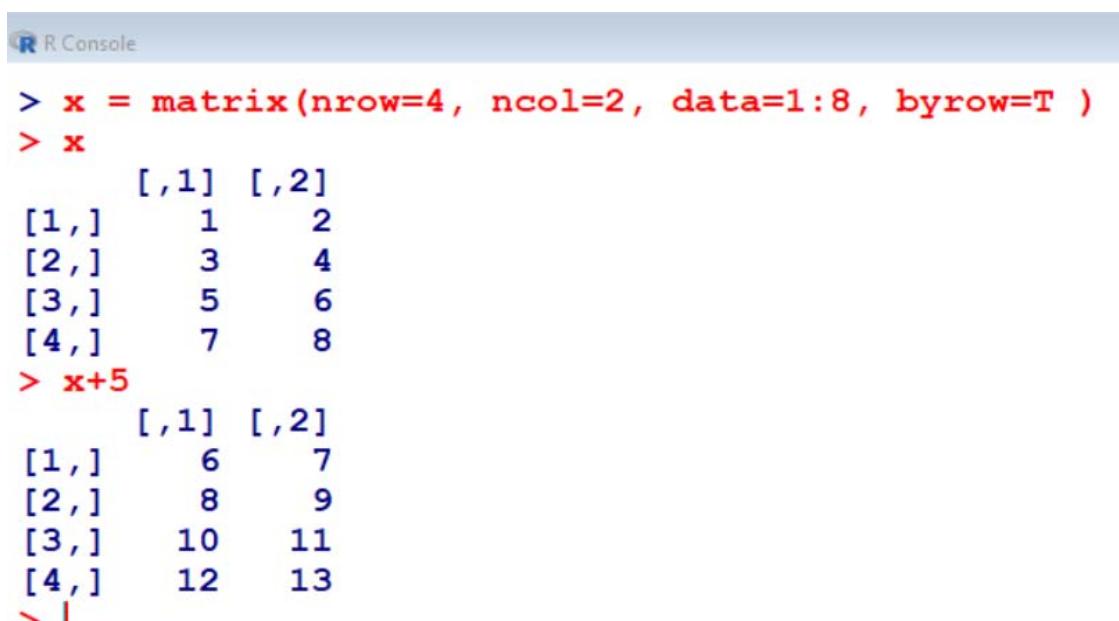
```
[1] 2 5 8 11 14
```

Not possible to know if It is a row or a column.

# Addition of a matrix with a constant

Every element is added with the scalar

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)  
  
> x  
     [,1]   [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
  
> x+5  
     [,1]   [,2]  
[1,]    6    7  
[2,]    8    9  
[3,]   10   11  
[4,]   12   13
```

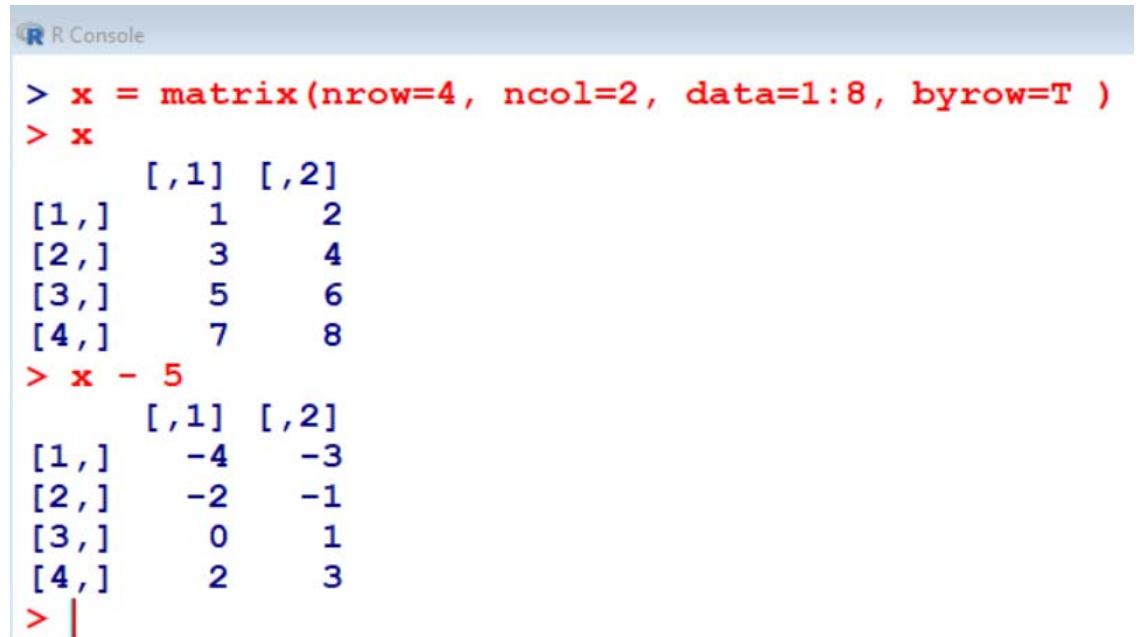


The screenshot shows the R console interface. The title bar says "R Console". The code area contains the R commands shown above, with the output of each command appearing directly below it. The output is color-coded: blue for variable names and function names, black for numbers, and red for operators like '+'. The matrix elements are aligned under their respective column headers [,1] and [,2].

# Subtraction of a matrix with a constant

Every element is subtracted by the scalar

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)  
  
> x  
      [,1]    [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
  
> x - 5  
      [,1]    [,2]  
[1,]   -4   -3  
[2,]   -2   -1  
[3,]    0    1  
[4,]    2    3
```



The screenshot shows the R console window with the following content:

```
R Console  
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T )  
> x  
      [,1]    [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
> x - 5  
      [,1]    [,2]  
[1,]   -4   -3  
[2,]   -2   -1  
[3,]    0    1  
[4,]    2    3
```

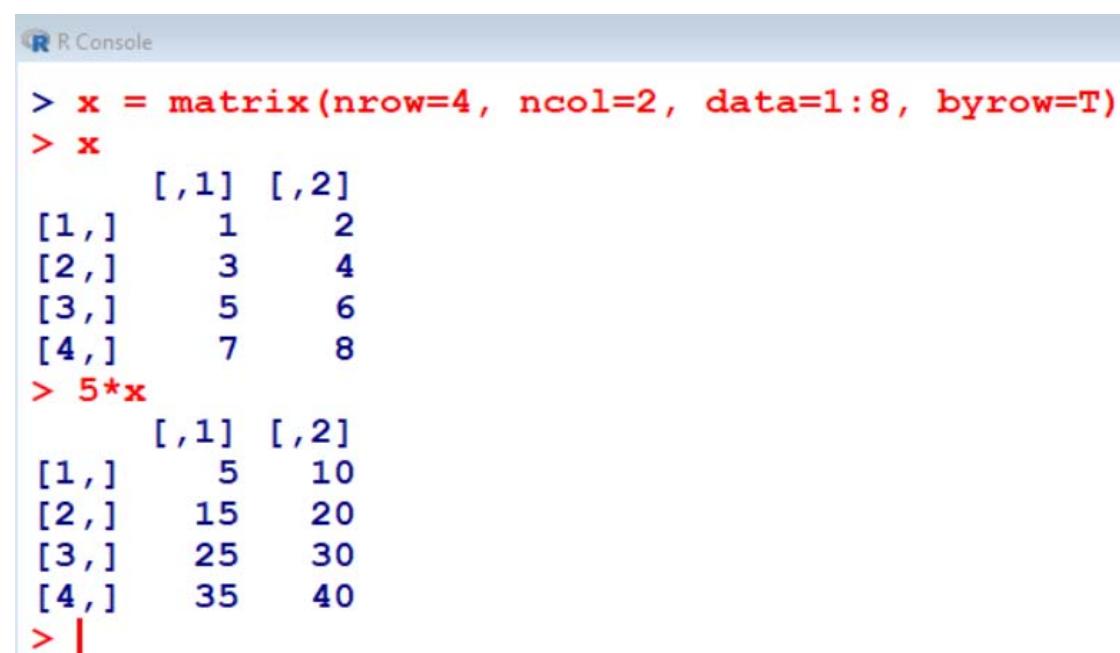
# Multiplication of a matrix with a constant

Every element is multiplied by the scalar

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)

> x
     [,1]   [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> 5*x
     [,1]   [,2]
[1,]    5   10
[2,]   15   20
[3,]   25   30
[4,]   35   40
```



The screenshot shows the R console interface with the title "R Console". Inside the console window, the same R code is run, resulting in the same output as shown on the left. The output is displayed in a light gray background with dark text, and the R prompt (>) is visible at the end of the multiplication line.

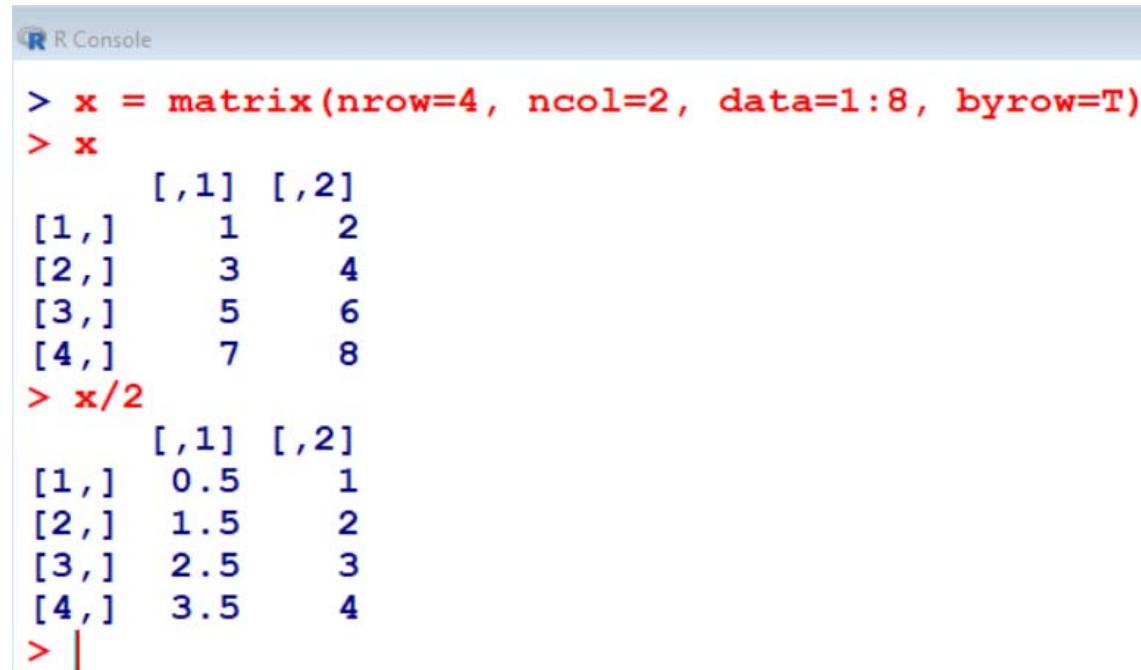
# Division of a matrix with a constant

Every element is divided by the scalar

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)

> x
     [,1]   [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> x/2
     [,1]   [,2]
[1,]  0.5    1
[2,]  1.5    2
[3,]  2.5    3
[4,]  3.5    4
```



The screenshot shows the R console interface. The title bar says "R Console". The code area contains the R commands shown above, with the division command "x/2" highlighted in red. The output area shows the resulting matrix [ ,1 ] [ ,2 ] [1,] 0.5 1 [2,] 1.5 2 [3,] 2.5 3 [4,] 3.5 4.

# Addition and subtraction of matrices

Addition and subtraction of matrices (of same dimensions) can be executed with the usual operators + and -

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
```

```
y = matrix(nrow=4, ncol=2, data=11:18, byrow=T)
```

<pre>&gt; x</pre> <pre>     [,1] [,2]</pre> <pre>[1,]    1    2</pre> <pre>[2,]    3    4</pre> <pre>[3,]    5    6</pre> <pre>[4,]    7    8</pre>	<pre>&gt; y</pre> <pre>     [,1] [,2]</pre> <pre>[1,]   11   12</pre> <pre>[2,]   13   14</pre> <pre>[3,]   15   16</pre> <pre>[4,]   17   18</pre>
--	--

# Addition and subtraction of matrices

```
> x + y
```

```
      [,1] [,2]  
[1,]   12   14  
[2,]   16   18  
[3,]   20   22  
[4,]   24   26
```

```
> x - y
```

```
      [,1] [,2]  
[1,]  -10  -10  
[2,]  -10  -10  
[3,]  -10  -10  
[4,]  -10  -10
```

R Console

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)  
> x  
      [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
> y = matrix(nrow=4, ncol=2, data=11:18, byrow=T)  
> y  
      [,1] [,2]  
[1,]   11   12  
[2,]   13   14  
[3,]   15   16  
[4,]   17   18  
> x+y  
      [,1] [,2]  
[1,]   12   14  
[2,]   16   18  
[3,]   20   22  
[4,]   24   26  
> |
```

# Addition and subtraction of matrices

```
> x - y
      [,1] [,2]
[1,] -10  -10
[2,] -10  -10
[3,] -10  -10
[4,] -10  -10

> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
> x
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> y = matrix(nrow=4, ncol=2, data=11:18, byrow=T)
> y
      [,1] [,2]
[1,]   11   12
[2,]   13   14
[3,]   15   16
[4,]   17   18

> x-y
      [,1] [,2]
[1,] -10  -10
[2,] -10  -10
[3,] -10  -10
[4,] -10  -10
> |
```

# **Foundations of R Software**

## **Lecture 15**

## **Basics of Calculations**

::::

## **Matrix Operations – Mathematical and Other Operations**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Addition and subtraction of matrices

Example:

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
```

```
> x
```

```
     [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8
```

```
> 4*x
```

```
     [,1] [,2]  
[1,]    4    8  
[2,]   12   16  
[3,]   20   24  
[4,]   28   32
```

```
R Console  
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)  
> x  
     [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
> 4*x  
     [,1] [,2]  
[1,]    4    8  
[2,]   12   16  
[3,]   20   24  
[4,]   28   32  
> |
```

# Addition and subtraction of matrices

Example:

```
> x + 4*x  
      [,1]    [,2]  
[1,]    5     10  
[2,]   15     20  
[3,]   25     30  
[4,]   35     40  
  
> 4*x - x  
      [,1]    [,2]  
[1,]    3     6  
[2,]    9    12  
[3,]   15    18  
[4,]   21    24
```

```
R Console  
> x  
      [,1]    [,2]  
[1,]    1     2  
[2,]    3     4  
[3,]    5     6  
[4,]    7     8  
> 4*x  
      [,1]    [,2]  
[1,]    4     8  
[2,]   12    16  
[3,]   20    24  
[4,]   28    32  
> x + 4*x  
      [,1]    [,2]  
[1,]    5    10  
[2,]   15    20  
[3,]   25    30  
[4,]   35    40  
> 4*x - x  
      [,1]    [,2]  
[1,]    3     6  
[2,]    9    12  
[3,]   15    18  
[4,]   21    24  
> |
```

# Multiplication of matrices

Multiplication of matrices can be executed with the operator `%*%`

```
x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
```

```
y = matrix(nrow=2, ncol=4, data=11:18, byrow=T)
```

```
> x
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```

```
> y
      [,1] [,2] [,3] [,4]
[1,]   11   12   13   14
[2,]   15   16   17   18
```

# Multiplication of matrices

```
> x%*%y  
      [,1] [,2] [,3] [,4]  
[1,]    41    44    47    50  
[2,]    93   100   107   114  
[3,]   145   156   167   178  
[4,]   197   212   227   242
```

```
> y%*%x  
      [,1] [,2]  
[1,]   210   260  
[2,]   274   340
```

```
R R Console  
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)  
>  
> y = matrix(nrow=2, ncol=4, data=11:18, byrow=T)  
> x  
      [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6  
[4,]    7    8  
> y  
      [,1] [,2] [,3] [,4]  
[1,]   11   12   13   14  
[2,]   15   16   17   18  
> x%*%y  
      [,1] [,2] [,3] [,4]  
[1,]    41    44    47    50  
[2,]    93   100   107   114  
[3,]   145   156   167   178  
[4,]   197   212   227   242  
> y%*%x  
      [,1] [,2]  
[1,]   210   260  
[2,]   274   340  
> |
```

# Multiplication of matrices

Another example: Consider the multiplication of  $X'$  and  $X$

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
> x
      [,1]    [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> t(x)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

# Multiplication of matrices

```
> t(x) %*% x
```

```
 [,1] [,2]  
[1,] 84 100  
[2,] 100 120
```

```
> x %*% t(x)
```

```
 [,1] [,2] [,3] [,4]  
[1,] 5 11 17 23  
[2,] 11 25 39 53  
[3,] 17 39 61 83  
[4,] 23 53 83 113
```

R Console

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)  
> x  
 [,1] [,2]  
[1,] 1 2  
[2,] 3 4  
[3,] 5 6  
[4,] 7 8  
> t(x)  
 [,1] [,2] [,3] [,4]  
[1,] 1 3 5 7  
[2,] 2 4 6 8  
> t(x) %*% x  
 [,1] [,2]  
[1,] 84 100  
[2,] 100 120  
> x %*% t(x)  
 [,1] [,2] [,3] [,4]  
[1,] 5 11 17 23  
[2,] 11 25 39 53  
[3,] 17 39 61 83  
[4,] 23 53 83 113
```

# Multiplication of matrices

Cross Product of a matrix X,  $X'X$ , with `crossprod()`

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
> x
      [,1]    [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> t(x)
      [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

# Multiplication of matrices

```
> crossprod(x)
      [,1] [,2]
[1,]    84   100
[2,]   100   120
```

```
R Console
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
> x
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> crossprod(x)
      [,1] [,2]
[1,]    84   100
[2,]   100   120
>
> t(x) %*% x
      [,1] [,2]
[1,]    84   100
[2,]   100   120
> |
```

Note: Command `crossprod()` executes the multiplication faster than the conventional method with `t(x)%*%x`

# Concatenating matrices

Concatenating matrices row wise: `rbind(x, y)`

Concatenating matrices column wise: `cbind(x, y)`

```
x = matrix(nrow=3, ncol=2, data=1:6, byrow=T)
```

```
y = matrix(nrow=3, ncol=2, data=11:16, byrow=T)
```

> x	> y	> rbind(x, y)
[,1] [,2]	[,1] [,2]	[,1] [,2]
[1,] 1 2	[1,] 11 12	[1,] 1 2
[2,] 3 4	[2,] 13 14	[2,] 3 4
[3,] 5 6	[3,] 15 16	[3,] 5 6
		[4,] 11 12
		[5,] 13 14
		[6,] 15 16

# Concatenating matrices

Concatenating matrices row wise: `rbind(x, y)`

Concatenating matrices column wise: `cbind(x, y)`

```
x = matrix(nrow=3, ncol=2, data=1:6, byrow=T)
```

```
y = matrix(nrow=3, ncol=2, data=11:16, byrow=T)
```

```
> x  
      [,1] [,2]  
[1,]    1    2  
[2,]    3    4  
[3,]    5    6
```

```
> y  
      [,1] [,2]  
[1,]   11   12  
[2,]   13   14  
[3,]   15   16
```

```
> cbind(x, y)  
      [,1] [,2] [,3] [,4]  
[1,]    1    2   11   12  
[2,]    3    4   13   14  
[3,]    5    6   15   16
```

# Concatenating matrices

Concatenating matrices row wise: **rbind(x, y)**

Concatenating matrices column wise: **cbind(x, y)**

```
R Console

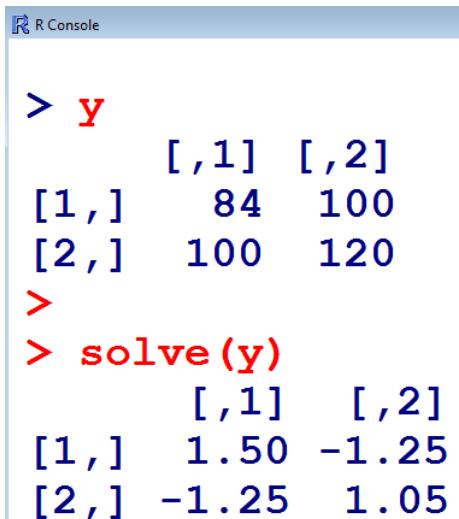
> x = matrix(nrow=3, ncol=2, data=1:6, byrow=T)
> y = matrix(nrow=3, ncol=2, data=11:16, byrow=T)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> y
     [,1] [,2]
[1,]   11   12
[2,]   13   14
[3,]   15   16
> rbind(x, y)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]   11   12
[5,]   13   14
[6,]   15   16
> cbind(x, y)
     [,1] [,2] [,3] [,4]
[1,]    1    2   11   12
[2,]    3    4   13   14
[3,]    5    6   15   16
```

# Inverse of matrix

`solve()` finds the inverse of a positive definite matrix

```
> y = matrix( nrow = 2, ncol = 2, byrow = T,  
data = c(84,100,100,120))
```

```
> y  
     [,1] [,2]  
[1,]    84   100  
[2,]   100   120  
  
> solve(y)  
     [,1]  [,2]  
[1,]  1.50 -1.25  
[2,] -1.25  1.05
```



R Console

```
> y  
     [,1] [,2]  
[1,]    84   100  
[2,]   100   120  
>  
> solve(y)  
     [,1]  [,2]  
[1,]  1.50 -1.25  
[2,] -1.25  1.05
```

# Eigen values and eigen vectors of matrix

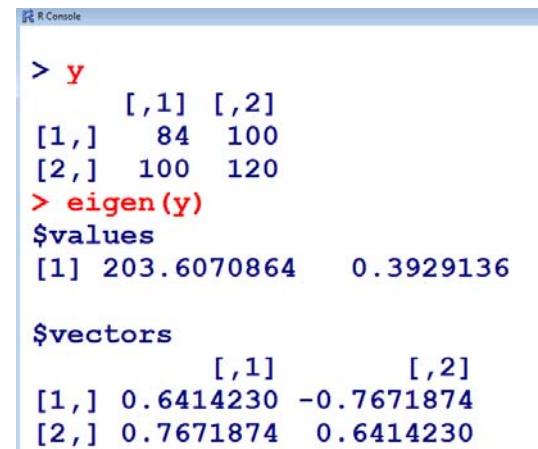
`eigen()` finds the eigen values and eigen vectors of a positive definite matrix

```
y = matrix( nrow = 2, ncol = 2, byrow = T, data  
= c(84,100,100,120))
```

```
> y  
      [,1] [,2]  
[1,]    84   100  
[2,]   100   120
```

# Eigen values and eigen vectors of matrix

```
> eigen(y)  
$values  
[1] 203.6070864      0.3929136  
  
$vectors  
          [,1]            [,2]  
[1,] 0.6414230 -0.7671874  
[2,] 0.7671874  0.6414230
```



The screenshot shows an R console window titled "R Console". It contains the following R code and its output:

```
> y  
     [,1] [,2]  
[1,]   84   100  
[2,]  100   120  
> eigen(y)  
$values  
[1] 203.6070864      0.3929136  
  
$vectors  
          [,1]            [,2]  
[1,] 0.6414230 -0.7671874  
[2,] 0.7671874  0.6414230
```

# **Foundations of R Software**

## **Lecture 16**

## **Basics of Calculations**

::::

## **Logical Operators**

Shalabh

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Logical Operators and Comparisons

The following table shows the operations and functions for logical comparisons (True or False).

Operator	Executions
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Exactly equal to
!=	Not equal to
!	Negation (not)
&, &&	and
,	or

Operator	Executions
xor(x,y)	either... or (exclusive)
isTRUE(x)	test if x is TRUE
isFALSE(x)	test if x is FALSE
TRUE	true
FALSE	false

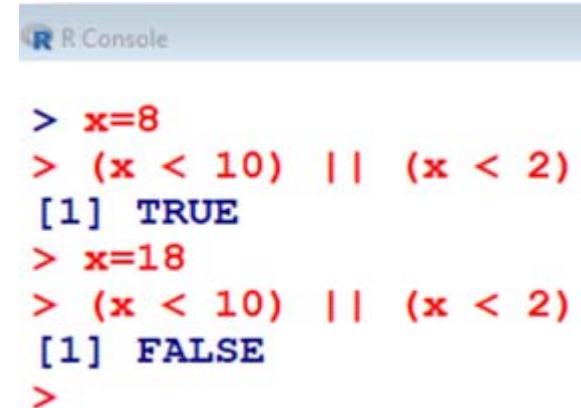
# Logical Operators and Comparisons

Operator	Executions
&, &&	and
,	or

- The shorter form performs element-wise comparisons in almost the same way as arithmetic operators.
- The longer form evaluates left to right examining only the first element of each vector. Evaluation proceeds only until the result is determined.

## Examples: Use of | and ||

```
> x=8  
> (x < 10) || (x < 2)  
[1] TRUE
```



R Console

```
> x=8  
> (x < 10) || (x < 2)  
[1] TRUE  
> x=18  
> (x < 10) || (x < 2)  
[1] FALSE  
>
```

```
> x=18  
> (x < 10) || (x < 2)  
[1] FALSE
```

## Examples: Use of | and ||

```
> x = c(8,18)
> (x < 10) || (x < 2)
[1] TRUE
```

|| Operates only on the first element and not on remaining elements.



```
> x = c(8,18)
> (x < 10) || (x < 2)
[1] TRUE
>
> (x < 10) | (x < 2)
[1] TRUE FALSE
> |
```

```
> (x < 10) | (x < 2)
[1] TRUE FALSE
| Operates on all the elements.
```

## Examples: Use of & and &&

```
> x = 5  
> (x < 10) && (x > 2)  
[1] TRUE
```

R Console

```
> x = 5  
> (x < 10) && (x > 2)  
[1] TRUE  
>  
> x = 15  
> (x < 10) && (x > 2)  
[1] FALSE  
> |
```

```
> x = 15  
> (x < 10) && (x > 2)  
[1] FALSE
```

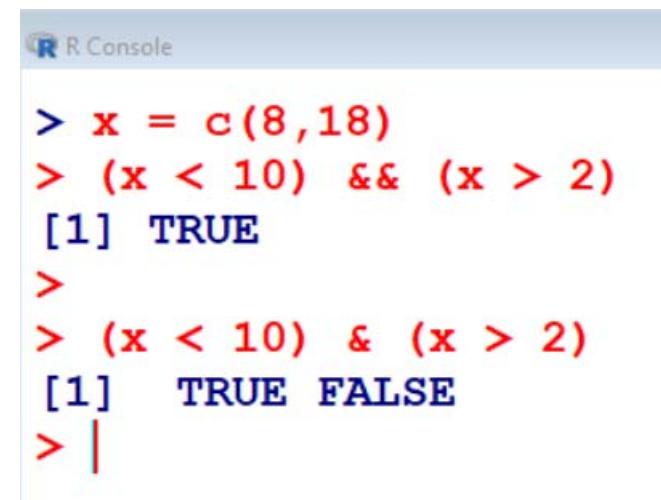
## Examples: Use of & and &&

```
> x = c(8,18)
> (x < 10) && (x > 2)
[1] TRUE
```

&& Operates only on the first element and not on remaining elements.

```
> (x < 10) & (x > 2)
[1] TRUE FALSE
```

& Operates on all the elements.



The screenshot shows the R Console interface with a light blue header bar containing the R logo and the text "R Console". Below the header, there is a scrollable text area displaying R code and its output. The code examples are identical to those shown on the left of the slide, demonstrating the difference between the logical operators & and &&.

```
R Console
> x = c(8,18)
> (x < 10) && (x > 2)
[1] TRUE
>
> (x < 10) & (x > 2)
[1] TRUE FALSE
> |
```

## Examples:

```
> x = 1:6      # Generates x=1,2,3,4,5,6
```

```
> (x > 2) & (x < 5)  # Checks whether the  
values are greater  
than 2 and less than 5
```

```
[1] FALSE FALSE TRUE TRUE FALSE FALSE
```

```
> x[(x > 2) & (x < 5)] # Finds which values  
are greater than 2 and  
smaller than 5
```

```
[1] 3 4
```

## Examples:

```
R Console
> x = 1:6
> x
[1] 1 2 3 4 5 6
>
> (x > 2) & (x < 5)
[1] FALSE FALSE TRUE TRUE FALSE FALSE
>
> x[(x > 2) & (x < 5)]
[1] 3 4
```

## Examples:

```
> x = 1:6      # Generates x=1,2,3,4,5,6

> (x > 2) | (x < 5) # Checks whether the
                        values are greater
                        than 2 or less than 5

[1] TRUE TRUE TRUE TRUE TRUE TRUE

> x[(x > 2) | (x < 5)] # Finds which values
                            are greater than 2 or
                            smaller than 5

[1] 1 2 3 4 5 6
```

## Examples:

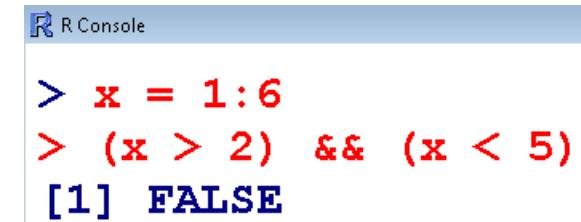
```
R Console

> x = 1:6
> (x > 2) | (x < 5)
[1] TRUE TRUE TRUE TRUE TRUE TRUE
>
> x[(x > 2) | (x < 5)]
[1] 1 2 3 4 5 6
```

## Example of “The longer form evaluates left to right examining only the first element of each vector”

```
> x = 1:6      # Generates x = 1,2,3,4,5,6
```

```
> (x > 2) && (x < 5)  
[1] FALSE
```



```
R Console  
> x = 1:6  
> (x > 2) && (x < 5)  
[1] FALSE
```

is equivalent to:

```
> (x[1] > 2) & (x[1] < 5)  
[1] FALSE
```



```
R Console  
> (x[1] > 2) & (x[1] < 5)  
[1] FALSE
```

Note that `x[1]` is only the first element in `x`

# **Foundations of R Software**

## **Lecture 17**

## **Basics of Calculations**

::::

## **Relational and Logical Operators**

Shalabh

Department of Mathematics and Statistics

Indian Institute of Technology Kanpur

# Relational Operators and Comparisons

The following table shows the operations and functions for relational comparisons (True or False).

**TRUE** and **FALSE** are reserved words denoting logical constants.

Operator	Executions
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Exactly equal to
!=	Not equal to
!	Negation (not)

Zero is considered as  
**FALSE**  
and  
non-zero numbers are  
taken as **TRUE**

# Logical Operators and Comparisons

Operator	Executions	
!	Logical not	
&	Element-wise logical and	Performs operation element-wise
&&	Logical and	examines only the first element of the operands
	Element-wise logical or	Performs operation element-wise
	Logical or	examines only the first element of the operands

# Logical Operators and Comparisons

Operator	Executions
<b>xor(x,y)</b>	Either <b>x or y</b> <b>(x or y but not x and y )</b>
<b>isTRUE(x)</b>	<b>test if x is TRUE</b>
<b>isFALSE(x)</b>	<b>test if x is FALSE</b>
<b>TRUE</b>	<b>true</b>
<b>FALSE</b>	<b>false</b>

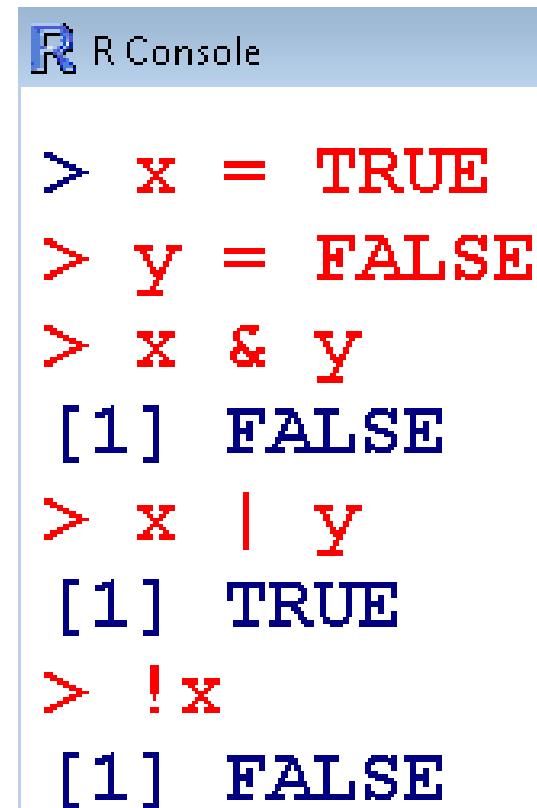
# Example of Standard logical operations

## Truth table

Statement 1 :: (x)	Statement 2 :: (y)	Outcome :: x and y	Outcome :: x or y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

# Example of Standard logical operations

```
> x = TRUE  
  
> y = FALSE  
  
> x & y      # x AND y  
[1] FALSE  
  
> x | y      # x OR y  
[1] TRUE  
  
> !x          # negation of x  
[1] FALSE
```



A screenshot of an R console window titled "R Console". The window shows the same R code as the left side, but the output is displayed in red text. The R logo is visible in the top-left corner of the window.

```
> x = TRUE  
 > y = FALSE  
 > x & y  
 [1] FALSE  
 > x | y  
 [1] TRUE  
 > !x  
 [1] FALSE
```

## Example

```
> x = 5
```

```
> Logical1 = (x > 2)
```

```
> Logical1
```

```
[1] TRUE
```

```
> is.logical(Logical1)
```

```
[1] TRUE
```

```
> Logical2 = (x < 10)
```

```
> Logical2
```

```
[1] TRUE
```

```
> is.logical(Logical2)
```

```
[1] TRUE
```

```
> Logical3 = (x != 5)
```

```
> Logical3
```

```
[1] FALSE
```

```
> is.logical(Logical3)
```

```
[1] TRUE
```

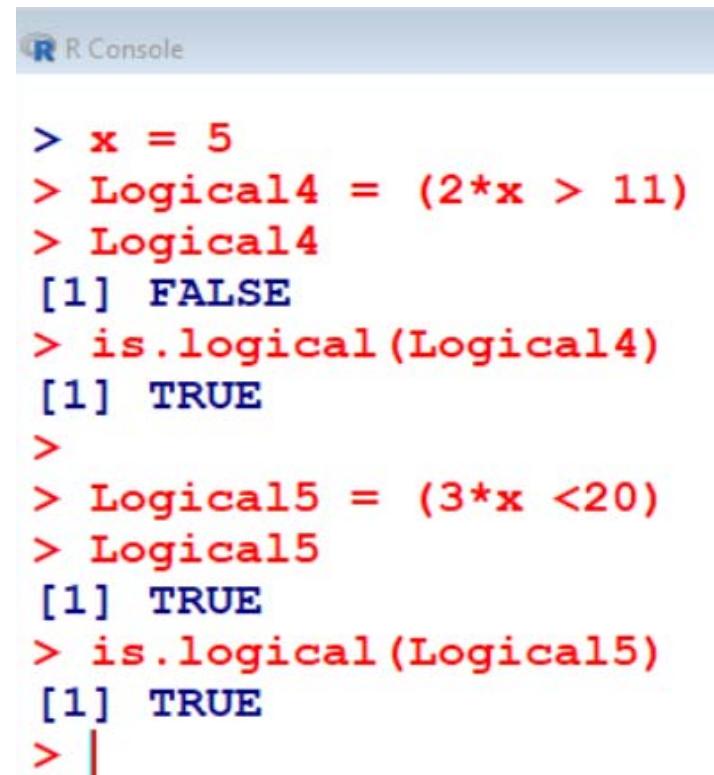
# Example

```
R R Console

> x = 5
> Logical1 = (x > 2)
> Logical1
[1] TRUE
> is.logical(Logical1)
[1] TRUE
>
> Logical2 = (x < 10)
> Logical2
[1] TRUE
> is.logical(Logical2)
[1] TRUE
>
> Logical3 = (x != 5)
> Logical3
[1] FALSE
> is.logical(Logical3)
[1] TRUE
> |
```

## Example

```
> x = 5  
  
> Logical4 = (2*x > 11)  
  
> Logical4  
[1] FALSE  
  
> is.logical(Logical4)  
[1] TRUE  
  
> Logical5 = (3*x <20)  
  
> Logical5  
[1] TRUE  
  
> is.logical(Logical5)  
[1] TRUE
```



The image shows a screenshot of an R console window. The title bar says "R Console". The main area contains R code and its corresponding output. The code is identical to the one shown on the left, demonstrating the creation of logical vectors and their properties.

```
> x = 5  
> Logical4 = (2*x > 11)  
> Logical4  
[1] FALSE  
> is.logical(Logical4)  
[1] TRUE  
>  
> Logical5 = (3*x <20)  
> Logical5  
[1] TRUE  
> is.logical(Logical5)  
[1] TRUE  
> |
```

## Examples:

```
> 8 > 7
```

```
[1] TRUE
```

```
> 7 < 5
```

```
[1] FALSE
```

```
> 7 > 7
```

```
[1] FALSE
```

```
> 7 >= 7
```

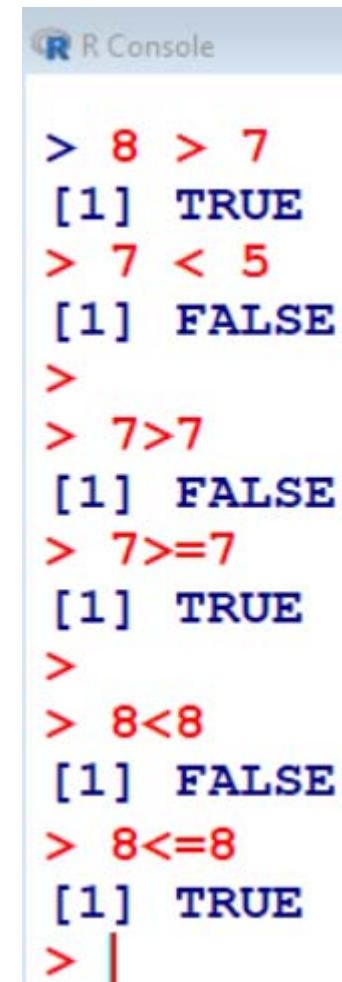
```
[1] TRUE
```

```
> 8 < 8
```

```
[1] FALSE
```

```
> 8 <= 8
```

```
[1] TRUE
```



The screenshot shows an R console window titled "R Console". It displays several R commands and their outputs. The commands are in red, and the outputs are in blue. The commands include comparisons like >, <, >=, <=, and logical operators like & and |.

```
> 8 > 7
[1] TRUE
> 7 < 5
[1] FALSE
>
> 7>7
[1] FALSE
> 7>=7
[1] TRUE
>
> 8<8
[1] FALSE
> 8<=8
[1] TRUE
> |
```

## Examples:

```
> 8 != 9
```

```
[1] TRUE
```

```
> 9 != 9
```

```
[1] FALSE
```

```
> 7 == 7
```

```
[1] TRUE
```

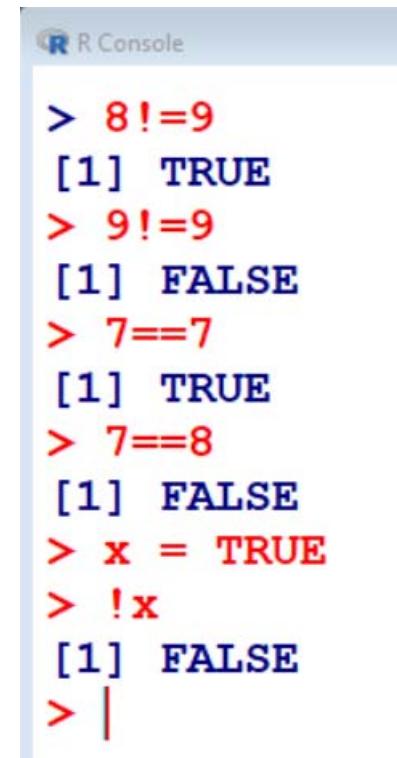
```
> 7 == 8
```

```
[1] FALSE
```

```
> x = TRUE
```

```
> !x
```

```
[1] FALSE
```



The screenshot shows an R console window titled "R Console". It contains the following R code and output:

```
> 8!=9
[1] TRUE
> 9!=9
[1] FALSE
> 7==7
[1] TRUE
> 7==8
[1] FALSE
> x=TRUE
> !x
[1] FALSE
> |
```

## Examples:

```
> x = c(1, 2, 3)
```

```
> y = c(4, 5, 6)
```

```
> x > y      # Compares 1 > 4, 2 > 5, 3 > 6  
[1] FALSE FALSE FALSE
```

```
> x < y      # Compares 1 < 4, 2 < 5, 3 < 6  
[1] TRUE TRUE TRUE
```

```
> x != y      # Compares 1 ≠ 4, 2 ≠ 5, 3 ≠ 6  
[1] TRUE TRUE TRUE
```

```
> x == y      # Compares 1 = 4, 2 = 5, 3 = 6  
[1] FALSE FALSE FALSE
```

```
R R Console  
> x = c(1, 2, 3)  
> y = c(4, 5, 6)  
> x > y  
[1] FALSE FALSE FALSE  
>  
> x < y  
[1] TRUE TRUE TRUE  
>  
> x != y  
[1] TRUE TRUE TRUE  
>  
> x == y  
[1] FALSE FALSE FALSE  
> |
```

## Examples:

Is 8 less than 6?

```
> isTRUE(8 < 6)
[1] FALSE
```

Is 8 greater than 6?

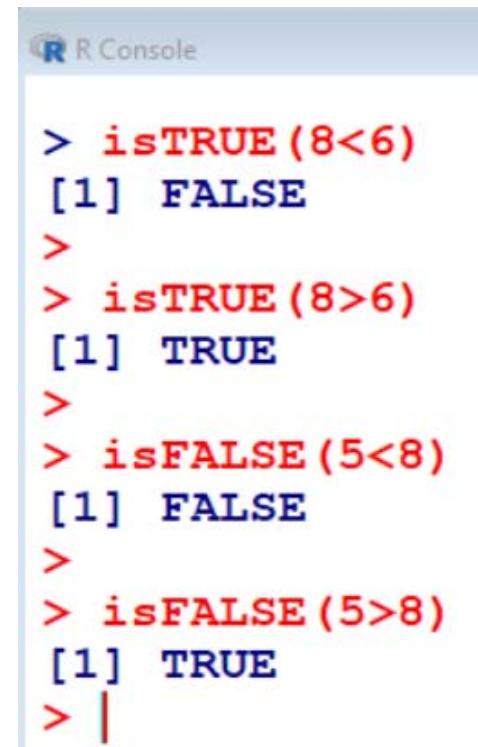
```
> isTRUE(8 > 6)
[1] TRUE
```

Is 5 less than 8?

```
> isFALSE(5 < 8)
[1] FALSE
```

Is 5 greater than 8?

```
> isFALSE(5 > 8)
[1] TRUE
```

A screenshot of an R console window titled "R Console". The window shows several lines of R code in red and blue, followed by their corresponding output in blue. The code includes comparisons like 8<6, 8>6, 5<8, and 5>8, along with isTRUE and isFALSE functions.

```
> isTRUE(8<6)
[1] FALSE
>
> isTRUE(8>6)
[1] TRUE
>
> isFALSE(5<8)
[1] FALSE
>
> isFALSE(5>8)
[1] TRUE
> |
```

# **Foundations of R Software**

## **Lecture 18**

## **Basics of Calculations**

::::

## **Missing Data Handling**

Shalabh

**Department of Mathematics and Statistics**

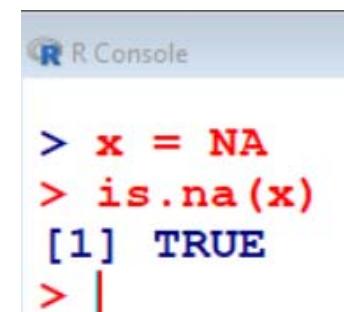
**Indian Institute of Technology Kanpur**

## Missing data

R represents missing observations through the data value **NA**

We can detect missing values using **is.na**

The command **is.na( )** returns a logical vector with **TRUE** in the element locations that contain missing values represented by **NA**.

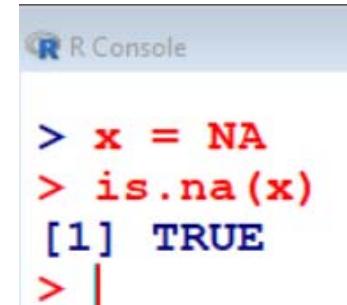


```
R Console
> x = NA
> is.na(x)
[1] TRUE
> |
```

## Missing data

`is.na()` will work on vectors, lists, matrices, and data frames.

```
> x = NA      # assign NA to variable x  
> is.na(x)   # is it missing?  
[1] TRUE
```

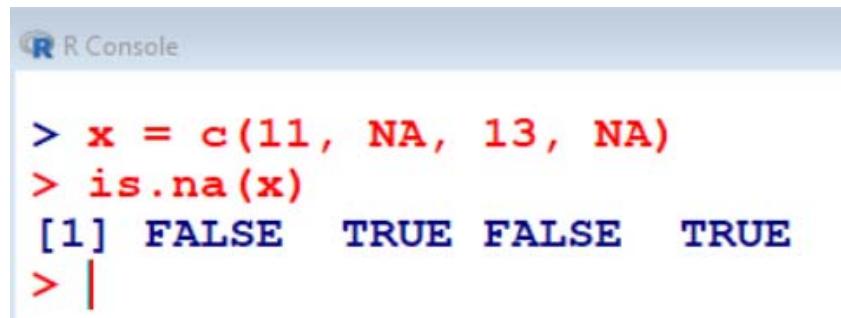


The image shows a screenshot of an R console window. The title bar says "R Console". The main area contains the following text:  
> x = NA  
> is.na(x)  
[1] TRUE  
> |

## Missing data

Now try a vector to know if any value is missing?

```
> x = c(11, NA, 13, NA)  
  
> is.na(x)  
[1] FALSE  TRUE FALSE  TRUE
```



The screenshot shows the R Console window. The title bar says "R Console". The console area contains the following text:

```
> x = c(11, NA, 13, NA)
> is.na(x)
[1] FALSE  TRUE FALSE  TRUE
> |
```

## Example : How to work with missing data

```
> x = c(11,NA,13, NA) # vector
```

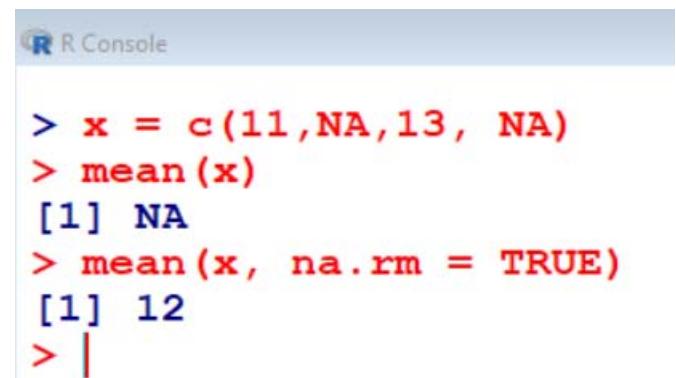
```
> mean(x)    $\frac{11+NA+13+NA}{4}$   
[1] NA
```

```
> mean(x, na.rm = TRUE) # NAs can be removed
```

```
[1] 12  

$$\frac{11+13}{2} = 12$$

```



R Console

```
> x = c(11,NA,13, NA)  
> mean(x)  
[1] NA  
> mean(x, na.rm = TRUE)  
[1] 12  
> |
```

## **NA versus NULL**

The null object, called **NULL**, is returned by some functions and expressions.

Note that **NA** and **NULL** are not the same.

**NA** is a placeholder for something that exists but is missing.

**NULL** stands for something that never existed at all.

## Missing data: Location of missing values

To identify the location of NAs, use `which()` function as

```
which(is.na( ))
```

```
> x = c(11,NA,13,NA)
```

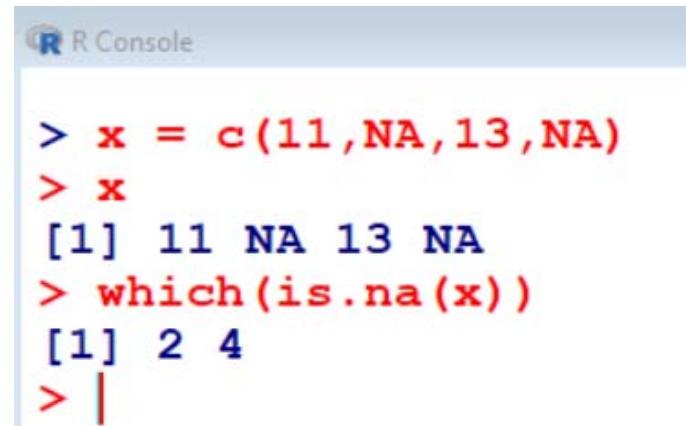
```
> x
```

```
[1] 11 NA 13 NA
```

```
> which(is.na(x))
```

```
[1] 2 4
```

It indicates that the missing values occur at 2<sup>nd</sup> and 4<sup>th</sup> places.



The screenshot shows an R console window titled "R Console". The code entered is:

```
> x = c(11,NA,13,NA)
> x
[1] 11 NA 13 NA
> which(is.na(x))
[1] 2 4
> |
```

The output shows the vector `x` with two NA values at positions 2 and 4, and the command `which(is.na(x))` returning the indices 2 and 4 where the values are NA.

## Missing data

To count of NAs the number of **NAs** , use **sum( )** function as

```
sum(is.na( ))
```

```
> x = c(11,NA,13,NA)
```

```
> x
```

```
[1] 11 NA 13 NA
```

```
> sum(is.na(x))
```

```
[1] 2
```

It indicates that there are 2 missing values.

R R Console

```
> x = c(11,NA,13,NA)
> x
[1] 11 NA 13 NA
> sum(is.na(x))
[1] 2
> |
```

## Missing data: Finding complete cases

To find complete cases, use `complete.cases()` function which returns a logical vector identifying rows which are complete cases.

```
> x = c(11,NA,13,NA)  
  
> x  
[1] 11 NA 13 NA
```

```
> complete.cases(x)  
[1] TRUE FALSE TRUE FALSE
```

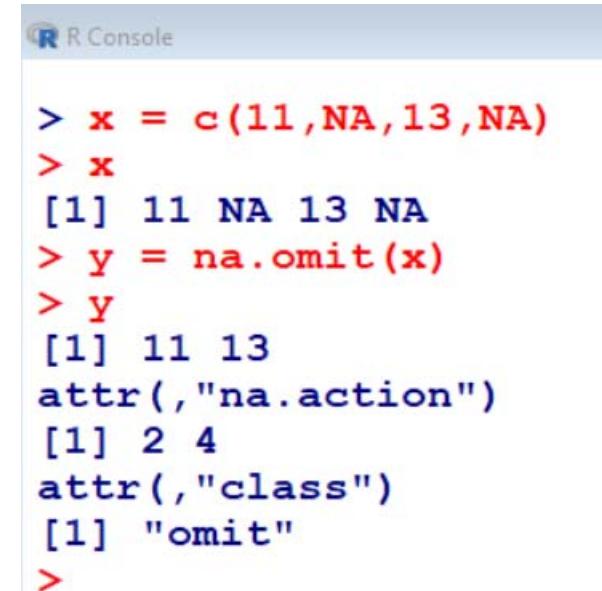
It indicates that the values at 1<sup>st</sup> and 3<sup>rd</sup> places are not missing.



## Missing data: Finding complete data set

The function `na.omit()` returns the object with listwise deletion of missing values. Drop out any rows with missing values anywhere in them and forgets them forever.

```
> x = c(11,NA,13,NA)
> x
[1] 11 NA 13 NA
> y = na.omit(x)
> y
[1] 11 13
attr(,"na.action")
[1] 2 4
attr(,"class")
[1] "omit"
```



R Console

```
> x = c(11,NA,13,NA)
> x
[1] 11 NA 13 NA
> y = na.omit(x)
> y
[1] 11 13
attr(,"na.action")
[1] 2 4
attr(,"class")
[1] "omit"
>
```

## Missing data: Handling values

```
> x = c(11,NA,13,NA)  
  
> y = na.omit(x)  
  
> y  
[1] 11 13  
attr(,"na.action")  
[1] 2 4  
attr(,"class")  
[1] "omit"  
  
> mean(x)  
[1] NA  
> mean(y)  
[1] 12
```

R Console

```
> x = c(11,NA,13,NA)  
> x  
[1] 11 NA 13 NA  
> y = na.omit(x)  
> y  
[1] 11 13  
attr(,"na.action")  
[1] 2 4  
attr(,"class")  
[1] "omit"  
>  
> mean(x)  
[1] NA  
> mean(y)  
[1] 12  
> |
```

# **Foundations of R Software**

## **Lecture 19**

## **Basics of Calculations**

::::

## **Conditional Executions – If and If-Else**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# **Control structures in R :**

**Control statements,**

**Functions,**

**Loops.**

# 1. Conditional execution: if()

## Syntax

```
if (condition) {execute commands if condition is TRUE}
```

The code in the {} is evaluated if the logical test contained in the () is TRUE.

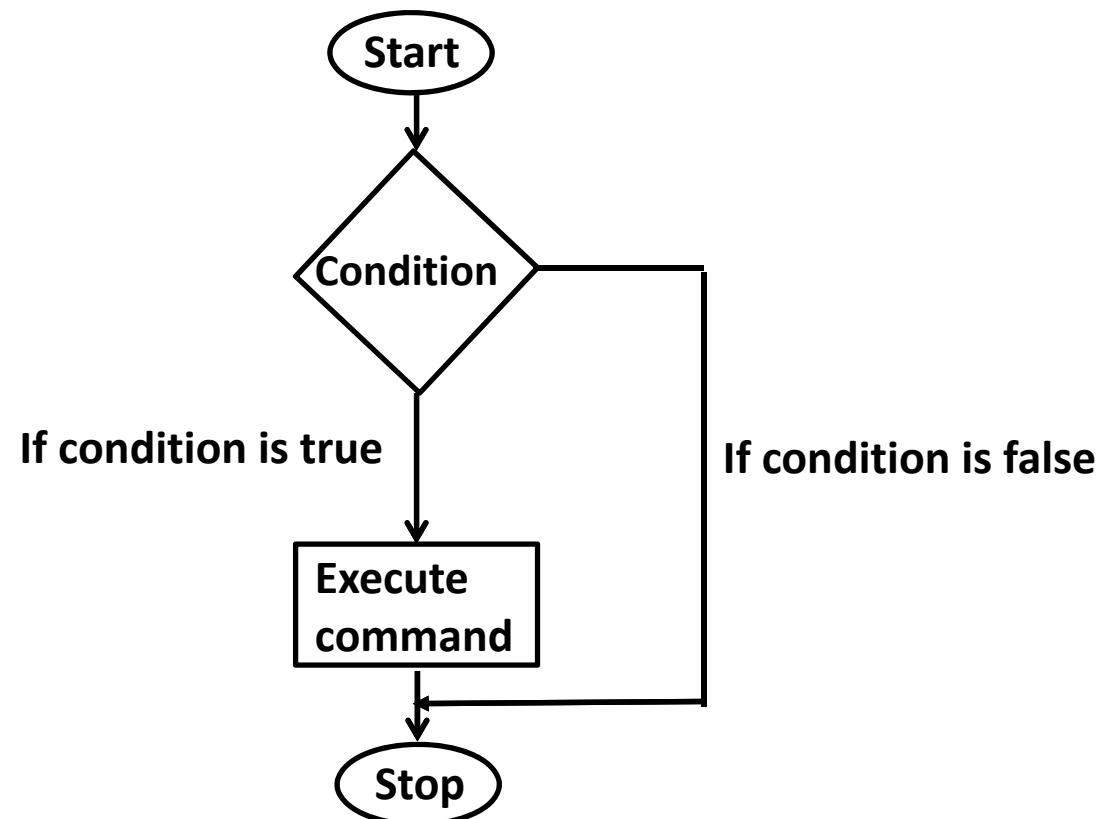
If the logical test is FALSE, R will ignore all of the code in the {}.

if() should not be applied when the condition being evaluated is a vector. It is best used only when meeting a single element condition.

# 1. Conditional execution: if()

## Syntax

```
if (condition) {execute commands if condition is  
TRUE}
```

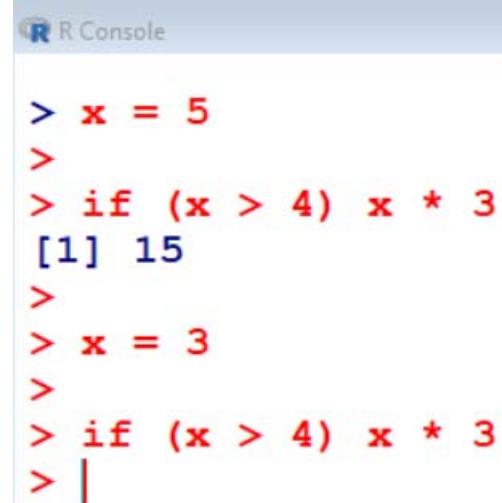


# 1. Conditional execution: if()

## Example 1:

Suppose we want to multiply all the values  $x > 4$  by 3.

```
> x = 5  
  
> if (x > 4) x * 3  
[1] 15  
  
> x = 3  
  
> if (x > 4) x * 3
```



```
R Console  
> x = 5  
>  
> if (x > 4) x * 3  
[1] 15  
>  
> x = 3  
>  
> if (x > 4) x * 3  
> |
```

No response is obtained.

# 1. Conditional execution: if()

Example 1: Same with using {}

Suppose we want to multiply all the values  $x > 4$  by 3.

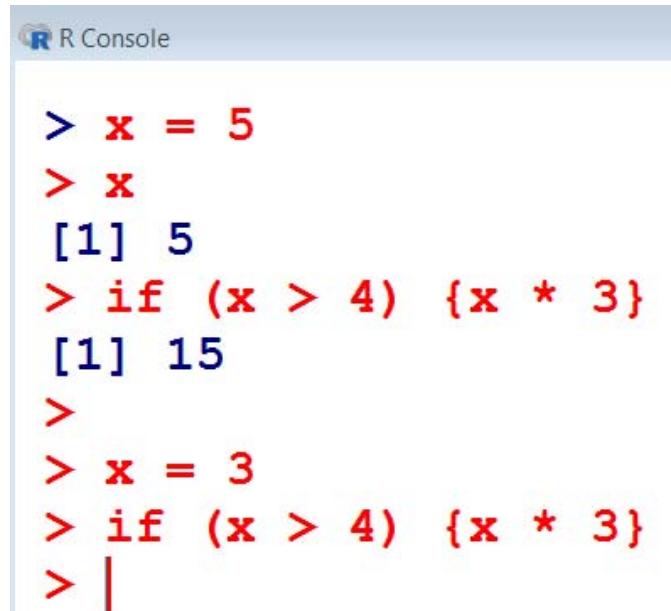
```
> x = 5
```

```
> if (x > 4) {x * 3}  
[1] 15
```

```
> x = 3
```

```
> if (x > 4) {x * 3}
```

No response is obtained.



```
R Console  
> x = 5  
> x  
[1] 5  
> if (x > 4) {x * 3}  
[1] 15  
>  
> x = 3  
> if (x > 4) {x * 3}  
> |
```

# 1. Conditional execution: if()

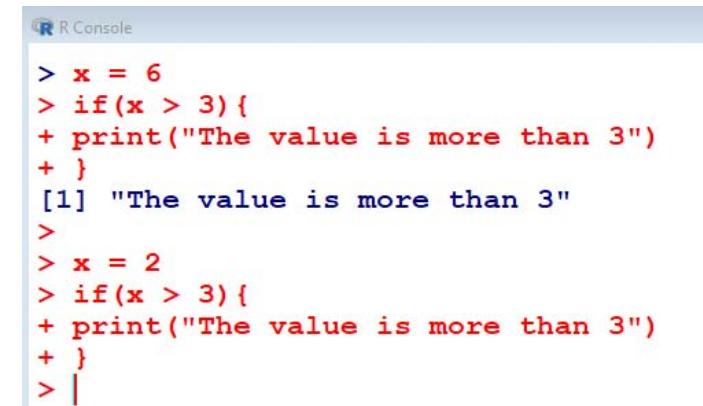
## Example 2:

Suppose we want to print if a value is more than 3.

```
x = 6
```

```
if(x > 3){  
  print("The value is more than 3")  
}
```

```
[1] "The value is more than 3"
```



The screenshot shows the R console interface. The title bar says 'R Console'. The main area contains the R code and its output. The code is:  
> x = 6  
> if(x > 3){  
+ print("The value is more than 3")  
+ }  
[1] "The value is more than 3"  
>  
> x = 2  
> if(x > 3){  
+ print("The value is more than 3")  
+ }  
> |

# 1. Conditional execution: if()

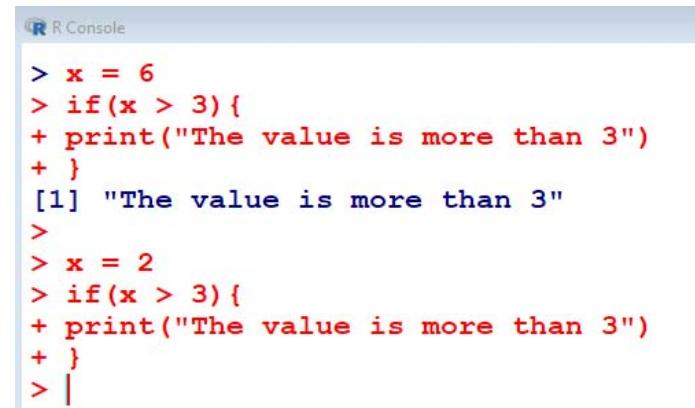
## Example 3:

Suppose we want to print if a value is more than 3.

```
x = 2
```

```
if(x > 3){  
  print("The value is more than 3")  
}
```

No outcome is obtained.

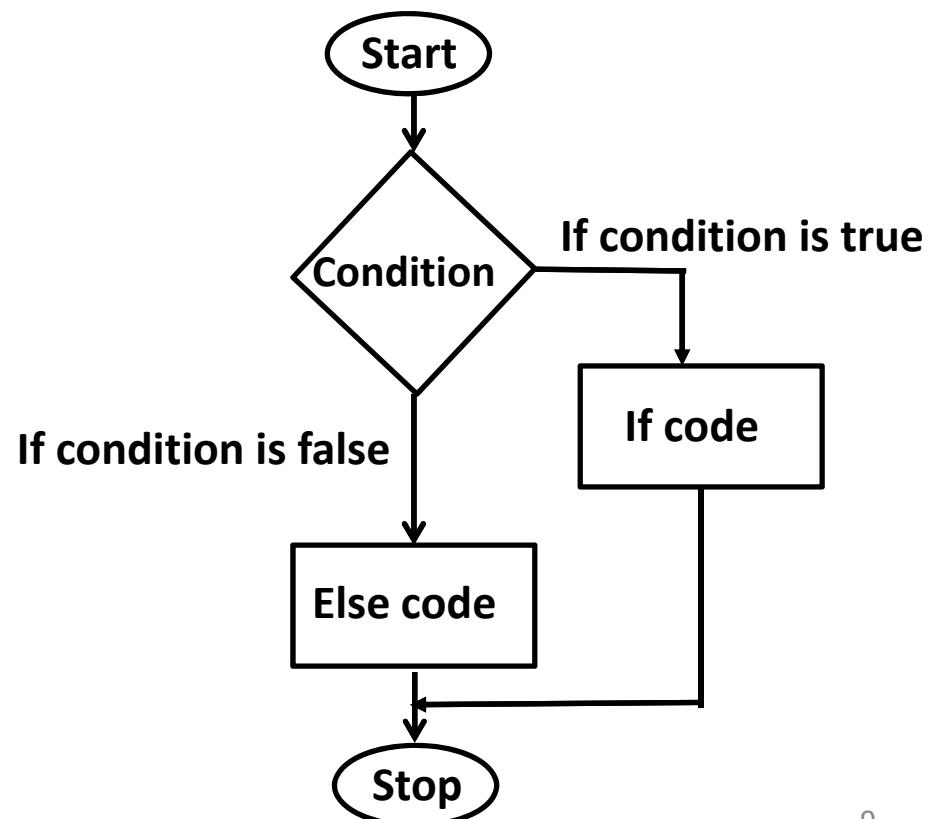


```
R Console  
> x = 6  
> if(x > 3){  
+   print("The value is more than 3")  
+ }  
[1] "The value is more than 3"  
>  
> x = 2  
> if(x > 3){  
+   print("The value is more than 3")  
+ }  
> |
```

## 2. Conditional execution: if else()

### Syntax

```
if (condition) { executes commands if condition is TRUE}  
else { executes commands if condition is FALSE }
```



## 2. Conditional execution: if else()

Please note:

- The condition in this control statement may not be vector valued and if so, only the first element of the vector is used.
- `if else()` should not be applied when the `condition` being evaluated is a vector. It is best used only when meeting a single element condition.
- The condition may be a complex expression where the logical operators "and" (`&&`) and "or" (`||`) can be used.

## 2. Conditional execution: if else()

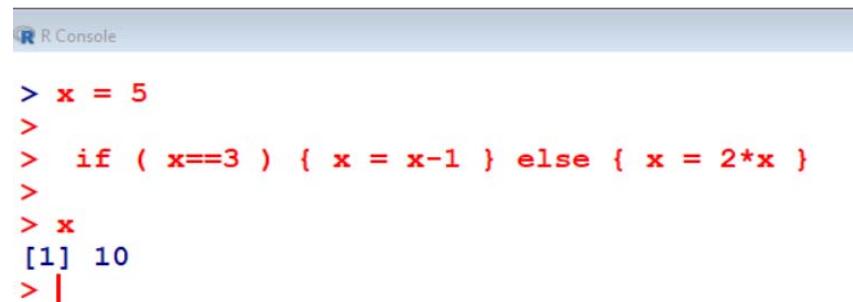
Example 4:

```
x = 5
if ( x==3 ) { x = x-1 } else { x = 2*x }
x
[1] 10
```

Interpretation:

- If  $x = 3$ , then execute  $x = x - 1$ .
- If  $x \neq 3$ , then execute  $x = 2*x$ .

In this case,  $x = 5$ , so  $x \neq 3$ . Thus  $x = 2*5$



```
R Console
> x = 5
>
> if ( x==3 ) { x = x-1 } else { x = 2*x }
>
> x
[1] 10
> |
```

## 2. Conditional execution: if else()

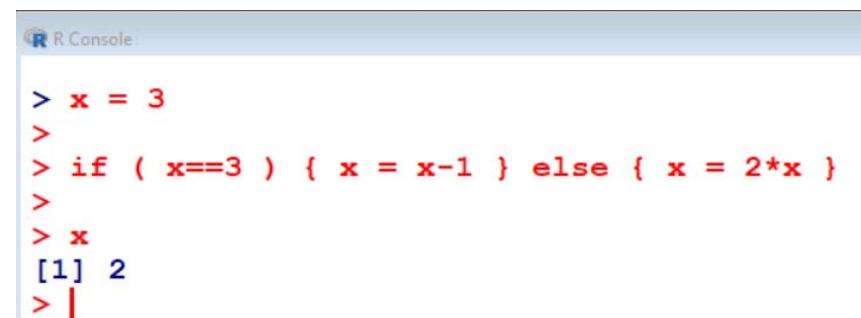
Example 5:

```
x = 3
if ( x==3 ) { x = x-1 } else { x = 2*x }
x
[1] 2
```

Interpretation:

- If  $x = 3$ , then execute  $x = x - 1$ .
- If  $x \neq 3$ , then execute  $x = 2*x$ .

In this case,  $x = 3$ . Thus  $x = 3-1 = 2$



The screenshot shows the R console interface. The title bar says "R Console". The main area contains the following R code and its output:

```
> x = 3
>
> if ( x==3 ) { x = x-1 } else { x = 2*x }
>
> x
[1] 2
> |
```

## 2. Conditional execution: if else()

### Example 6:

Suppose we want to print if a value is more than 3 or less than 3.

```
x = 6
```

```
if(x > 3){  
  
print("The value is more than 3")  
  
} else {  
  
print("The value is less than 3")  
  
}  
  
[1] "The value is more than 3"
```

```
R Console  
> x = 6  
> if(x > 3){  
+ print("The value is more than 3")  
+ } else {  
+ print("The value is less than 3")  
+ }  
[1] "The value is more than 3"  
>  
> x = 2  
> if(x > 3){  
+ print("The value is more than 3")  
+ } else {  
+ print("The value is less than 3")  
+ }  
[1] "The value is less than 3"  
> |
```

## 2. Conditional execution: if else()

### Example 7:

Suppose we want to print if a value is more than 3 or less than 3.

```
x = 2

if(x > 3){

print("The value is more than 3")

} else {

print("The value is less than 3")

}

[1] "The value is less than 3"
```

## 2. Conditional execution: if else()

### Example 6:

```
R Console

> x = 6
> if(x > 3) {
+ print("The value is more than 3")
+ } else {
+ print("The value is less than 3")
+ }
[1] "The value is more than 3"
>
> x = 2
> if(x > 3) {
+ print("The value is more than 3")
+ } else {
+ print("The value is less than 3")
+ }
[1] "The value is less than 3"
> |
```

# Foundations of R Software

## Lecture 20 Basics of Calculations

::::

### Conditional Executions – Nested if else if and ifelse

Shalabh

Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur

# **Control structures in R :**

**Control statements,**

**Functions,**

**Loops.**

# 1. Conditional execution: if()

## Syntax

```
if (condition) {execute commands if condition is TRUE}
```

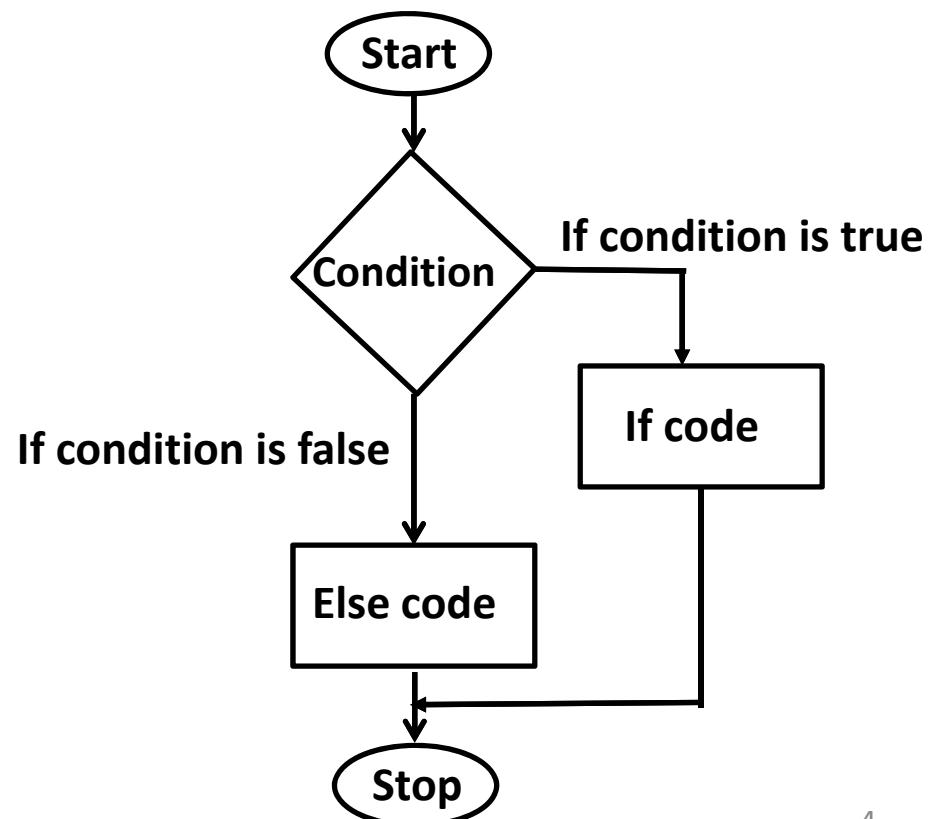
The code in the {} is evaluated if the logical test contained in the () is TRUE.

If the logical test is FALSE, R will ignore all of the code in the {}.

## 2. Conditional execution: if else()

### Syntax

```
if (condition) { executes commands if condition is TRUE}  
else { executes commands if condition is FALSE }
```



### **3. Conditional execution: Nested if else if()**

The **if...else...if** statement allows to execute a block of code when there are more than two alternatives.

It extends the earlier **if else ()** condition.

### 3. Conditional execution: Nested if else if()

#### Syntax

```
if (condition1) {  
    executes commands if condition1 is TRUE  
} else if (condition2) {  
    executes commands if condition2 is TRUE  
} else if (condition3) {  
    executes commands if condition3 is TRUE  
}  
... ...  
else {  
    executes commands if all conditions are FALSE  
}
```

### 3. Conditional execution: Nested if else if()

Example 1:

```
x = 5
if ( x==3 ) {
x = x-1
} else if ( x < 3 ) {
x = x+5
} else { x = 2*x }
x
[1] 10
```

```
R R Console
> x = 5
> if ( x==3 ) {
+ x = x-1
+ } else if ( x < 3 ) {
+ x = x+5
+ } else { x = 2*x }
> x
[1] 10
`
```

Interpretation:

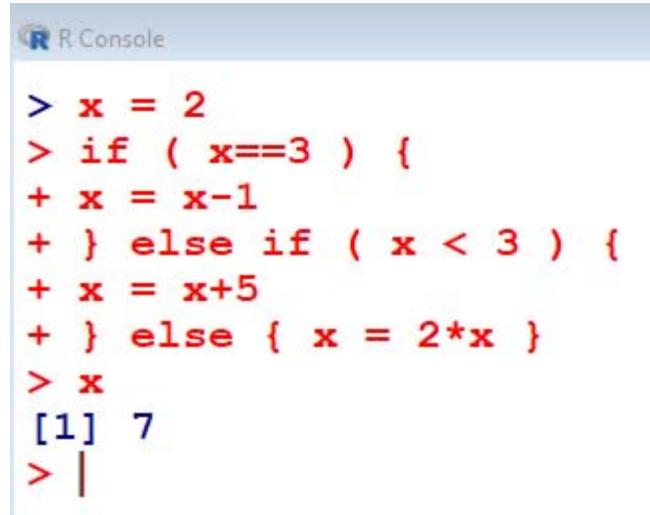
- If  $x = 3$ , then execute  $x = x - 1$ .
- If  $x < 3$ , then execute  $x = x + 5$ .
- If  $x > 3$ , then execute  $x = 2*x$ .

In this case,  $x = 5$ , so  $x > 3$ . Thus  $x = 2*5$

### 3. Conditional execution: Nested if else if()

Example 2:

```
x = 2
if ( x==3 ) {
x = x-1
} else if ( x < 3 ) {
x = x+5
} else { x = 2*x }
x
[1] 7
```



The screenshot shows the R Console window. The user has entered the R code from the left panel. The console output shows the assignment of x=2, the execution of the if-else block (which sets x=x-1), and then the execution of the else-if block (which sets x=x+5). Finally, the value of x is printed as [1] 7.

```
R Console
> x = 2
> if ( x==3 ) {
+ x = x-1
+ } else if ( x < 3 ) {
+ x = x+5
+ } else { x = 2*x }
> x
[1] 7
> |
```

Interpretation:

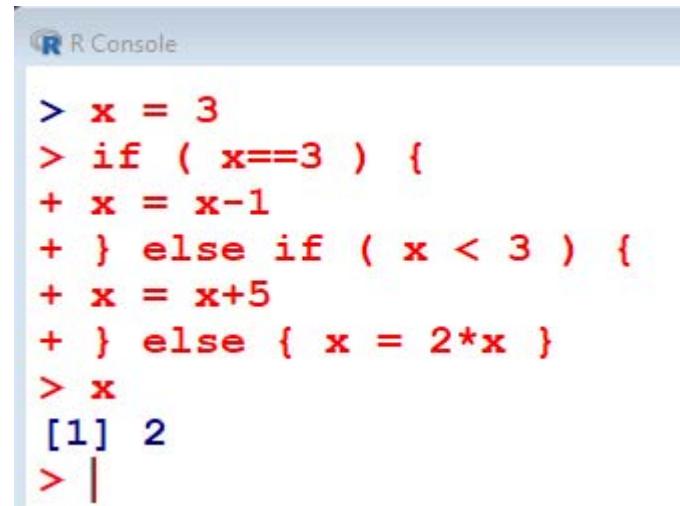
- If  $x = 3$ , then execute  $x = x - 1$ .
- If  $x < 3$ , then execute  $x = x + 5$ .
- If  $x > 3$ , then execute  $x = 2*x$ .

In this case,  $x = 2$ , so  $x < 3$ . Thus  $x = 2+5$

### 3. Conditional execution: Nested if else if()

Example 3:

```
x = 3
if ( x==3 ) {
x = x-1
} else if ( x < 3 ) {
x = x+5
} else { x = 2*x }
x
[1] 2
```



The screenshot shows an R console window titled "R Console". The code entered is:  
> x = 3  
> if ( x==3 ) {  
+ x = x-1  
+ } else if ( x < 3 ) {  
+ x = x+5  
+ } else { x = 2\*x }  
> x  
[1] 2  
> |

Interpretation:

- If  $x = 3$ , then execute  $x = x - 1$ .
- If  $x < 3$ , then execute  $x = x + 5$ .
- If  $x > 3$ , then execute  $x = 2*x$ .

In this case,  $x = 3$ . Thus  $x = 3-1$

## 4. Conditional execution: ifelse()

### Syntax

```
ifelse(test, yes, no)
```

- Vector-valued evaluation of conditions .
- For the components in the vector-valued logical expression **test** which provide the value **TRUE**, the operations given by **yes** are executed.
- For the components in the vector-valued logical expression **test** which provide the value **FALSE**, the operations given by **no** are executed.

## 4. Conditional execution: ifelse()

### Example 4:

```
> x = 1:10  
> x  
[1] 1 2 3 4 5 6 7 8 9 10  
> ifelse( x<6, x^2, x+1 )  
[1] 1 4 9 16 25 7 8 9 10 11
```

### Interpretation

- If  $x < 6$  (TRUE), then  $x = x^2$  (YES) .
  - If  $x \geq 6$  (FALSE), then  $x = x + 1$  (NO).
- 
- So for  $x = 1, 2, 3, 4, 5$ , we get  $x = x^2 = 1, 4, 9, 16, 25$
  - For  $x=6, 7, 8, 9, 10$ , we get  $x= x+1 = 7, 8, 9, 10, 11$

## 4. Conditional execution: ifelse()

R Console

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> ifelse( x<6, x^2, x+1 )
[1] 1 4 9 16 25 7 8 9 10 11
> |
```

## 4. Conditional execution: ifelse()

Example 5:

```
x = c(7,9,8,4)  
ifelse(x %% 2 == 0, "even number", "odd number")  
[1] "odd number"  "odd number"  "even number" "even number"  
[%%: Modulo Division- Finds the remainder after division of one number by  
another.]
```

Interpretation

If the remainder of x divided by 2

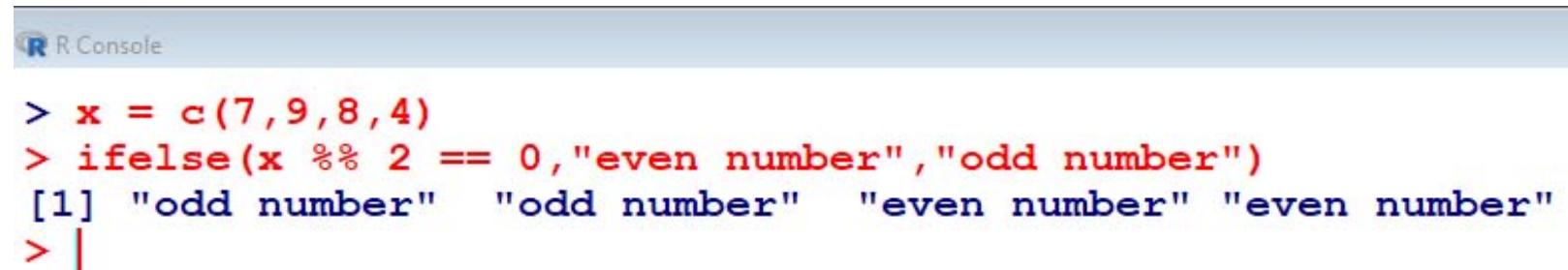
- is 0, then print "even number" (YES) .
- is not equal to 0, then print "odd number" (NO).

So for x = 7, 9, we get x = "odd number" and

for x=8, 4, we get x= "even number"

## 4. Conditional execution: ifelse()

Example 5:



A screenshot of an R console window titled "R Console". The window shows the following R code and its output:

```
> x = c(7,9,8,4)
> ifelse(x %% 2 == 0, "even number", "odd number")
[1] "odd number"  "odd number"  "even number" "even number"
> |
```

# **Foundations of R Software**

## **Lecture 21**

## **Basics of Calculations**

::::

## **Functions for Conditional Executions – switch and which commands**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# **Control structures in R :**

**Control statements,**

**Functions,**

**Loops.**

# Some functions useful in conditional execution: `switch()`

`switch` is a substitute for long `if` statements that compare a variable to several integral values.

`switch` is a multiway branch statement.

`switch` tests a variable for equality against a list of values.

`switch` map and search over a list of values.

If there are more than one matches for any given value, then `switch` returns the first matched value.

# Some functions useful in conditional execution: `switch()`

`switch` evaluates `expr` and accordingly chooses one of the further arguments (in ...).

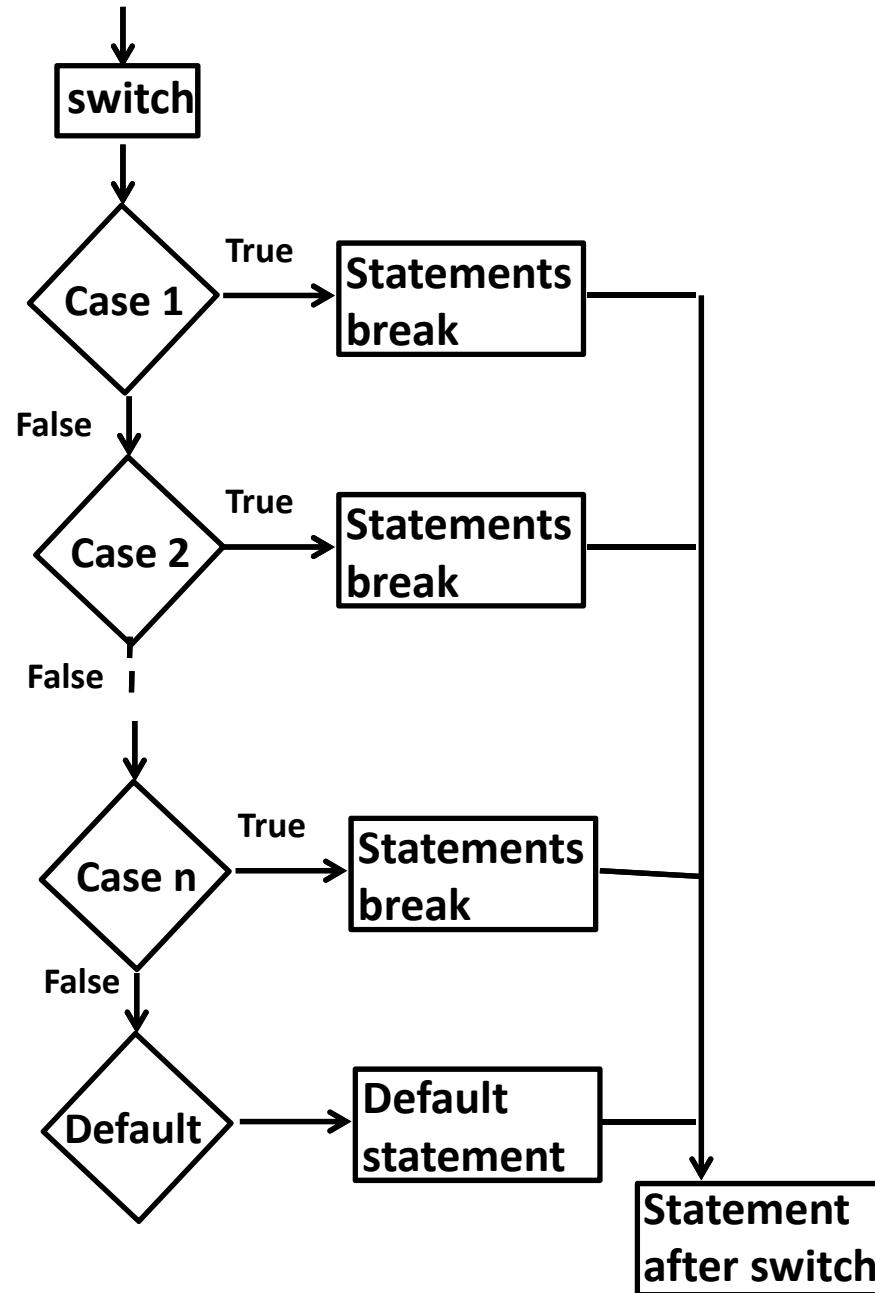
`switch(expr, ...)`

`expr` : an expression evaluating to a number or a character string.

`switch` command allows a variable to be checked for equality against a list of values or cases.

`switch(expr, case1, case2,...)`

# Some functions useful in conditional execution: switch()



# **Some functions useful in conditional execution: switch()**

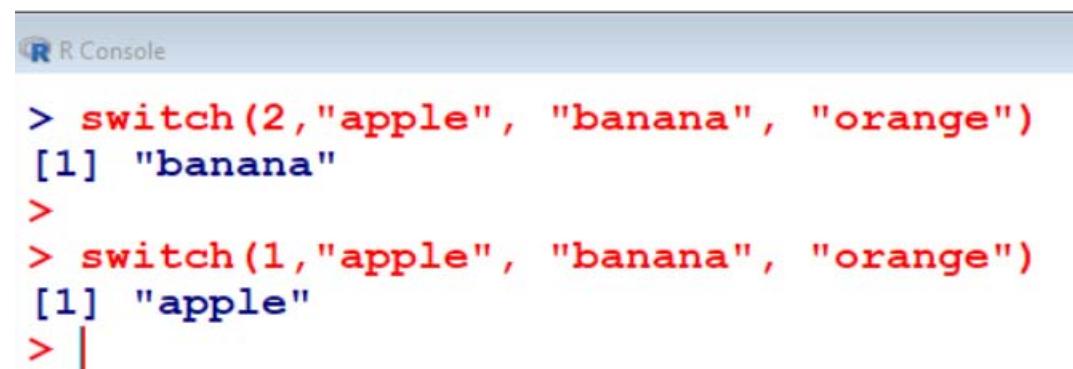
- A character string expression always matched to the listed cases.
- A non character string expression is coerced to integer.
- For multiple matches, the first match element will be used.

# Some functions useful in conditional execution: switch()

Example 1: switch () function used as an integer

```
> switch(2,"apple", "banana", "orange")  
[1] "banana"
```

```
> switch(1,"apple", "banana", "orange")  
[1] "apple"
```



The image shows a screenshot of an R console window. The title bar says "R Console". The console area contains the following text:

```
> switch(2,"apple", "banana", "orange")
[1] "banana"
>
> switch(1,"apple", "banana", "orange")
[1] "apple"
> |
```

# Some functions useful in conditional execution: switch()

**Example 2:** `switch( )` function used as a string as well. The matching named item's value is returned.

```
> switch("colour", "colour" = "blue", "gender" =  
"male", "volume" = 50)  
[1] "blue"
```

```
> switch("volume", "colour" = "blue", "gender" =  
"male", "volume" = 50)  
[1] 50
```

```
R R Console  
> switch("colour", "colour" = "blue", "gender" = "male", "volume" = 50)  
[1] "blue"  
>  
> switch("volume", "colour" = "blue", "gender" = "male", "volume" = 50)  
[1] 50  
> |
```

# Some functions useful in conditional execution: switch()

**Example 3:** In the case of no match, if there is a unnamed element of ... its value is returned.

```
> switch(4,"apple", "banana", "orange")
```

No outcome

```
> switch("size", "colour" = "blue", "gender" =  
"male", "volume" = 50)
```

No outcome

```
R Console  
> switch(4,"apple", "banana", "orange")  
> switch("size", "colour" = "blue", "gender" = "male", "volume" = 50)  
> |
```

# Some functions useful in conditional execution: which()

The `which( )` function returns the position of the elements in a logical vector which are `TRUE`.

Give the `TRUE` indices of a logical object, allowing for array indices.

`which(x, arr.ind, useNames)`

`x`: Specified input logical vector

`arr.ind`: logical, returns the array indices if `x` is an array.

`useNames`: logical, says the dimension names of an array.

# Some functions useful in conditional execution: which()

## Example 4:

```
> x = c(10,15,8,14,6,12)
> x
[1] 10 15   8 14   6 12

> which(x == 14)
[1] 4

> which(x != 12)
[1] 1 2 3 4 5

> which(x > 10)
[1] 2 4 6
```



```
R Console

> x = c(10,15,8,14,6,12)
> x
[1] 10 15   8 14   6 12
> which(x == 14)
[1] 4
> which(x != 12)
[1] 1 2 3 4 5
> which(x > 10)
[1] 2 4 6
>
```

# Some functions useful in conditional execution: which()

Example 5:

```
> x = matrix(nrow=3, ncol=3, data=1:9)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> which.min(x) #find which is the minimum value
[1] 1

> which.max(x) #find which is the maximum value
[1] 9
```

# Some functions useful in conditional execution: which()

Example 5:

```
> x = matrix(nrow=3, ncol=3, data=1:9)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> which(x %% 2 == 1)
[1] 1 3 5 7 9

> which(x %% 2 == 1, arr.ind = TRUE)
      row col
[1,]    1    1
[2,]    3    1      Gives the positions (row, columns) of the values
[3,]    2    2
[4,]    1    3
[5,]    3    3
```

# Some functions useful in conditional execution: which()

```
R Console
> x = matrix(nrow=3, ncol=3, data=1:9)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> which.min(x)
[1] 1
> which.max(x)
[1] 9
> which(x %% 2 == 1)
[1] 1 3 5 7 9
> which(x %% 2 == 1, arr.ind=TRUE)
      row col
[1,]   1   1
[2,]   3   1
[3,]   2   2
[4,]   1   3
[5,]   3   3
```

# **Foundations of R Software**

## **Lecture 22** **Basics of Calculations**

::::

### **Loops – for loop**

Shalabh

Department of Mathematics and Statistics

Indian Institute of Technology Kanpur

# **Control structures in R :**

**Control statements,**

**Functions,**

**Loops.**

# **Control structures in R :**

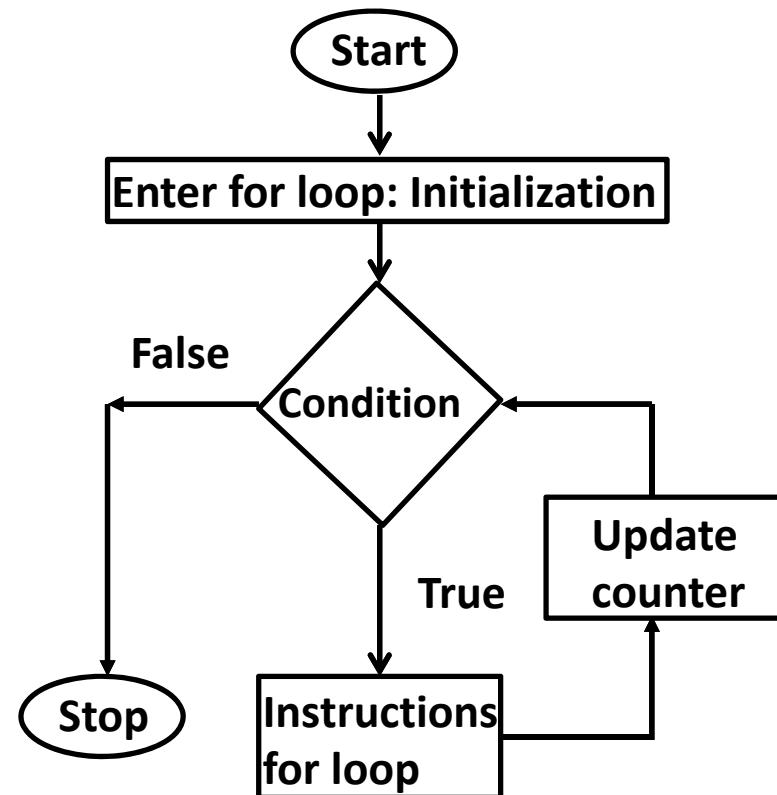
## **Loops**

**Repetitive commands are executed by loops**

- **for loop**
- **while loop**
- **repeat loop**

# 1. The `for` loop

If the number of repetitions is known in advance (e.g. if all commands have to be executed for all cases  $i = 1, 2, \dots, n$  in the data), a `for()` loop can be used.



# 1. The `for` loop

## Syntax

```
for (name in vector) {commands to be executed}
```

A variable with name `name` is sequentially set to all values, which contained in the vector `vector`.

All operations/commands are executed for all these values.

## Example 1:

```
> for ( i in 1:5 ) { print( i^2 ) }
```

```
[1] 1  
[1] 4  
[1] 9  
[1] 16  
[1] 25
```

R R Console

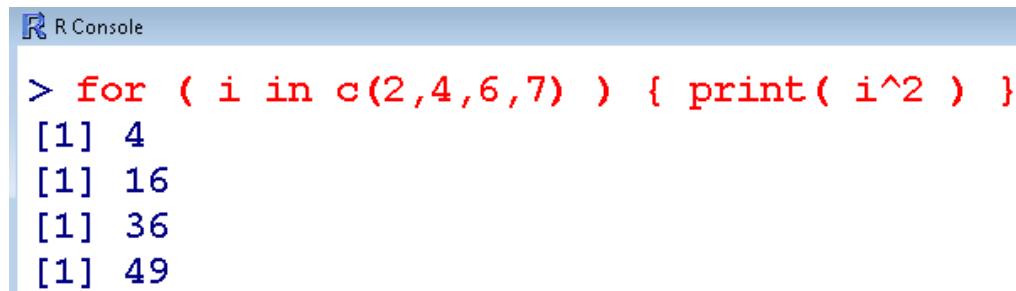
```
> for ( i in 1:5 ) { print( i^2 ) }
```

```
[1] 1  
[1] 4  
[1] 9  
[1] 16  
[1] 25
```

## Example 2:

Note: `print` is a function to print the argument

```
> for ( i in c(2,4,6,7) ) { print( i^2 ) }  
[1] 4  
[1] 16  
[1] 36  
[1] 49
```

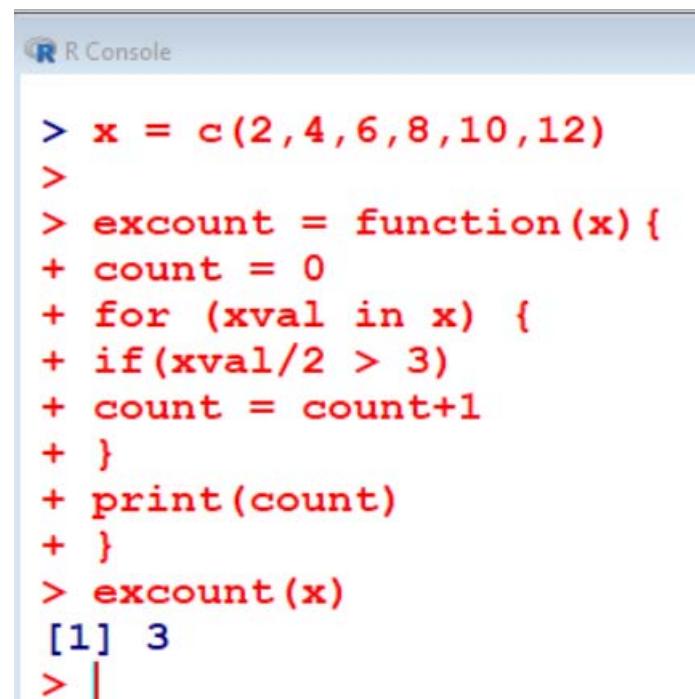


A screenshot of an R console window titled "R Console". The window shows the R command line interface with the following text:  
> for ( i in c(2,4,6,7) ) { print( i^2 ) }  
[1] 4  
[1] 16  
[1] 36  
[1] 49

## Example 3:

```
x = c(2,4,6,8,10,12)

excount = function(x){
  count = 0
  for (xval in x) {
    if(xval/2 > 3)
      count = count+1
  }
  print(count)
}
excount(x)
[1] 3
```



The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R code and its output. The code defines a vector `x` and a function `excount`. The function counts how many elements in `x` are greater than 3 when divided by 2. The output shows the function being called with `x` and the result `[1] 3`.

```
> x = c(2,4,6,8,10,12)
>
> excount = function(x) {
+   count = 0
+   for (xval in x) {
+     if(xval/2 > 3)
+       count = count+1
+   }
+   print(count)
+ }
> excount(x)
[1] 3
> |
```

# Nested looping with for loop

We can have a loop inside a loop.

**Example 4:**

```
child = c("child1", "child2", "child3")
```

```
sweet = c("sweet1", "sweet2", "sweet3")
```

```
for (x in child) {  
  for (y in sweet) {  
    print(paste(x, y))  
  }  
}
```

# Nested looping with for loop

```
> for (x in child) {  
+   for (y in sweet) {  
+     print(paste(x, y))  
+   }  
+ }  
  
[1] "child1 sweet1"  
  
[1] "child1 sweet2"  
  
[1] "child1 sweet3"  
  
[1] "child2 sweet1"  
  
[1] "child2 sweet2"  
  
[1] "child2 sweet3"  
  
[1] "child3 sweet1"  
  
[1] "child3 sweet2"  
  
[1] "child3 sweet3"
```

# Nested looping with for loop

```
R Console

> child = c("child1", "child2", "child3")
> sweet = c("sweet1", "sweet2", "sweet3")
>
> for (x in child) {
+   for (y in sweet) {
+     print(paste(x, y))
+   }
+ }
[1] "child1 sweet1"
[1] "child1 sweet2"
[1] "child1 sweet3"
[1] "child2 sweet1"
[1] "child2 sweet2"
[1] "child2 sweet3"
[1] "child3 sweet1"
[1] "child3 sweet2"
[1] "child3 sweet3"
> |
```

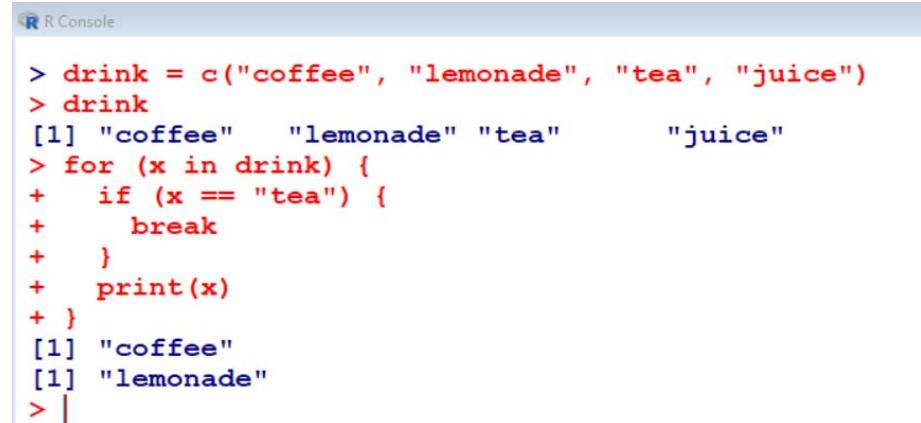
# The `break` command

Using the `break` command, we can stop the loop before it has looped through all the items.

```
drink = c("coffee", "lemonade", "tea", "juice")
```

```
for (x in drink) {  
  if (x == "tea") {  
    break  
  }  
  print(x)  
}
```

```
[1] "coffee"  
[1] "lemonade"
```



R Console

```
> drink = c("coffee", "lemonade", "tea", "juice")
> drink
[1] "coffee"   "lemonade" "tea"       "juice"
> for (x in drink) {
+   if (x == "tea") {
+     break
+   }
+   print(x)
+ }
[1] "coffee"
[1] "lemonade"
> |
```

# The `next` command

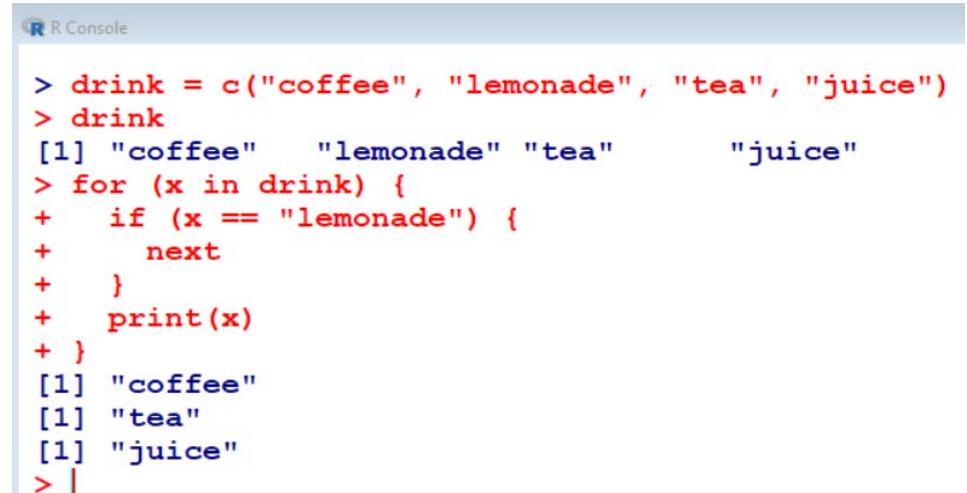
Using the `next` command, we can skip an iteration without terminating the loop.

Suppose we want to skip lemonade.

```
drink = c("coffee", "lemonade", "tea", "juice")
```

```
for (x in drink) {  
  if (x == "lemonade") {  
    next  
  }  
  print(x)  
}
```

```
[1] "coffee"  
[1] "tea"  
[1] "juice"
```



R Console

```
> drink = c("coffee", "lemonade", "tea", "juice")  
> drink  
[1] "coffee"   "lemonade" "tea"       "juice"  
> for (x in drink) {  
+   if (x == "lemonade") {  
+     next  
+   }  
+   print(x)  
+ }  
[1] "coffee"  
[1] "tea"  
[1] "juice"  
> |
```

# The next command

```
drink = c("coffee", "lemonade", "tea", "juice")
```

Suppose we want to skip tea.

```
for (x in drink) {  
  if (x == "tea") {  
    next  
  }  
  print(x)  
}
```

```
[1] "coffee"  
[1] "lemonade"  
[1] "juice"
```

```
R Console  
> drink = c("coffee", "lemonade", "tea", "juice")  
> drink  
[1] "coffee"  "lemonade" "tea"      "juice"  
> for (x in drink) {  
+   if (x == "tea") {  
+     next  
+   }  
+   print(x)  
+ }  
[1] "coffee"  
[1] "lemonade"  
[1] "juice"  
> |
```

# **Foundations of R Software**

## **Lecture 23**

## **Basics of Calculations**

::::

## **Loops – while and repeat**

Shalabh

Department of Mathematics and Statistics

Indian Institute of Technology Kanpur

# **Control structures in R :**

**Control statements,**

**Functions,**

**Loops.**

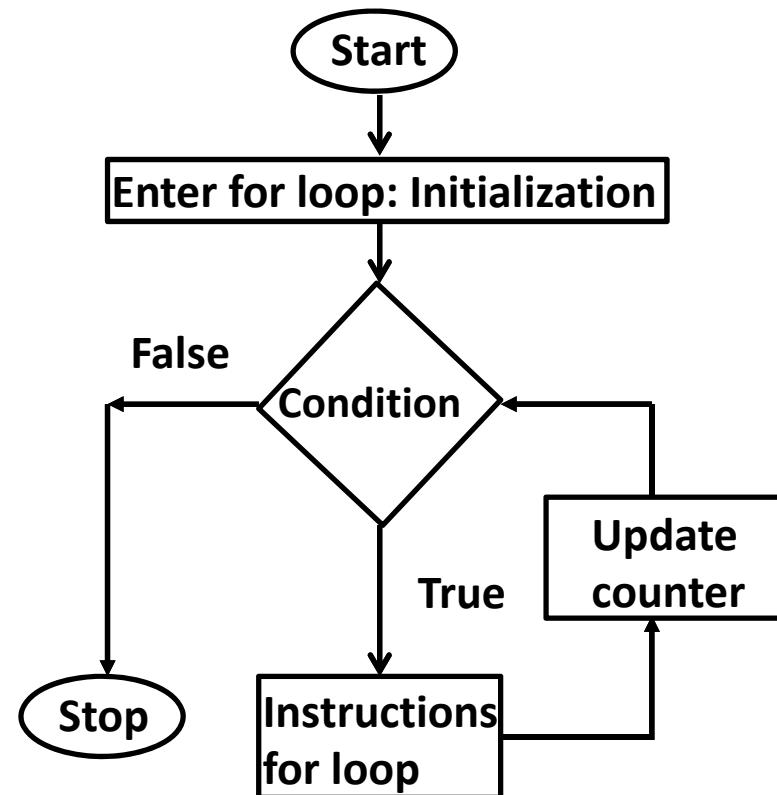
# Loops

Repetitive commands are executed by loops

- **for loop**
- **while loop**
- **repeat loop**

# 1. The `for` loop

If the number of repetitions is known in advance (e.g. if all commands have to be executed for all cases  $i = 1, 2, \dots, n$  in the data), a `for()` loop can be used.



# 1. The `for` loop

## Syntax

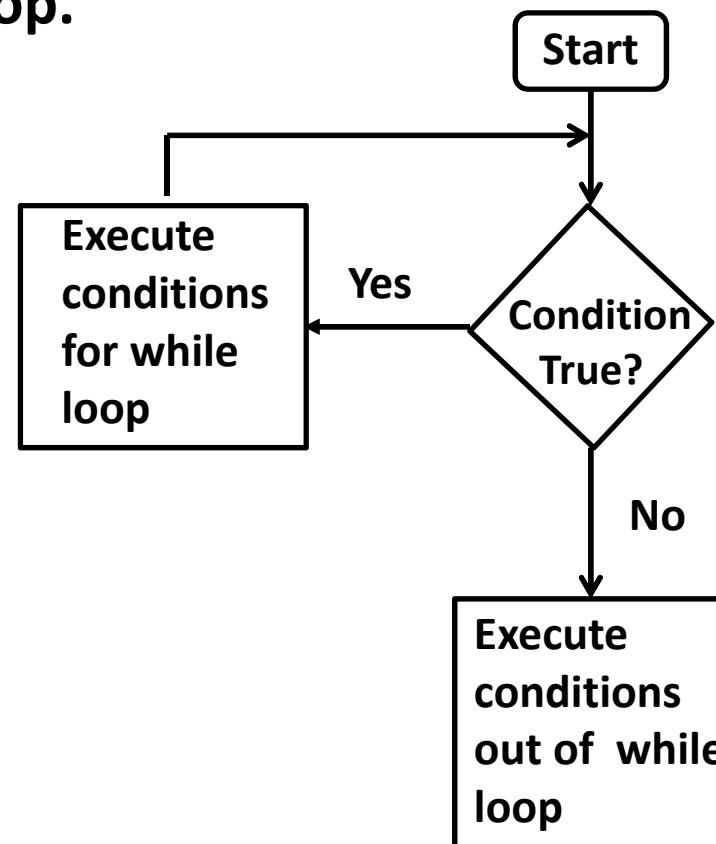
```
for (name in vector) {commands to be executed}
```

A variable with name `name` is sequentially set to all values, which contained in the vector `vector`.

All operations/commands are executed for all these values.

## 2. The `while` loop

If the number of loops is not known in before, e.g. when an iterative algorithm to maximize a likelihood function is used, one can use a `while()` loop.



## 2. The while loop

### Syntax

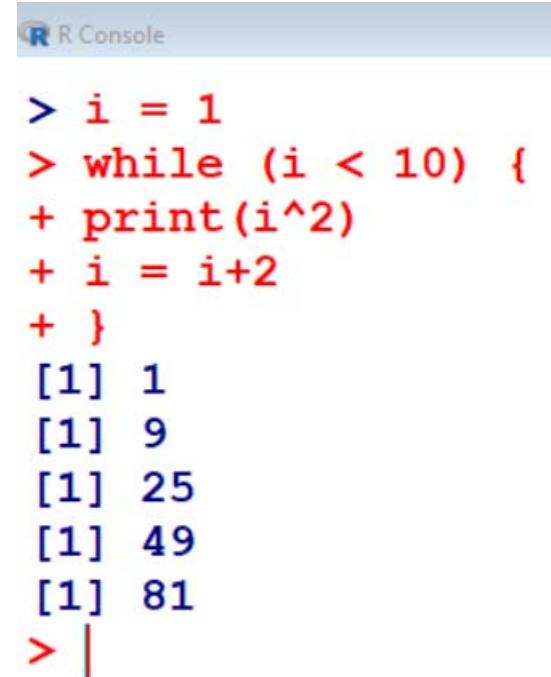
```
while(condition){ commands to be executed as  
long as condition is TRUE }
```

If the condition is not true *before entering* the loop, no commands within the loop are executed.

## 2. The while loop

### Example 1:

```
i = 1  
  
while (i < 10) {  
  
  print(i^2)  
  
  i = i+2  
  
}  
  
[1] 1  
  
[1] 9  
  
[1] 25  
  
[1] 49  
  
[1] 81
```



R Console

```
> i = 1  
> while (i < 10) {  
+ print(i^2)  
+ i = i+2  
+ }  
[1] 1  
[1] 9  
[1] 25  
[1] 49  
[1] 81  
> |
```

## 2. The while loop

Example 2:

```
sumfunction = function(){  
  sum = 0  
  
  number = as.integer(readline(prompt="Please  
select any number less than 25:  "))  
  
  while (number <= 25)  {  
    sum = sum + number  
  
    number = number + 1  }  
  
  print(paste("The sum of numbers received from  
the While Loop:  ", sum))  
}
```

## 2. The while loop

### Example 2:

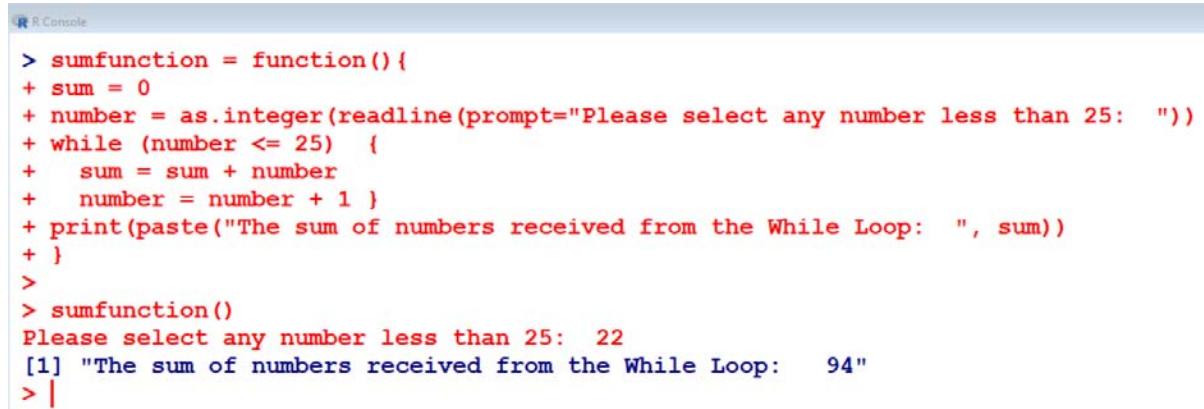
```
> sumfunction()
```

```
Please select any number less than 25 : 22
```

```
[1] "The sum of numbers received from the  
While Loop: 94"
```

```
> 22+23+24+25
```

```
[1] 94
```



The screenshot shows the R Console window with the following text:

```
R Console
> sumfunction = function(){
+ sum = 0
+ number = as.integer(readline(prompt="Please select any number less than 25: "))
+ while (number <= 25) {
+   sum = sum + number
+   number = number + 1
+ print(paste("The sum of numbers received from the While Loop: ", sum))
+ }
>
> sumfunction()
Please select any number less than 25: 22
[1] "The sum of numbers received from the While Loop: 94"
> |
```

### 3. The repeat loop

The repeat loop doesn't test any condition — in contrast to the `while()` loop — *before entering* the loop and also not during the execution of the loop.

Again, the programmer is responsible that the loop terminates after the appropriate number of iterations. For this the `break` command can be used.

#### Syntax

```
repeat{ commands to be executed }
```

## Example 3:

```
i = 1
repeat{
  print( i^2 )
  i = i+2
  if ( i > 10 )
    break
}
[1] 1
[1] 9
[1] 25
[1] 49
[1] 81
```

R Console

```
> i = 1
> repeat{
+ print( i^2 )
+ i = i+2
+ if ( i > 10 )
+ break
+
[1] 1
[1] 9
[1] 25
[1] 49
[1] 81
> |
```

## Example 4:

Additionally, the command `next` is available, to return to the beginning of the loop (to return to the first command in the loop).

```
i = 1
repeat{
  i = i+1
  if (i < 10) next
  print(i^2)
  if (i >= 13) break
}
[1] 100
[1] 121
[1] 144
[1] 169
```

```
R Console
> i = 1
> repeat{
+ i = i+1
+ if (i < 10) next
+ print(i^2)
+ if (i >= 13) break
+
[1] 100
[1] 121
[1] 144
[1] 169
> |
```

# **Foundations of R Software**

## **Lecture 24**

## **Basics of Calculations**

::::

## **Functions**

Shalabh

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Functions

- Functions are a bunch of commands grouped together in a sensible unit
- Functions take input arguments, do calculations (or make some graphics, call other functions) and produce some output and return a result in a variable. The returned variable can be a complex construct, like a list.

# **Functions : Components**

**Function Name** – This is the name of the function which is stored as an object with this name.

**Arguments** – An argument contains the input values. A function may contain no arguments.

**Function Body** – Contains statements that defines what the function has to do.

**Return Value** – The evaluated value of the last expression in the function body.

# **Functions : Built-in functions**

**R has Built-in functions also.**

**Simple examples of in-built functions are**

**sum( ), prod( ), mean( ), max( ), sum(x) etc.**

# Functions

## Syntax

```
Name <- function(Argument1, Argument2, ...)  
 {  
   expression(s)  
 }
```

where **expression(s)** is a single command or a group of commands.

You can use **=** operator also.

```
Name = function(Argument1, Argument2, ...)  
 {  
   expression(s)  
 }
```

# Function arguments with description and default values

- Function arguments can be given a meaningful name
- Function arguments can be set to default values
- Functions can have the special argument '...'

# Functions (Single variable)

The sign = or <- is furthermore used for defining functions:

```
> abc = function(x){  
  x^2  
}
```

R Console

```
> abc = function(x) {  
+   x^2  
+ }  
> abc  
function(x) {  
  x^2  
}  
> abc(3)  
[1] 9  
> abc(6)  
[1] 36  
> abc(9)  
[1] 81  
> |
```

A function is called with argument as

```
> abc(3)  
[1] 9
```

```
> abc(6)  
[1] 36
```

```
> abc(9)  
[1] 81
```

# Functions (Two variables)

```
> abc = function(x,y){  
+   x^2+y^2  
+ }
```

```
> abc(3,4)  
[1] 25
```

```
> abc(10, 10)  
[1] 200
```

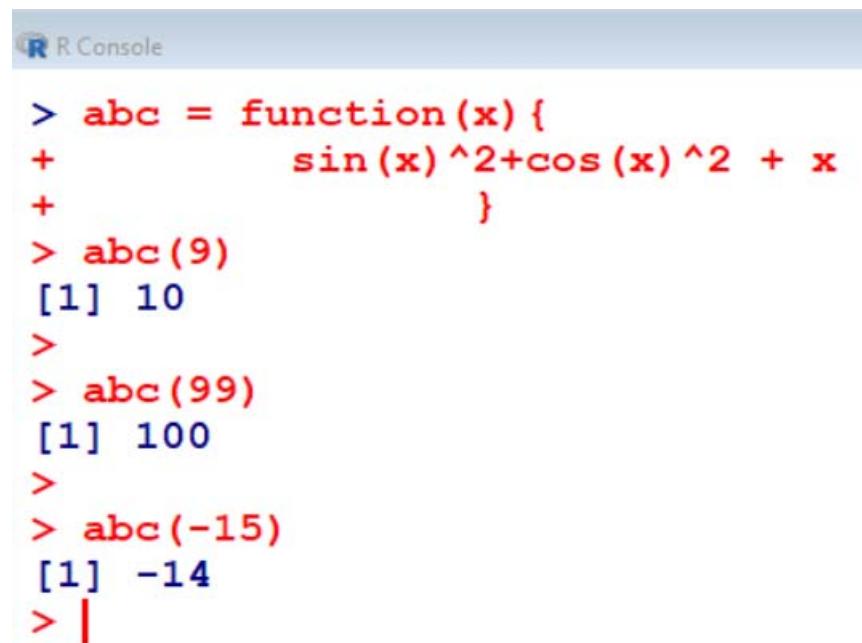
```
> abc(-2, -3)  
[1] 13
```

R Console

```
> abc = function(x,y) {  
+   x^2+y^2  
+ }  
> abc(3,4)  
[1] 25  
>  
> abc(10, 10)  
[1] 200  
>  
> abc(-2, -3)  
[1] 13  
> |
```

# Functions- Another example

```
> abc = function(x){  
  sin(x)^2+cos(x)^2 + x  
}  
  
> abc(9)  
[1] 10  
  
> abc(99)  
[1] 100  
  
> abc(-15)  
[1] -14
```



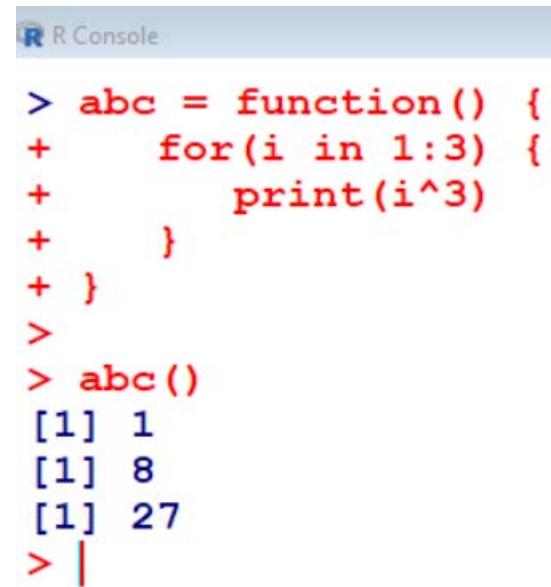
R Console

```
> abc = function(x) {  
+   sin(x)^2+cos(x)^2 + x  
+ }  
> abc(9)  
[1] 10  
>  
> abc(99)  
[1] 100  
>  
> abc(-15)  
[1] -14  
> |
```

# Functions- Another example

## Calling a function without an Argument

```
abc = function() {  
  for(i in 1:3) {  
    print(i^3)  
  }  
}  
  
> abc()  
[1] 1  
[1] 8  
[1] 27
```



R Console

```
> abc = function() {  
+   for(i in 1:3) {  
+     print(i^3)  
+   }  
+ }  
>  
> abc()  
[1] 1  
[1] 8  
[1] 27  
> |
```

# **Foundations of R Software**

## **Lecture 25**

### **Sequences**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Sequences

A sequence is a set of related numbers, events, movements, or items that follow each other in a particular order.

The regular sequences can be generated in R.

Syntax

`seq()`

```
seq(from = 1, to = 1,  
by = ((to - from)/(length.out - 1)),  
length.out = NULL, along.with = NULL, ...)
```

Help for `seq`

```
> help("seq")
```

# Sequences

127.0.0.1:13069/library/base/html/seq.html

seq {base}

R Documentation

## Sequence Generation

### Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

### Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

### Arguments

... arguments passed to or from methods.

from, to the starting and (maximal) end values of the sequence. Of length 1 unless just `from` is supplied as an unnamed argument.

•  
•  
•  
•

# Sequences

- The default increment is +1 or -1

```
> seq(from=2, to=4)  
[1] 2 3 4
```

```
> seq(from=4, to=2)  
[1] 4 3 2
```

```
> seq(from=-4, to=4)  
[1] -4 -3 -2 -1 0 1 2 3 4
```

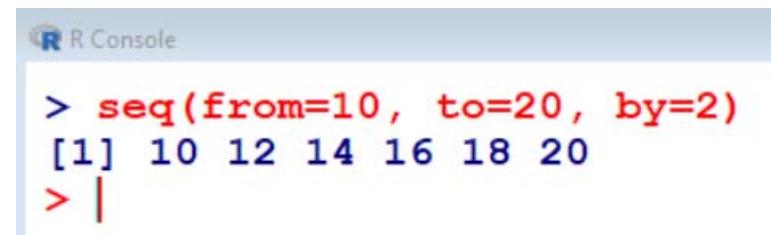
```
R Console  
> seq(from=2, to=4)  
[1] 2 3 4  
>  
> seq(from=4, to=2)  
[1] 4 3 2  
>  
> seq(from=-4, to=4)  
[1] -4 -3 -2 -1 0 1 2 3 4
```

# Sequences

## □ Sequence with constant increment:

Generate a sequence from 10 to 20 with an increment of 2 units

```
> seq(from=10, to=20, by=2)  
[1] 10 12 14 16 18 20
```



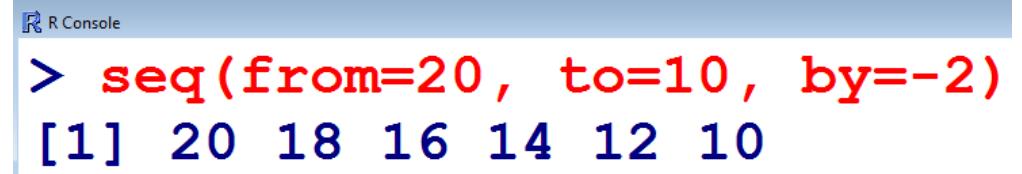
A screenshot of an R console window titled "R Console". The window shows the command `> seq(from=10, to=20, by=2)` in red, followed by its output [1] 10 12 14 16 18 20 in blue, and a cursor character > in red.

# Sequences

## □ Sequence with constant decrement:

Generate a sequence from 20 to 10 with an decrement of 2 units

```
> seq(from=20, to=10, by=-2)  
[1] 20 18 16 14 12 10
```



R Console

```
> seq(from=20, to=10, by=-2)  
[1] 20 18 16 14 12 10
```

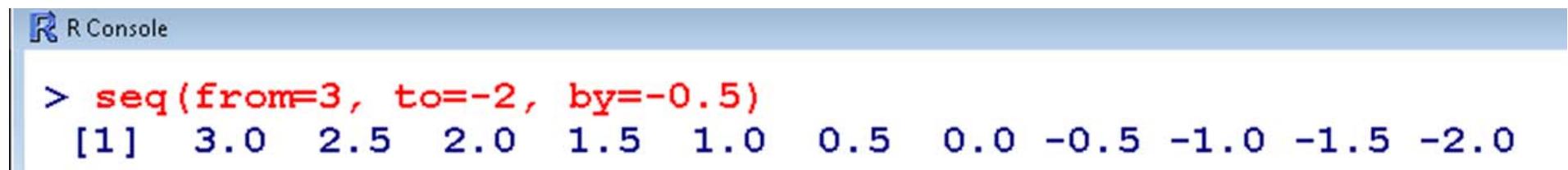
A screenshot of an R console window. The title bar says "R Console". The main area contains a command in red text: "> seq(from=20, to=10, by=-2)". Below it is the output in blue text: "[1] 20 18 16 14 12 10".

# Sequences

- ☐ Downstream sequence with constant decrement:

Generate a sequence from 3 to -2 with a decrement of 0.5 units

```
> seq(from=3, to=-2, by=-0.5)
[1] 3.0 2.5 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2
```



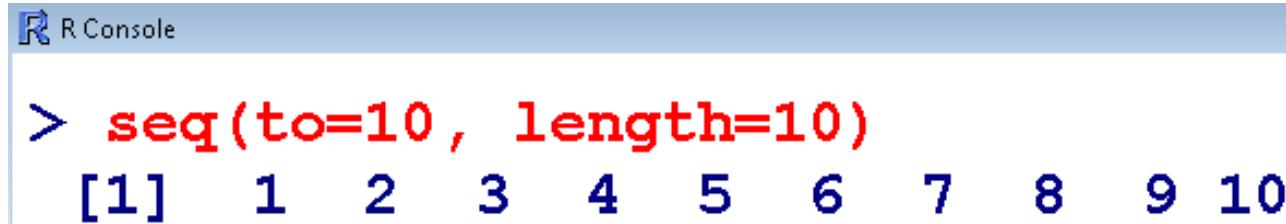
The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R command `> seq(from=3, to=-2, by=-0.5)` and its output [1] 3.0 2.5 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2.0.

```
R Console
> seq(from=3, to=-2, by=-0.5)
[1] 3.0 2.5 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2.0
```

# Sequences

- Sequences with a predefined length with default increment +1

```
> seq(to=10, length=10)  
[1] 1 2 3 4 5 6 7 8 9 10
```



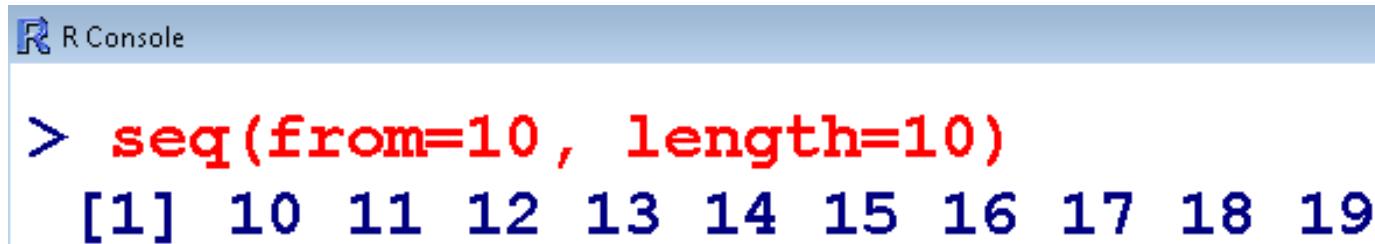
R Console

```
> seq(to=10, length=10)  
[1] 1 2 3 4 5 6 7 8 9 10
```

# Sequences

- Sequences with a predefined length with default increment +1

```
> seq(from=10, length=10)  
[1] 10 11 12 13 14 15 16 17 18 19
```



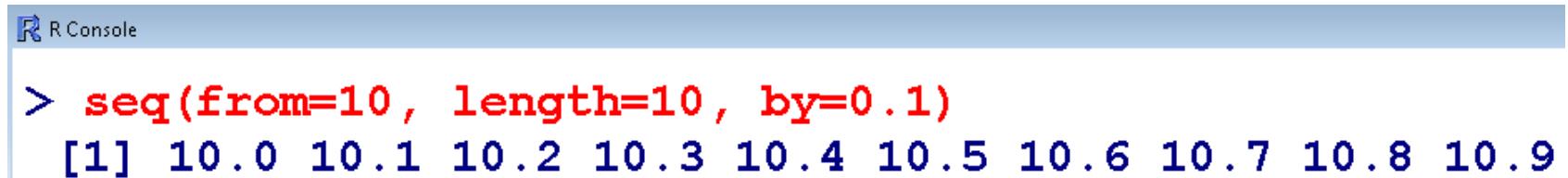
A screenshot of an R console window titled "R Console". The window shows the command `> seq(from=10, length=10)` in red, indicating it is a user input, followed by the output [1] 10 11 12 13 14 15 16 17 18 19 in blue.

```
> seq(from=10, length=10)  
[1] 10 11 12 13 14 15 16 17 18 19
```

# Sequences

- Sequences with a predefined length with constant fractional increment

```
> seq(from=10, length=10, by=0.1)  
[1] 10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9
```



A screenshot of an R console window titled "R Console". The window shows the command `> seq(from=10, length=10, by=0.1)` entered at the prompt, followed by the output [1] 10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9.

```
R Console  
> seq(from=10, length=10, by=0.1)  
[1] 10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9
```

# Sequences

- Sequences with a predefined length with constant decrement

```
> seq(from=10, length=10, by=-2)  
[1] 10  8  6  4  2  0 -2 -4 -6 -8
```

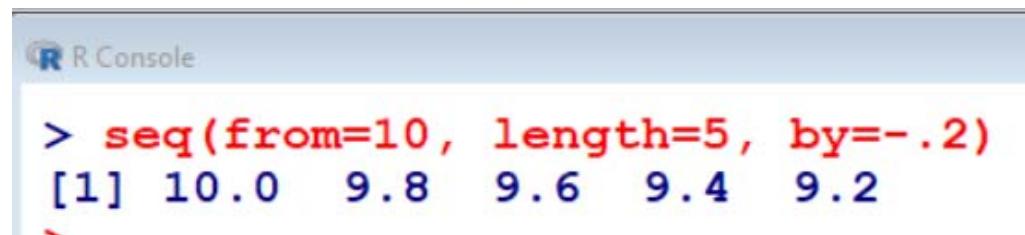


```
R Console  
> seq(from=10, length=10, by=-2)  
[1] 10  8  6  4  2  0 -2 -4 -6 -8  
> |
```

# Sequences

- Sequences with a predefined length with constant fractional decrement

```
> seq(from=10, length=5, by=-.2)  
[1] 10.0  9.8  9.6  9.4  9.2
```



A screenshot of an R console window titled "R Console". The window shows the command `> seq(from=10, length=5, by=-.2)` in red, indicating it is a user input. Below it, the output is shown in blue: `[1] 10.0 9.8 9.6 9.4 9.2`. A red cursor arrow points to the first digit of the first number, "1".

# **Foundations of R Software**

## **Lecture 26**

### **More Sequences and Other Operations**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Sequences

A sequence is a set of related numbers, events, movements, or items that follow each other in a particular order.

The regular sequences can be generated in R.

Syntax

`seq()`

```
seq(from = 1, to = 1,  
by = ((to - from)/(length.out - 1)),  
length.out = NULL, along.with = NULL, ...)
```

Help for `seq`

# Sequences

□ Continuous sequences with constant unit increment and decrement

```
> 1:10
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> 10:1
```

```
[1] 10 9 8 7 6 5 4 3 2 1
```

```
> 5:15
```

```
[1] 5 6 7 8 9 10 11 12 13 14 15
```

```
> 15:5
```

```
[1] 15 14 13 12 11 10 9 8 7 6 5
```

# Sequences

□ Continuous sequences with constant unit increment and decrement

```
> -1:-10
```

```
[1] -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

```
> -10:-1
```

```
[1] -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
```

```
> -5:-15
```

```
[1] -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15
```

```
> -15:-5
```

```
[1] -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5
```

# Sequences

```
R Console
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> 10:1
[1] 10 9 8 7 6 5 4 3 2 1
> 5:15
[1] 5 6 7 8 9 10 11 12 13 14 15
> 15:5
[1] 15 14 13 12 11 10 9 8 7 6 5
>
> -1:-10
[1] -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
> -10:-1
[1] -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
> -5:-15
[1] -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15
> -15:-5
[1] -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5
> |
```

# Sequences

□ Continuous sequences with constant unit increment and decrement

```
> 1.23:10
```

```
[1] 1.23 2.23 3.23 4.23 5.23 6.23 7.23 8.23 9.23
```

```
> 1.23:10.54
```

```
[1] 1.23 2.23 3.23 4.23 5.23 6.23 7.23 8.23 9.23
```

```
10.23
```

```
> 10.54:2.23
```

```
[1] 10.54 9.54 8.54 7.54 6.54 5.54 4.54
```

```
3.54 2.54
```

# Sequences

- Continuous sequences with constant unit increment and decrement

```
> -1.23:-10
```

```
[1] -1.23 -2.23 -3.23 -4.23 -5.23 -6.23 -7.23 -  
8.23 -9.23
```

```
> -5.23:6
```

```
[1] -5.23 -4.23 -3.23 -2.23 -1.23 -0.23  0.77  
1.77  2.77  3.77  4.77  5.77
```

# Sequences

```
R Console

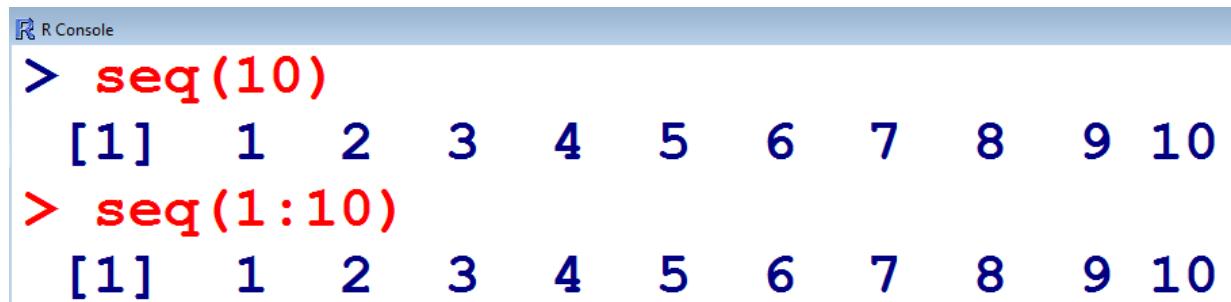
> 1.23:10
[1] 1.23 2.23 3.23 4.23 5.23 6.23 7.23 8.23 9.23
> 1.23:10.54
[1] 1.23 2.23 3.23 4.23 5.23 6.23 7.23 8.23 9.23 10.23
>
> 10.54:2.23
[1] 10.54 9.54 8.54 7.54 6.54 5.54 4.54 3.54 2.54
>
> -1.23:-10
[1] -1.23 -2.23 -3.23 -4.23 -5.23 -6.23 -7.23 -8.23 -9.23
>
> -5.23:6
[1] -5.23 -4.23 -3.23 -2.23 -1.23 -0.23  0.77  1.77  2.77  3.77  4.77  5.77
> |
```

# Sequences

```
> seq(10)  
[1] 1 2 3 4 5 6 7 8 9 10
```

is the same as

```
> seq(1:10)  
[1] 1 2 3 4 5 6 7 8 9 10
```



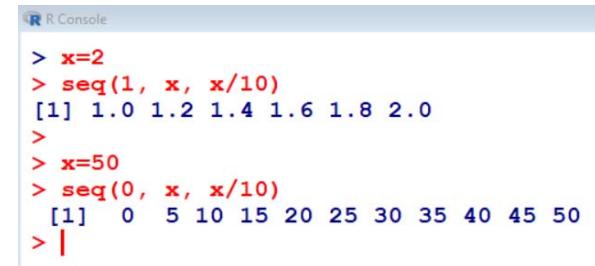
A screenshot of an R console window titled "R Console". It displays two identical sequences of numbers from 1 to 10. The first line shows the command "seq(10)" followed by the output "[1] 1 2 3 4 5 6 7 8 9 10". The second line shows the command "seq(1:10)" followed by the same output. Both lines are highlighted in red.

```
> seq(10)  
[1] 1 2 3 4 5 6 7 8 9 10  
> seq(1:10)  
[1] 1 2 3 4 5 6 7 8 9 10
```

# Sequences

Sequences with a predefined variable and constant increment

```
> x=2  
  
> seq(1, x, x/10)  
[1] 1.0 1.2 1.4 1.6 1.8 2.0
```



R Console output:

```
> x=2  
> seq(1, x, x/10)  
[1] 1.0 1.2 1.4 1.6 1.8 2.0  
>  
> x=50  
> seq(0, x, x/10)  
[1] 0 5 10 15 20 25 30 35 40 45 50  
>
```

```
> x=50  
  
> seq(0, x, x/10)  
[1] 0 5 10 15 20 25 30 35 40 45 50
```

# Sequences

Outcome of sequences can be stored.

```
x = seq(1, 50, 1/2)
```

**x**

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0  
[18] 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0 17.5  
[35] 18.0 18.5 19.0 19.5 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5 26.0  
[52] 26.5 27.0 27.5 28.0 28.5 29.0 29.5 30.0 30.5 31.0 31.5 32.0 32.5 33.0 33.5 34.0 34.5  
[69] 35.0 35.5 36.0 36.5 37.0 37.5 38.0 38.5 39.0 39.5 40.0 40.5 41.0 41.5 42.0 42.5 43.0  
[86] 43.5 44.0 44.5 45.0 45.5 46.0 46.5 47.0 47.5 48.0 48.5 49.0 49.5 50.0
```

**y=2\*x**

**y**

```
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22  
[22] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43  
[43] 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64  
[64] 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85  
[85] 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

# Sequences

Outcome of sequences can be stored.

```
R Console
> x = seq(1, 50, 1/2)
> x
[1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0
[18]  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0 17.5
[35] 18.0 18.5 19.0 19.5 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5 26.0
[52] 26.5 27.0 27.5 28.0 28.5 29.0 29.5 30.0 30.5 31.0 31.5 32.0 32.5 33.0 33.5 34.0 34.5
[69] 35.0 35.5 36.0 36.5 37.0 37.5 38.0 38.5 39.0 39.5 40.0 40.5 41.0 41.5 42.0 42.5 43.0
[86] 43.5 44.0 44.5 45.0 45.5 46.0 46.5 47.0 47.5 48.0 48.5 49.0 49.5 50.0
>
> y=2*x
> y
[1]   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22
[22]  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43
[43]  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
[64]  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
[85]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
> |
```

## Sequences: Index vector

### □ Assignment of an index-vector

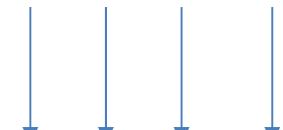
```
x = c(9,8,7,6)
```

```
ind = seq(along=x)
```

```
ind
```

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

9,8,7,6



1 2 3 4

## Accessing a value in the vector through index vector

### □ Accessing an element of an index-vector

```
> x[ ind[2] ]
```

```
[1] 8
```

```
R Console
> x = c(9,8,7,6)
> ind = seq(along=x)
> ind
[1] 1 2 3 4
>
> x[ ind[2] ]
[1] 8
> |
```

# Sequences

Generating sequences of dates

Generating current time and date

`Sys.time()` command provides the current time and date from the computer system.

```
> Sys.time()  
[1] "2021-11-29 21:23:57 IST"
```

`Sys.Date()` command provides the current date from the computer system.

```
> Sys.Date()  
[1] "2021-11-29"
```

```
R Console  
> Sys.time()  
[1] "2021-11-29 21:23:57 IST"  
> Sys.Date()  
[1] "2021-11-29"  
> |
```

# **Foundations of R Software**

**Lecture 27**

## **Sequences of Dates and Alphabets**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Sequences

The regular sequences can be generated in R.

## Syntax

`seq()`

```
seq(from = 1, to = 1,  
by = ((to - from)/(length.out - 1)),  
length.out = NULL, along.with = NULL, ...)
```

# Sequences

## Generating sequences of dates

### Usage

```
seq(from, to, by, length.out = NULL, along.with  
= NULL, ...)
```

### Arguments

**from** starting date (Required)

**to** end date (Optional)

**by** increment of the sequence. "day", "week",  
"month", "quarter" or "year".

**length.out** integer, optional. Desired length of the sequence.

**along.with** take the length from the length of this argument. 3

# Sequences

## Generating sequences of dates

### Sequence of first day of years

```
> seq(as.Date("2010-01-01"), as.Date("2017-01-01"), by = "years")  
[1] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01"  
[5] "2014-01-01" "2015-01-01" "2016-01-01" "2017-01-01"
```

R Console

```
> seq(as.Date("2010-01-01"),  
+ as.Date("2017-01-01"), by = "years")  
[1] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01"  
[5] "2014-01-01" "2015-01-01" "2016-01-01" "2017-01-01"
```

# Sequences

## Generating sequences of dates

### Sequence of days

```
> seq(as.Date("2017-01-01"), by = "days",
length = 6)
[1] "2017-01-01" "2017-01-02" "2017-01-03" "2017-01-04"
[5] "2017-01-05" "2017-01-06"
```

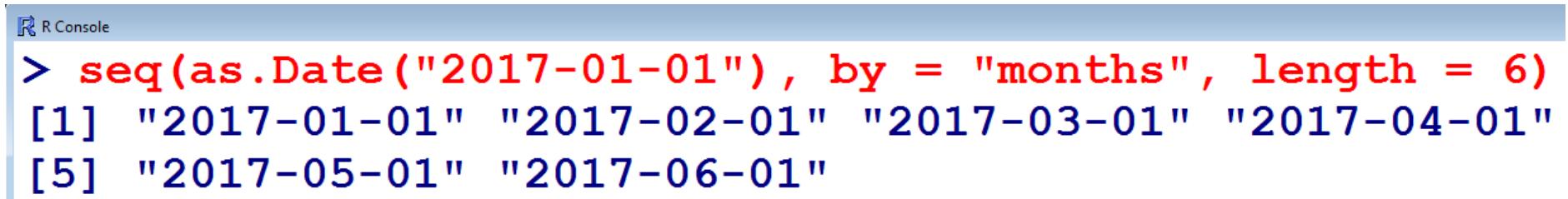
```
R Console
> seq(as.Date("2017-01-01"), by = "days", length = 6)
[1] "2017-01-01" "2017-01-02" "2017-01-03" "2017-01-04"
[5] "2017-01-05" "2017-01-06"
```

# Sequences

## Generating sequences of dates

### Sequence of months

```
> seq(as.Date("2017-01-01"), by = "months",  
length = 6)  
[1] "2017-01-01" "2017-02-01" "2017-03-01" "2017-04-01"  
[5] "2017-05-01" "2017-06-01"
```



R Console

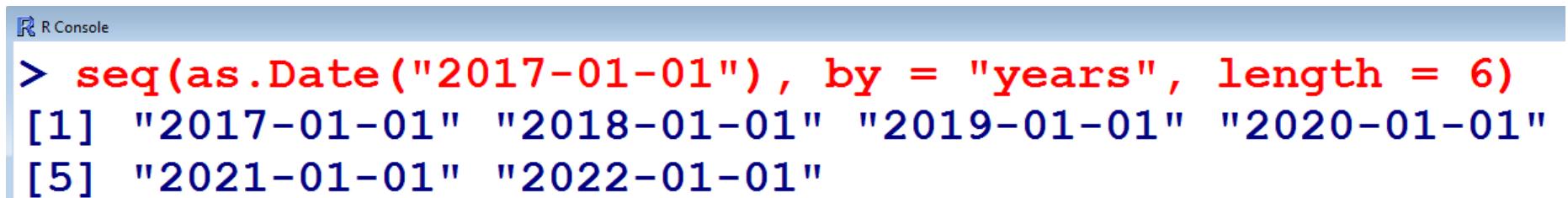
```
> seq(as.Date("2017-01-01"), by = "months", length = 6)  
[1] "2017-01-01" "2017-02-01" "2017-03-01" "2017-04-01"  
[5] "2017-05-01" "2017-06-01"
```

# Sequences

## Generating sequences of dates

### Sequence by years

```
> seq(as.Date("2017-01-01"), by = "years",  
length = 6)  
[1] "2017-01-01" "2018-01-01" "2019-01-01" "2020-01-01"  
[5] "2021-01-01" "2022-01-01"
```



R Console

```
> seq(as.Date("2017-01-01"), by = "years", length = 6)  
[1] "2017-01-01" "2018-01-01" "2019-01-01" "2020-01-01"  
[5] "2021-01-01" "2022-01-01"
```

# Sequences

## Generating sequences of dates

To find sequence with defining start and end dates

```
> startdate = as.Date("2016-1-1")
> enddate = as.Date("2017-1-1")

> out = seq(enddate, startdate, by = "-1 month")
[1] "2017-01-01" "2016-12-01" "2016-11-01" "2016-10-01"
[5] "2016-09-01" "2016-08-01" "2016-07-01" "2016-06-01"
[9] "2016-05-01" "2016-04-01" "2016-03-01" "2016-02-01"
[13] "2016-01-01"
```

## Sequences

### Generating sequences of dates

R Console

```
> startdate = as.Date("2016-1-1")
> enddate = as.Date("2017-1-1")
>
> out = seq(enddate, startdate, by = "-1 month")
> out
[1] "2017-01-01" "2016-12-01" "2016-11-01"
[4] "2016-10-01" "2016-09-01" "2016-08-01"
[7] "2016-07-01" "2016-06-01" "2016-05-01"
[10] "2016-04-01" "2016-03-01" "2016-02-01"
[13] "2016-01-01"
> |
```

# Sequences

## Generating sequences of alphabets

**letters** is used to find sequence of lowercase alphabets

```
> letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"  
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
R Console  
> letters  
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"  
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

# Sequences

## Generating sequences of alphabets

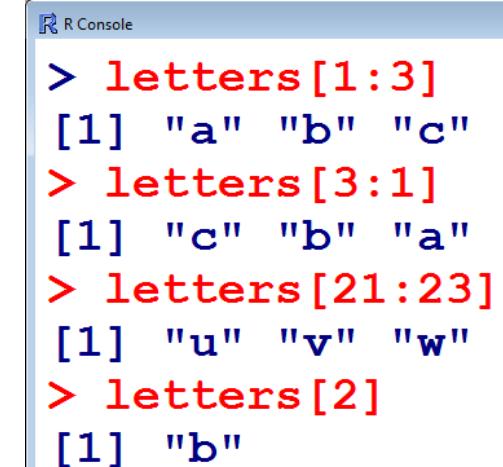
`letters[from_index:to_index]` is used to find sequence of lowercase alphabets from a particular index to a specified index.

```
> letters[1:3]
[1] "a" "b" "c"

> letters[3:1]
[1] "c" "b" "a"

> letters[21:23]
[1] "u" "v" "w"

> letters[2]
[1] "b"
```



R Console

```
> letters[1:3]
[1] "a" "b" "c"
> letters[3:1]
[1] "c" "b" "a"
> letters[21:23]
[1] "u" "v" "w"
> letters[2]
[1] "b"
```

# Sequences

## Generating sequences of alphabets

**LETTERS** is used to find sequence of uppercase alphabets

```
> LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"  
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

R Console

```
> LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"  
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

# Sequences

## Generating sequences of alphabets

`LETTERS[from_index:to_index]` is used to find sequence of uppercase alphabets from a particular index to a specified index.

```
> LETTERS[1:3]
[1] "A" "B" "C"

> LETTERS[3:1]
[1] "C" "B" "A"

> LETTERS[21:23]
[1] "U" "V" "W"

> LETTERS[2]
[1] "B"
```

```
R Console
> LETTERS[1:3]
[1] "A" "B" "C"
> LETTERS[3:1]
[1] "C" "B" "A"
> LETTERS[21:23]
[1] "U" "V" "W"
> LETTERS[2]
[1] "B"
```

# **Foundations of R Software**

**Lecture 28**

**Repeats**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Repeats

The `rep()` function replicates numeric values, or text, or the values of a vector for a specific number of times.

Command `rep` is used to replicates the values in a vector.

Syntax `rep(x)` replicates the values in a vector `x`.

`rep(x, times=n) # Repeat x as a whole n times`

`rep(x, each=n) # Repeat each cell n times`

# Repeats

Following commands repeat each cell for the desired length of the output vector

`rep(x, length.out=n)`

`rep(x, length=n)`

`rep_len(x, length.out)`

# Repeats

## Help for the command `rep`

```
> help("rep")
```

The screenshot shows a web browser window with the URL 127.0.0.1:13069/library/base/html/rep.html. The page title is "Replicate Elements of Vectors and Lists". The content includes sections for "Description", "Usage", and "Arguments".

**Description**

`rep` replicates the values in `x`. It is a generic function, and the (internal) default method is described here.

`rep.int` and `rep_len` are faster simplified versions for two common cases. They are not generic.

**Usage**

```
rep(x, ...)  
rep.int(x, times)  
rep_len(x, length.out)
```

**Arguments**

- `x` a vector (of any mode including a list) or a factor or (for `rep` only) a `POSIXct` or `POSIXlt` or `Date` object; or an S4 object containing such an object.
- `...` further arguments to be passed to or from other methods. For the internal default method these can include:

`times`

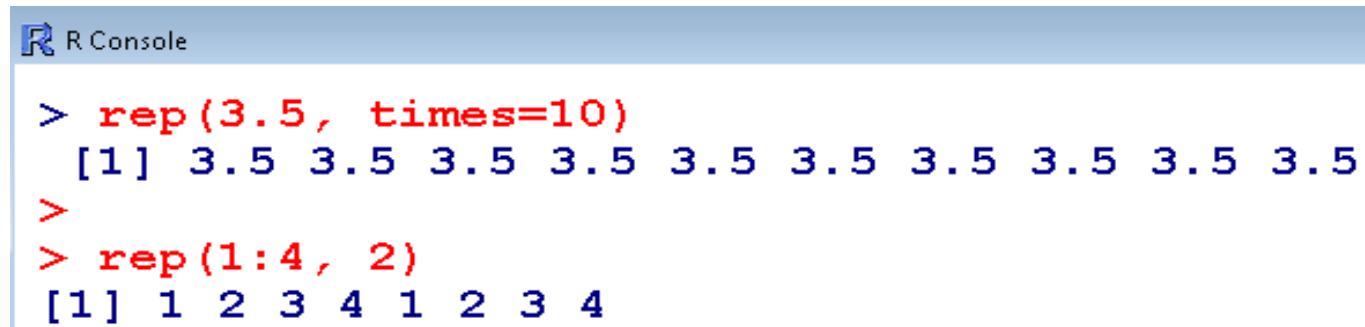
# Repeats

The command `rep`

Repeat an object  $n$ -times:

```
> rep(3.5, times=10)
[1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5
```

```
> rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4
```



A screenshot of the R console window. The title bar says "R Console". The console area contains two code snippets. The first snippet shows the command `rep(3.5, times=10)` followed by its output [1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5. The second snippet shows the command `rep(1:4, 2)` followed by its output [1] 1 2 3 4 1 2 3 4.

```
> rep(3.5, times=10)
[1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5
>
> rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4
```

# Repeats

Repeat an object  $n$ -times:

```
rep(x, times = n)
```

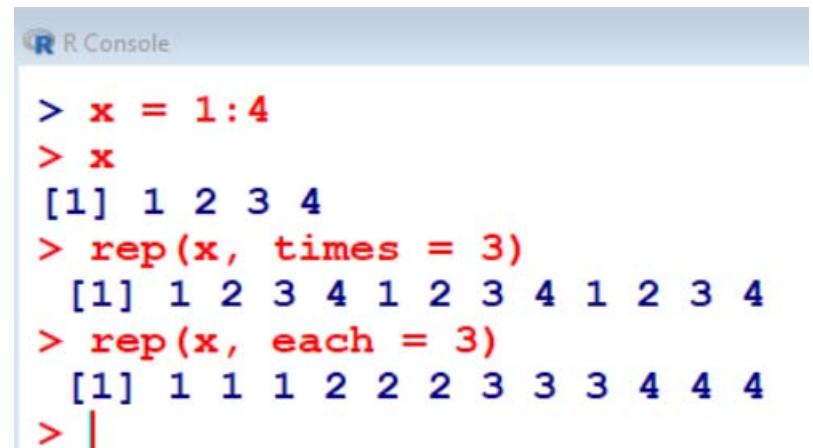
Repeat each cell  $n$ -times:

```
rep(x, each = n)
```

```
> x = 1:4  
> x  
[1] 1 2 3 4
```

```
> rep(x, times = 3)  
[1] 1 2 3 4 1 2 3 4 1 2 3 4
```

```
> rep(x, each = 3)  
[1] 1 1 1 2 2 2 3 3 3 4 4 4
```



The screenshot shows the R console interface. The title bar says "R Console". The command line shows several R commands being run in red, and their corresponding outputs in blue. The outputs are the results of the rep() function calls.

```
R Console  
> x = 1:4  
> x  
[1] 1 2 3 4  
> rep(x, times = 3)  
[1] 1 2 3 4 1 2 3 4 1 2 3 4  
> rep(x, each = 3)  
[1] 1 1 1 2 2 2 3 3 3 4 4 4  
> |
```

# Repeats

Every object is repeated several times successively:

```
> rep(1:4, each = 2)
```

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

```
> rep(1:4, each = 2, times = 3)
```

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

```
> rep(1:4, times = 3, each = 2)
```

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

Observe the effect of ordering of `each` and `times`.

# Repeats

```
R Console

> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
> rep(1:4, each = 2, times = 3)
[1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
> rep(1:4, times = 3, each = 2)
[1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
> |
```

# Repeats

Every object is repeated a different number of times:

```
> rep(1:4, 2:5)
```

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

R Console

```
> rep(1:4, 2:5)
[1] 1 1 2 2 2 3 3 3 3 4 4 4 4 4
```

# Repeats

Every object is repeated a different number of times:

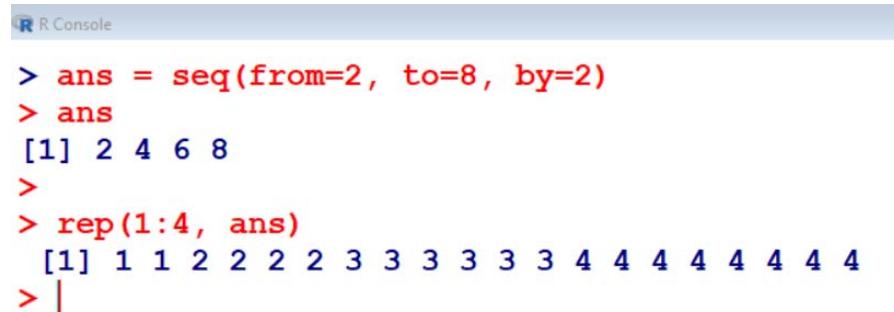
```
> ans = seq(from=2, to=8, by=2)
```

```
> ans
```

```
[1] 2 4 6 8
```

```
> rep(1:4, ans)
```

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

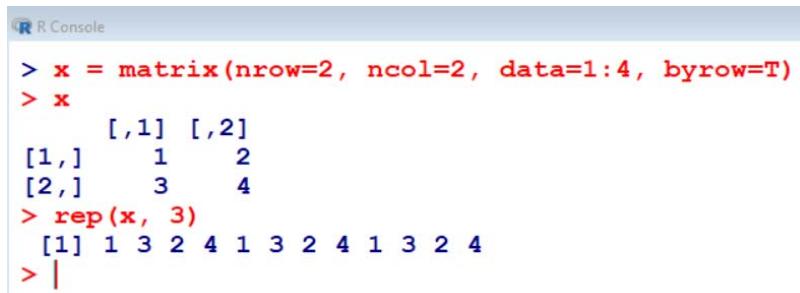


A screenshot of an R console window. The title bar says "R Console". The window contains the following R code and its output:

```
> ans = seq(from=2, to=8, by=2)
> ans
[1] 2 4 6 8
>
> rep(1:4, ans)
[1] 1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4
```

# Repeats

```
> x = matrix(nrow=2, ncol=2, data=1:4, byrow=T)  
  
> x  
     [,1] [,2]  
[1,]   1    2  
[2,]   3    4  
  
> rep(x, 3)  
[1] 1 3 2 4 1 3 2 4 1 3 2 4
```



A screenshot of an R console window titled "R Console". The window shows the following R code being run:

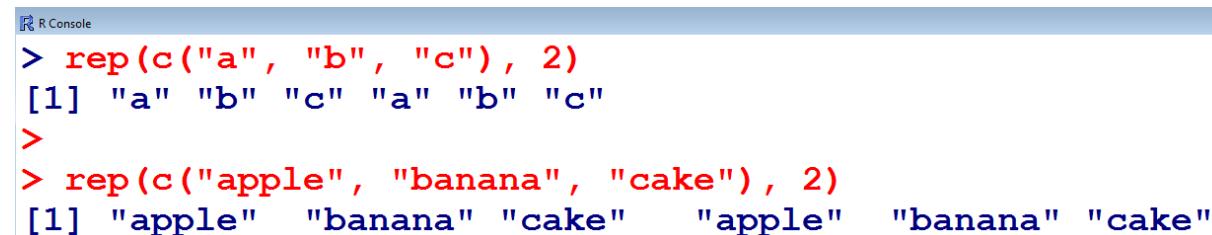
```
> x = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x
     [,1] [,2]
[1,]   1    2
[2,]   3    4
> rep(x, 3)
[1] 1 3 2 4 1 3 2 4 1 3 2 4
> |
```

# Repeats

## Repetition of characters

```
> rep(c("a", "b", "c"), 2)  
[1] "a" "b" "c" "a" "b" "c"
```

```
> rep(c("apple", "banana", "cake"), 2)  
[1] "apple"   "banana"  "cake"    "apple"   "banana"  "cake"
```



The image shows a screenshot of an R console window titled "R Console". It contains two examples of the `rep` function. The first example repeats the characters "a", "b", and "c" twice, resulting in the vector [1] "a" "b" "c" "a" "b" "c". The second example repeats the words "apple", "banana", and "cake" twice, resulting in the vector [1] "apple" "banana" "cake" "apple" "banana" "cake". The console output is in red, while the prompt and variable names are in blue.

```
R Console  
> rep(c("a", "b", "c"), 2)  
[1] "a" "b" "c" "a" "b" "c"  
>  
> rep(c("apple", "banana", "cake"), 2)  
[1] "apple"   "banana"  "cake"    "apple"   "banana"  "cake"
```

# Repeats

Repetition of characters for pre-specified length

```
> rep(2, length.out=5)
```

```
[1] 2 2 2 2 2
```

```
> rep(2, length=5)
```

```
[1] 2 2 2 2 2
```

```
> rep(c(2,3), length=5)
```

```
[1] 2 3 2 3 2
```

```
> rep(c(2,3,4), length=5)
```

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



R Console

```
> rep(2, length.out=5)
[1] 2 2 2 2 2
> rep(2, length=5)
[1] 2 2 2 2 2
>
> rep(c(2,3), length=5)
[1] 2 3 2 3 2
>
> rep(c(2,3,4), length=5)
[1] 2 3 4 2 3
```

# Repeats

Repetition of characters for pre-specified length

```
> rep("apple", length=5)  
[1] "apple" "apple" "apple" "apple" "apple"
```

```
> rep(c("a", "b", "c"), length=2)  
[1] "a" "b"
```

```
> rep(c("a", "b", "c"), length=5)  
[1] "a" "b" "c" "a" "b"
```

# Repeats

Repetition of characters for pre-specified length

R R Console

```
> rep("apple", length=5)
[1] "apple" "apple" "apple" "apple" "apple"
>
> rep(c("a", "b", "c"), length=2)
[1] "a" "b"
>
> rep(c("a", "b", "c"), length=5)
[1] "a" "b" "c" "a" "b"
> |
```

# **Foundations of R Software**

## **Lecture 29**

## **Sorting, Ordering and Mode**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Sorting

`sort` function sorts the values of a vector in ascending order (by default) or descending order.

## Syntax

`sort(x, decreasing = FALSE, ...)`

`sort(x, decreasing = FALSE, na.last = NA, ...)`

**x** Vector of values to be sorted

**decreasing** Should the sort be increasing or decreasing

**na.last** for controlling the treatment of `NAs`.

If `TRUE`, missing values in the data are put last;  
if `FALSE`, they are put first;  
if `NA`, they are removed.

# Sorting

## Example

```
> y = c(8,5,7,6)
```

```
> y
```

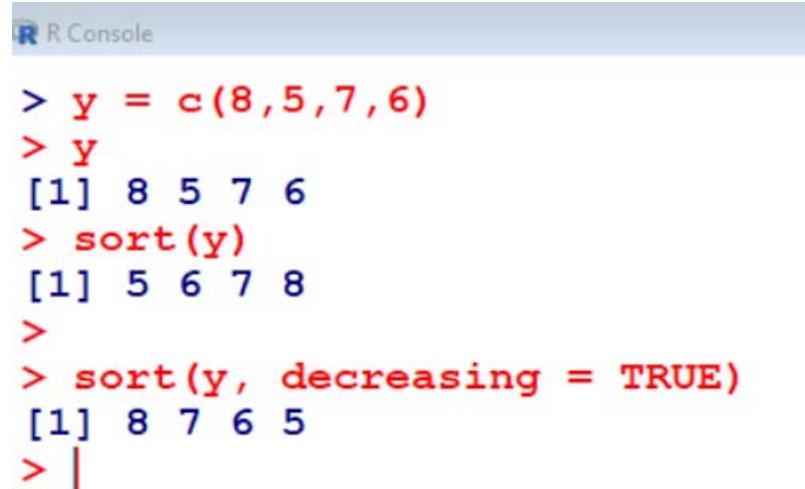
```
[1] 8 5 7 6
```

```
> sort(y)
```

```
[1] 5 6 7 8
```

```
> sort(y, decreasing = TRUE)
```

```
[1] 8 7 6 5
```

A screenshot of an R console window titled "R Console". The window shows several lines of R code and their corresponding output. The code includes creating a vector "y", printing it, sorting it in ascending order, and then sorting it in descending order. The output shows the vector "y" with values 8, 5, 7, 6, the sorted vector [1] 5 6 7 8, and the reversed sorted vector [1] 8 7 6 5.

```
R Console
> y = c(8,5,7,6)
> y
[1] 8 5 7 6
> sort(y)
[1] 5 6 7 8
>
> sort(y, decreasing = TRUE)
[1] 8 7 6 5
> |
```

# Ordering

`order` function sorts a variable according to the order of variable.

## Syntax

`order(x, decreasing = FALSE, ...)`

`order(x, decreasing = FALSE, na.last = TRUE, ...)`

`x`

Vector of values to be sorted

`decreasing`

Should the sort be increasing or decreasing

`na.last`

for controlling the treatment of `NAs`.

If `TRUE`, missing values in the data are put last;  
if `FALSE`, they are put first;  
if `NA`, they are removed.

# Ordering

## Example

```
> y = c(9,8,5,7,6)
```

```
> y
```

```
[1] 9 8 5 7 6
```

<b>Value</b>	9	8	5	7	6
<b>Position</b>	1	2	3	4	5

<b>Ordered value</b>	5	6	7	8	9
<b>Position</b>	3	5	4	2	1

# Ordering

## Example

```
> y
```

```
[1] 9 8 5 7 6
```

Value	9	8	5	7	6
Position	1	2	3	4	5
Ordered value	5	6	7	8	9
Position	3	5	4	2	1

```
> order(y)
```

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

```
> order(y, decreasing = TRUE)
```

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

# Ordering

```
R Console  
> y = c(9,8,5,7,6)  
> y  
[1] 9 8 5 7 6  
> order(y)  
[1] 3 5 4 2 1  
>  
> order(y, decreasing = TRUE)  
[1] 1 2 4 5 3  
> |
```

# Mode

**Every object has a mode.**

**The mode indicates how the object is stored in memory:** as a

- ✓ **number,**
- ✓ **character string,**
- ✓ **list of pointers to other objects,**
- ✓ **function etc.**

# Mode

**mode** function gives us such information.

## Syntax

**mode( )**

# Mode

Object	Example	Mode
Number	<code>1.234</code>	numeric
Vector of numbers	<code>c(6, 7, 8, 9)</code>	numeric
Character string	<code>"India"</code>	character
Vector of character strings	<code>c("India", "CANADA")</code>	character
Factor	<code>factor(c("MP", "UP"))</code>	numeric
List	<code>list("India", "CANADA")</code>	list
Data frame	<code>data.frame(x=1:2, y=c("India", "CANADA"))</code>	list
Function	<code>print</code>	function

# Mode

## Example

```
> mode(2.432)
```

```
[1] "numeric"
```

```
> mode(c(3,4,5,6,7,8))
```

```
[1] "numeric"
```

```
> mode("India")
```

```
[1] "character"
```

```
> mode(c("India", "CANADA"))
```

```
[1] "character"
```

R Console

```
> mode(2.432)
[1] "numeric"
>
> mode(c(3,4,5,6,7,8))
[1] "numeric"
>
> mode("India")
[1] "character"
>
```

# Mode:

## Example

```
> mode(factor(c("UP", "MP")) )  
[1] "numeric"  
  
> mode(list("India", "USA"))  
[1] "list"  
  
> mode(data.frame(x=1:2, y=c("India", "USA")))  
[1] "list"  
  
> mode(print)  
[1] "function"
```

# Mode

```
R R Console
> mode(factor(c("UP", "MP")))
[1] "numeric"
> mode(list("India", "USA"))
[1] "list"
> mode(data.frame(x=1:2, y=c("India", "USA")))
[1] "list"
> mode(print)
[1] "function"
```

# **Foundations of R Software**

## **Lecture 30**

### **Lists**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# **Lists**

**Vectors, matrices, and arrays is that each of these types of objects may only contain one type of data.**

**For example, a vector may contain all numeric data or all character data.**

**A list is a special type of object that can contain data of multiple types.**

**Lists are characterized by the fact that their elements do not need to be of the same object type.**

# Lists

- ❖ Lists can contain elements of different types so that the list elements may have different modes.
- ❖ Lists can even contain other structured objects, such as lists and data frames which allows to create recursive data structures.
- ❖ Lists can be indexed by position.  
So `x[[5]]` refers to the fifth element of `x`.

# Lists

- ❖ Lists can extract sublists.  
So `x[c(2,5)]` is a sublist of `x` that consists of the second and fifth elements.
- ❖ List elements can have names.  
Both `x[["Students"]]` and `x$Students` refer to the element named "Students".
- ❖ Difference between a vector and a list :
  - In a vector, all elements must have the same mode.
  - In a list, the elements can have different modes.

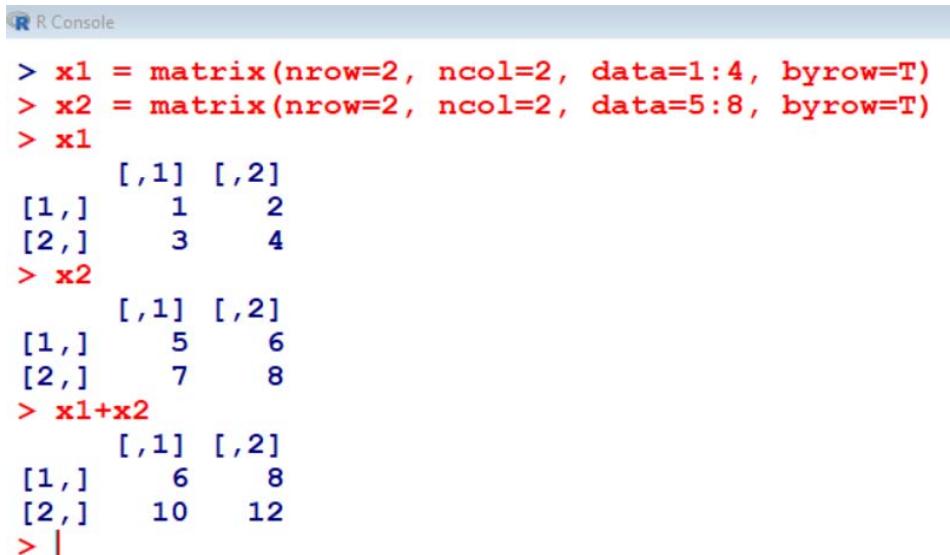
# Lists

## Example

```
> x1 = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x2 = matrix(nrow=2, ncol=2, data=5:8, byrow=T)
> x1
      [,1] [,2]
[1,]    1    2
[2,]    3    4

> x2
      [,1] [,2]
[1,]    5    6
[2,]    7    8

> x1+x2
      [,1] [,2]
[1,]    6    8
[2,]   10   12
```



The screenshot shows the R console interface. The title bar says "R Console". The main area displays the R code and its output. The code defines two matrices, x1 and x2, and then adds them together. The output shows the individual matrices and their sum.

# Lists

## Example

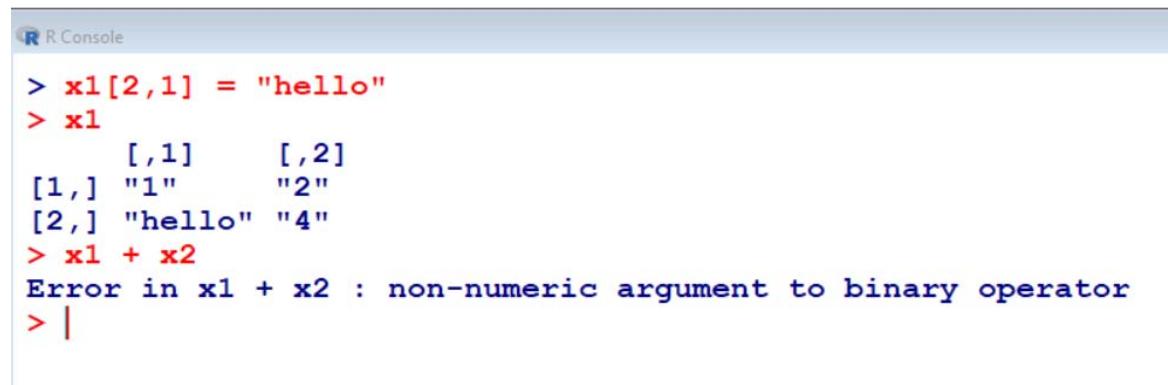
```
> x1[2,1] = "hello"
```

```
> x1
```

	[,1]	[,2]
[1,]	"1"	"2"
[2,]	"hello"	"4"

```
> x1 + x2
```

Error in x1 + x2 : non-numeric argument to  
binary operator



The screenshot shows an R console window with the title 'R Console'. The command line shows the following session:

```
R Console
> x1[2,1] = "hello"
> x1
      [,1]     [,2]
[1,] "1"      "2"
[2,] "hello"   "4"
> x1 + x2
Error in x1 + x2 : non-numeric argument to binary operator
> |
```

# Lists

Lists can contain any kind of objects as well as objects of different types. For example, lists can contain matrices as objects:

## Example

```
> x1 = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x2 = matrix(nrow=2, ncol=2, data=5:8, byrow=T)
> x1
      [,1] [,2]
[1,]    1    2
[2,]    3    4
> x2
      [,1] [,2]
[1,]    5    6
[2,]    7    8
```

```
R Console
> x1 = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x2 = matrix(nrow=2, ncol=2, data=5:8, byrow=T)
> x1
      [,1] [,2]
[1,]    1    2
[2,]    3    4
> x2
      [,1] [,2]
[1,]    5    6
[2,]    7    8
```

# Lists

## Example

```
> matlist = list(x1, x2)
```

```
> matlist
```

```
[[1]]
```

```
 [,1] [,2]
```

```
[1,] 1 2
```

```
[2,] 3 4
```

```
[[2]]
```

```
 [,1] [,2]
```

```
[1,] 5 6
```

```
[2,] 7 8
```

```
R Console
```

```
> x1 = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x2 = matrix(nrow=2, ncol=2, data=5:8, byrow=T)
> matlist = list(x1, x2)
> matlist
[[1]]
 [,1] [,2]
[1,] 1 2
[2,] 3 4

[[2]]
 [,1] [,2]
[1,] 5 6
[2,] 7 8
```

# Lists

## Example

```
> matlist[1]
[[1]]
  [,1] [,2]
[1,]   1   2
[2,]   3   4
```

```
> matlist[2]
[[1]]
  [,1] [,2]
[1,]   5   6
[2,]   7   8
```

R Console

```
> matlist[1]
[[1]]
  [,1] [,2]
[1,]   1   2
[2,]   3   4

> matlist[2]
[[1]]
  [,1] [,2]
[1,]   5   6
[2,]   7   8
```

# Lists

An example of a list that contains different object types:

```
> z1 = list( c("water", "juice", "lemonade"),
  rep(1:4, each=2), matrix(data=5:8, nrow=2,
  ncol=2, byrow=T) )

> z1
[[1]]
[1] "water"      "juice"       "lemonade"

[[2]]
[1] 1 1 2 2 3 3 4 4

[[3]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```

# Lists

```
R Console
> z1 = list( c("water", "juice", "lemonade"), rep(1:4, each=2), matrix(data=5:8, nrow=2, ncol=2, byrow=T) )
> |
```

```
R Console
> z1
[[1]]
[1] "water"      "juice"       "lemonade"

[[2]]
[1] 1 1 2 2 3 3 4 4

[[3]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```

# Lists

Access the elements of a list using the operator `[[[]]]`

Following commands work.

```
> z1[[1]]  
[1] "water" "juice" "lemonade"
```

Suppose we want to extract `"juice"`. The command

```
> z1[1][2] # Notice the positions of brackets  
[[1]] NULL
```

returns `NULL` instead of `"juice"`, while

```
> z1[[1]][2] # Notice the positions of brackets  
[1] "juice"
```

finally returns the desired result.

# Lists

```
R Console
> z1[[1]]
[1] "water"      "juice"       "lemonade"
>
> z1[1][2]
[[1]]
NULL

> z1[[1]][2]
[1] "juice"
```

# **Foundations of R Software**

## **Lecture 31**

## **Operations on Lists**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Merging Lists

Merging the lists.

Create two lists.

```
list1 = list(1,2,3)
```

```
list2 = list("water", "juice", "lemonade")
```

```
> list1  
[[1]]  
[1] 1  
  
[[2]]  
[1] 2  
  
[[3]]  
[1] 3
```

```
> list2  
[[1]]  
[1] "water"  
  
[[2]]  
[1] "juice"  
  
[[3]]  
[1] "lemonade"
```

# Merging Lists

```
list12 = c(list1, list2)
```

```
> list12
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 2
```

```
[[3]]
```

```
[1] 3
```

```
[[4]]
```

```
[1] "water"
```

```
[[5]]
```

```
[1] "juice"
```

```
[[6]]
```

```
[1] "lemonade"
```

# Merging Lists

R Console

```
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> list1
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> list2
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"
```

R Console

```
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> list12 = c(list1,list2)
> list12
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "water"

[[5]]
[1] "juice"

[[6]]
[1] "lemonade"
```

# Lists to vector

Converting list to vector. Use `unlist()` command.

```
list1 = list(1,2,3)
```

```
list2 = list("water", "juice", "lemonade")
```

```
> unlist(list1)
```

```
[1] 1 2 3
```

```
> unlist(list2)
```

```
[1] "water"    "juice"    "lemonade"
```

```
> mode(list1)
```

```
[1] "list"
```

```
> mode(unlist(list1))
```

```
[1] "numeric"
```

# Lists to vector

Converting list to vector. Use `unlist()` command

```
R Console
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> unlist(list1)
[1] 1 2 3
> unlist(list2)
[1] "water"      "juice"       "lemonade"
>
> mode(list1)
[1] "list"
> mode(unlist(list1))
[1] "numeric"
> |
```

# Appending lists

Appending list. Use `append()` command.

```
list1 = list(1,2,3)
```

```
list2 = list("water", "juice", "lemonade")
```

```
> append(list1, 100)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 100
```

```
> append(list2, "coffee")
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"

[[4]]
[1] "coffee"
```

# Appending lists

Appending list. Use **append( )** command.

```
R Console

> list2 = list("water", "juice", "lemonade")
> append(list1, 100)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 100

> append(list2, "coffee")
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"

[[4]]
[1] "coffee"
```

# Appending lists

Appending list after a position. Use `append( , after)` command.

```
list1 = list(1,2,3)
```

```
list2 = list("water", "juice", "lemonade")
```

```
> append(list1, 100,  
after = 2)
```

```
[[1]]  
[1] 1
```

```
[[2]]  
[1] 2
```

```
[[3]]  
[1] 100
```

```
[[4]]  
[1] 3
```

```
> append(list2, "coffee",  
after = 2)
```

```
[[1]]  
[1] "water"
```

```
[[2]]  
[1] "juice"
```

```
[[3]]  
[1] "coffee"
```

```
[[4]]  
[1] "lemonade"
```

# Appending lists

Appending list after a position. Use `append( , after )` command.

```
R Console
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> list1
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> list2
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"
```

```
R Console
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> append(list1, 100, after = 2)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 100

[[4]]
[1] 3

> append(list2, "coffee", after = 2)
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "coffee"

[[4]]
[1] "lemonade"
```

# Removing from list

Removing from list at a position. Use `listname[-position]` command.

```
list1 = list(1,2,3)
list2 = list("water", "juice", "lemonade")
> list1[-2]
[[1]]
[1] 1

[[2]]
[1] 3

> list2[-1]
[[1]]
[1] "juice"

[[2]]
[1] "lemonade"
```

# Removing from list

R Console

```
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> list1
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> list2
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"
```

R Console

```
> list1 = list(1,2,3)
> list2 = list("water", "juice", "lemonade")
> list1[-2]
[[1]]
[1] 1

[[2]]
[1] 3

> list2[-1]
[[1]]
[1] "juice"

[[2]]
[1] "lemonade"
```

## Extracting from list

Extracting from list. Use a range of indexes as `listname[indexes]` command.

```
list1 = list(1,2,3,4,5,6)
```

```
list2 = list("water", "juice", "lemonade", "tea",
"coffee", "milk")
```

Execute following commands

```
list1[2:4]
```

```
list1[c(1,3,5)]
```

```
list1[2:4]
```

```
list1[c(1,3,5)]
```

# Extracting from list

Extracting from list. Use a range of indexes as `listname[indexes]` command.

```
list1 = list(1,2,3,4,5,6)
```

```
> list1  
[[1]]  
[1] 1  
  
[[2]]  
[1] 2  
  
[[3]]  
[1] 3  
  
[[4]]  
[1] 4  
  
[[5]]  
[1] 5  
  
[[6]]  
[1] 6
```

```
> list1[2:4]  
[[1]]  
[1] 2  
  
[[2]]  
[1] 3  
  
[[3]]  
[1] 4
```

```
> list1[c(1,3,5)]  
[[1]]  
[1] 1  
  
[[2]]  
[1] 3  
  
[[3]]  
[1] 5
```

# Extracting from list

Extracting from list. Use a range of indexes as `listname[indexes]` command.

```
list1 = list(1,2,3,4,5,6)
```

R Console

```
> list1 = list(1,2,3,4,5,6)
> list1
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 4

[[5]]
[1] 5

[[6]]
[1] 6
```

R Console

```
> list1[2:4]
[[1]]
[1] 2

[[2]]
[1] 3

[[3]]
[1] 4
```

R Console

```
> list1[c(1,3,5)]
[[1]]
[1] 1

[[2]]
[1] 3

[[3]]
[1] 5
```

# Extracting from list

Extracting from list. Use a range of indexes as `listname[indexes]` command.

```
list2 = list("water", "juice", "lemonade", "tea",
"coffee", "milk")
```

```
> list2
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"

[[4]]
[1] "tea"

[[5]]
[1] "coffee"

[[6]]
[1] "milk"
```

```
> list2[2:4]
[[1]]
[1] "juice"

[[2]]
[1] "lemonade"

[[3]]
[1] "tea"
```

```
> list2[c(1,3,5)]
[[1]]
[1] "water"

[[2]]
[1] "lemonade"

[[3]]
[1] "coffee"
```

# Extracting from list

Extracting from list. Use a range of indexes as

`listname[indexes]` command.

```
list2 = list("water", "juice", "lemonade",
"tea", "coffee", "milk")
```

The image shows three separate R console windows side-by-side. The leftmost window displays the creation of a list named 'list2' containing six items: 'water', 'juice', 'lemonade', 'tea', 'coffee', and 'milk'. The middle window shows the extraction of elements at indices 2 through 4, resulting in a list containing 'juice', 'lemonade', and 'tea'. The rightmost window shows the extraction of elements at indices 1, 3, and 5, resulting in a list containing 'water', 'lemonade', and 'coffee'.

```
R Console
> list2 = list("water", "juice", "lemonade", "tea", "coffee", "milk")
> list2
[[1]]
[1] "water"

[[2]]
[1] "juice"

[[3]]
[1] "lemonade"

[[4]]
[1] "tea"

[[5]]
[1] "coffee"

[[6]]
[1] "milk"

R Console
> list2[2:4]
[[1]]
[1] "juice"

[[2]]
[1] "lemonade"

[[3]]
[1] "tea"

R Console
> list2[c(1,3,5)]
[[1]]
[1] "water"

[[2]]
[1] "lemonade"

[[3]]
[1] "coffee"
```

# **Foundations of R Software**

## **Lecture 32**

## **Vector Indexing**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Vector Indexing

## □ Vector and its values

```
> x = 1:10
```

```
> x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> x[ (x > 5) ]
```

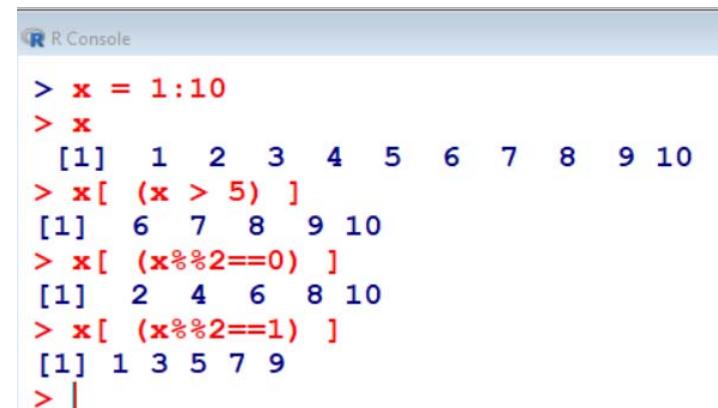
```
[1] 6 7 8 9 10
```

```
> x[ (x%%2==0) ] #%% indicates x mod y
```

```
[1] 2 4 6 8 10      #values for which x mod 2 is 0
```

```
> x[ (x%%2==1) ]
```

```
[1] 1 3 5 7 9      #values for which x mod 2 is 1
```



The screenshot shows the R console interface with a light blue header bar containing the R logo and the text "R Console". Below the header, there is a scrollable text area displaying R code and its output. The code demonstrates various ways to index a vector 'x' containing integers from 1 to 10.

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[ (x > 5) ]
[1] 6 7 8 9 10
> x[ (x%%2==0) ]
[1] 2 4 6 8 10
> x[ (x%%2==1) ]
[1] 1 3 5 7 9
> |
```

# Vector Indexing

## □ Vector with missing observation(s)

```
> x[5] = NA
```

```
> x
```

```
[1] 1 2 3 4 NA 6 7 8 9 10
```

```
> y = x[ !is.na(x) ] #! Means negation
```

```
> y
```

```
[1] 1 2 3 4 6 7 8 9 10 # 5 is missing
```

```
> mean(x)
```

```
[1] NA
```

```
> mean(y)
```

```
[1] 5.555556
```

# Vector Indexing

```
R R Console

> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[5] = NA
> x
[1] 1 2 3 4 NA 6 7 8 9 10
> y = x[ !is.na(x) ]
> y
[1] 1 2 3 4 6 7 8 9 10
> mean(x)
[1] NA
> mean(y)
[1] 5.555556
> |
```

# Vector Indexing

## □ Vector of negative integers

```
> x = 1:10
```

```
> x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

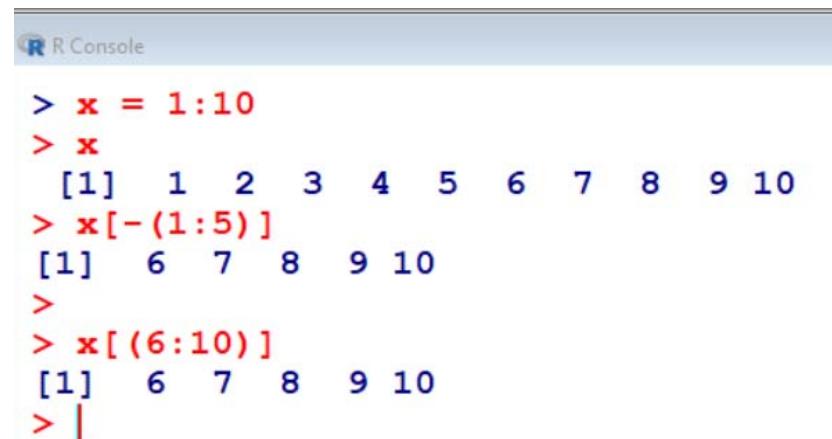
```
> x[ -(1:5) ]
```

```
[1] 6 7 8 9 10
```

has the same outcome as

```
> x[(6:10)]
```

```
[1] 6 7 8 9 10
```



The screenshot shows an R console window titled "R Console". It displays the following R session:

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[ -(1:5) ]
[1] 6 7 8 9 10
>
> x[(6:10)]
[1] 6 7 8 9 10
> |
```

## □ String vector

The elements of a vector can be named.

Using these **names**, we can access the vector elements.

**names** is used for functions to get or set the names of an object

```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
```

```
> z
```

```
$a1
```

```
[1] 1
```

```
$a2
```

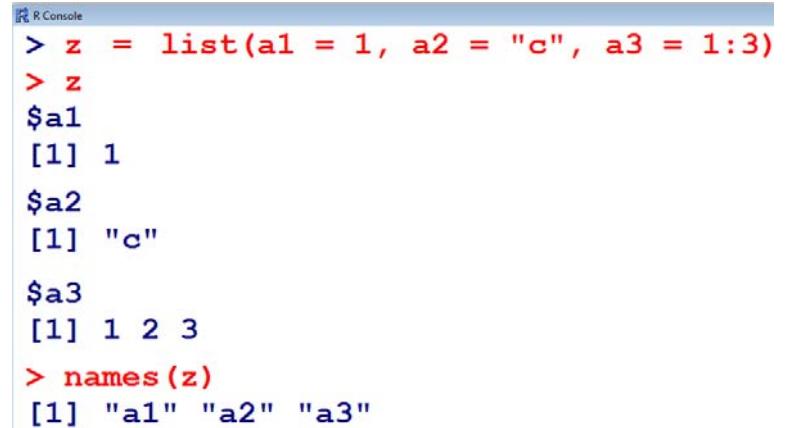
```
[1] "c"
```

```
$a3
```

```
[1] 1 2 3
```

```
> names(z)
```

```
[1] "a1" "a2" "a3"
```



```
R Console
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> z
$a1
[1] 1
$a2
[1] "c"
$a3
[1] 1 2 3
> names(z)
[1] "a1" "a2" "a3"
```

## □ String vector

Suppose want to change just the name of the third element.

```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)  
  
> names(z)[3] = "c2"  
  
> z  
  
$a1  
[1] 1  
  
$a2  
[1] "c"  
  
$c2  
[1] 1 2 3
```

```
R Console  
> z = list(a1 = 1, a2 = "c", a3 = 1:3)  
> names(z)[3] = "c2"  
> z  
$a1  
[1] 1  
$a2  
[1] "c"  
$c2  
[1] 1 2 3
```

## □ String vector

### Example

`names` is used for functions to get or set the names of an object

```
> x = c(water=1, juice=2, lemonade=3 )
```

```
> names(x)
[1] "water"      "juice"       "lemonade"
```

```
> x["juice"]
juice
2
```

```
R R Console
> x = c(water=1, juice=2, lemonade=3 )
> names(x)
[1] "water"      "juice"       "lemonade"
>
> x["juice"]
juice
2
```

## □ Empty index

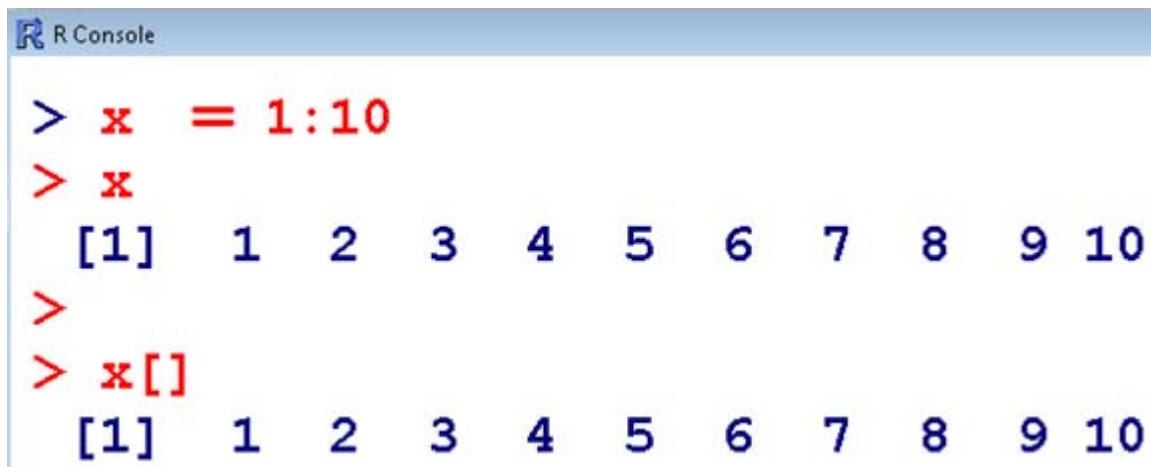
```
> x = 1:10
```

```
>x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> x[ ]
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```



A screenshot of an R console window titled "R Console". The window shows the following R session:

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
>
> x[ ]
[1] 1 2 3 4 5 6 7 8 9 10
```

## Mixed mode

List can be heterogeneous (mixed modes).

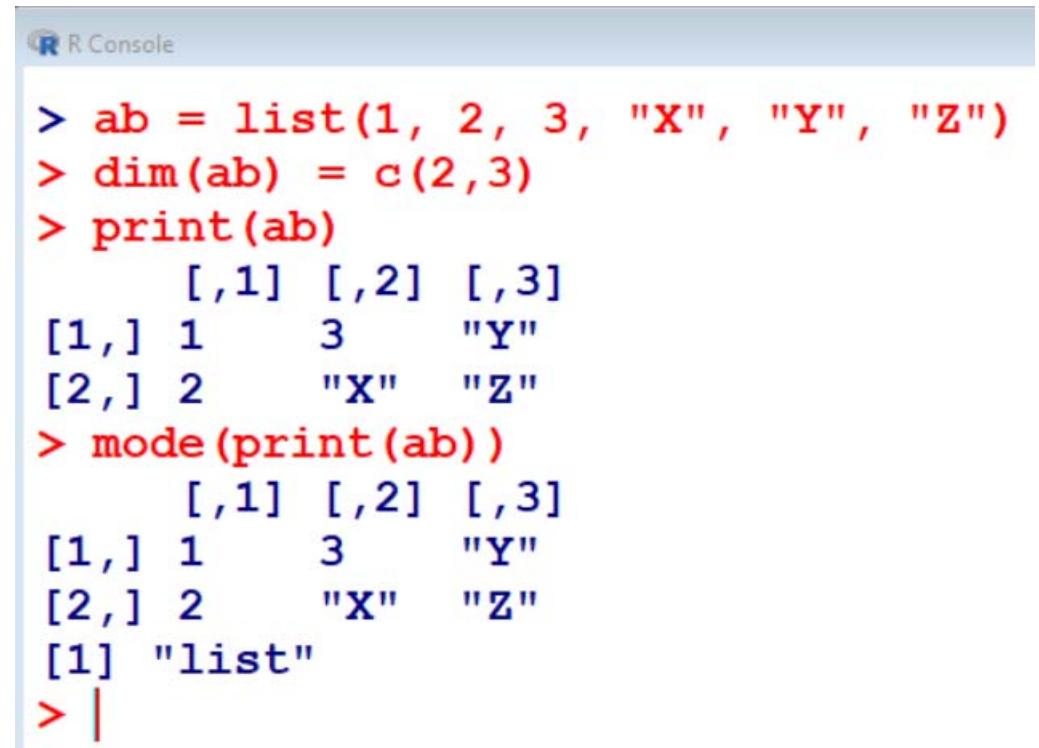
We can start with a heterogeneous list, give it dimensions, and thus create a heterogeneous list that is a mixture of numeric and character data:

### Example

```
> ab = list(1, 2, 3, "X", "Y", "Z")  
> dim(ab) = c(2,3)  
> print(ab)  
      [,1] [,2] [,3]  
[1,] 1     3     "Y"  
[2,] 2     "X"   "Z"
```

## □ Mixed mode

```
> mode(print(ab))  
[ ,1] [ ,2] [ ,3]  
[1, ] 1      3      "Y"  
[2, ] 2      "X"    "Z"  
[1] "list"
```



The screenshot shows an R console window titled "R Console". It displays the following R session:

```
> ab = list(1, 2, 3, "X", "Y", "Z")  
> dim(ab) = c(2,3)  
> print(ab)  
[ ,1] [ ,2] [ ,3]  
[1, ] 1      3      "Y"  
[2, ] 2      "X"    "Z"  
> mode(print(ab))  
[ ,1] [ ,2] [ ,3]  
[1, ] 1      3      "Y"  
[2, ] 2      "X"    "Z"  
[1] "list"  
> |
```

## □ Matrices created from Lists

```
R Console
> ab = list(1, 2, 3, "X", "Y", "Z")
> dim(ab) = c(2,3)
> print(ab)
      [,1] [,2] [,3]
[1,] 1     3     "Y"
[2,] 2     "X"   "Z"
```

# **Foundations of R Software**

## **Lecture 33**

## **Factors**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Categorical variables

Quantitative variables

Example:

Height (in meters) – 1.65, 1.76, ...

Qualitative variables

Example:

Gender – Male, Female

Performance – Excellent, Good, Average, Bad ...

# Categorical variables

## Categorical variables

**Example:**

X : Gender – Male, Female

X = 0 if a person is male

X = 1 if a person is female

**Example:**

Performance	Excellent	Average	Good	Bad	Labels
X	1	2	3	4	Numeric codes

The categories are stored internally as numeric codes, with labels to provide meaningful names for each code.

# Factors

**Factors represent categorical variables and are used as grouping indicators.**

# Factors

**Example:**

Suppose we denote the three colours of balls in a basket by following numbers:

**Red = 1, Blue = 2, Green = 3**



Suppose we draw five balls with following colours:

**Red, Green, Green, Blue, Red**

This outcome of colours can be coded by numbers

# Factors

**Each character is mapped to a code.**

**Factors represent categorical variables and are used as grouping indicators.**

**The categories are stored internally as numeric codes, with labels to provide meaningful names for each code.**

# Factors

The order of the labels is important.

First label is mapped to code 1.

Second label is mapped to code 2 and so on.

The values of the codes are always restricted to  $1, 2, \dots, k$ , to represent  $k$  discrete categories.

Here “**Red**” is mapped to code 1,

“**Blue**” is mapped to code 2 and

“**Green**” is mapped to code 3.



# Factors

We have a vector of character strings or integers.

R's term for a categorical variable is a factor.

In R, each possible value of a categorical variable is called a level.

A vector of levels is called a factor.

A categorical variable is characterized by a (here: finite) number of levels called as factor levels.

# Factors

To define a factor, we start with

- a vector of values,
- a second vector that gives the collection of possible values, and
- a third vector that gives labels to the possible values.

# Factors

The `factor` function encodes the vector of discrete values into a factor:

`factor(x)`

where `x` is a vector of strings or integers.

If the vector contains only a subset of possible values and not the entire values, then include a second argument that gives the possible levels of the factor:

`factor(x, levels)`

# Factors

## Usage

```
factor(x = character(), levels, labels =  
levels, exclude = NA, ...)
```

**levels** : Determines the categories of the factor variable.

Default is the sorted list of all the distinct values of **x**.

**labels** : (Optional) Vector of values that will be the labels of the categories in the **levels** argument.

**exclude** : (Optional) It defines which levels will be classified as **NA** in any output using the factor variable.

# Factors

Look into `help`

```
> help("factor")
```

factor {base}

R Documentation

## Factors

### Description

The function `factor` is used to encode a vector as a factor (the terms ‘category’ and ‘enumerated type’ are also used for factors). If argument `ordered` is `TRUE`, the factor levels are assumed to be ordered. For compatibility with S there is also a function `ordered`.

`is.factor`, `is.ordered`, `as.factor` and `as.ordered` are the membership and coercion functions for these classes.

### Usage

```
factor(x = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)

ordered(x, ...)

is.factor(x)
is.ordered(x)
```

# Factors

**Example:**

Suppose we roll a die seven times and observe the outcome in the vector **y**.

```
> y = c(1, 4, 3, 5, 4, 2, 4)
```



Possible values of upper face of die are 1 to 6 and we store them in a vector **possible.dieface**

```
> possible.dieface = c(1, 2, 3, 4, 5, 6)
```

# Factors

Example:

We wish to label the rolls by the words “one”, “two”, ..., “six”.

We put these labels in the vector `labels.diefaces`:

```
> labels.dieface = c("one", "two", "three",
"four", "five", "six")
```

Construct the factor variable `facy` using the function `factor`:

```
> facy = factor(y, levels = possible.dieface,
labels = labels.dieface)
```

# Factors

**Example:**

Observe the difference between a character vector and a factor.

```
> facy  
[1] one four three five four two four  
Levels: one two three four five six
```

Note that

```
y = c(1, 4, 3, 5, 4, 2, 4)
```

# Factors

```
R Console
> y = c(1, 4, 3, 5, 4, 2, 4)
> y
[1] 1 4 3 5 4 2 4
> possible.dieface = c(1, 2, 3, 4, 5, 6)
> possible.dieface
[1] 1 2 3 4 5 6
> labels.dieface = c("one", "two", "three", "four", "five", "six")
> labels.dieface
[1] "one"   "two"   "three" "four"  "five"  "six"
> facy = factor(y, levels = possible.dieface, labels = labels.dieface)
> facy
[1] one   four  three five  four  two   four
Levels: one two three four five six
> |
```

# **Foundations of R Software**

## **Lecture 34**

## **Factors – Class and Unclass**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Factors

The `factor` function encodes the vector of discrete values into a factor.

Usage

```
factor(x = character(), levels, labels =  
levels, exclude = NA, ...)
```

`levels` : Determines the categories of the factor variable.

Default is the sorted list of all the distinct values of `x`.

`labels` : (Optional) Vector of values that will be the labels of the categories in the `levels` argument.

`exclude` : (Optional) It defines which levels will be classified as `NA` in any output using the factor variable.

# Factors

A vector can be turned into a factor with the command `as.factor`:

```
> x = c(3, 4, 5, 6, 1, 2, 3, 3, 4, 4, 5, 6)
> x
[1] 3 4 5 6 1 2 3 3 4 4 5 6

> y = as.factor(x)
> y
[1] 3 4 5 6 1 2 3 3 4 4 5 6
Levels: 1 2 3 4 5 6
```

The single levels are ordered numerically:

1 → 2 → 3 → 4 → 5 → 6

# Factors

## Example

```
> x = factor( c("lemonade", "lemonade",
  "juice", "lemonade", "water") )

> x
[1] lemonade  lemonade  juice    lemonade  water
Levels: juice lemonade water
```

The single levels are ordered alphabetically:

**juice --- lemonade --- water**

# Factors

## Example

```
R Console
> x = c(3, 4, 5, 6, 1, 2, 3, 3, 4, 4, 5, 6)
> x
[1] 3 4 5 6 1 2 3 3 4 4 5 6
> y = as.factor(x)
> y
[1] 3 4 5 6 1 2 3 3 4 4 5 6
Levels: 1 2 3 4 5 6
> |
```

```
R Console
> x = factor( c("lemonade", "lemonade", "juice", "lemonade", "water") )
> x
[1] lemonade lemonade juice      lemonade water
Levels: juice lemonade water
> |
```

# Factors

`class` function :

All objects in R have a class and function `class` reports it.

For simple vectors, this is just the mode, e.g. "`numeric`", "`logical`", "`character`", "`list`", "`matrix`", "`array`", "`factor`" and "`data.frame`".

A special attribute class of the object is used to allow for an object-oriented style of programming in R.

# Factors

```
class function :
```

```
> class(9)
[1] "numeric"
```

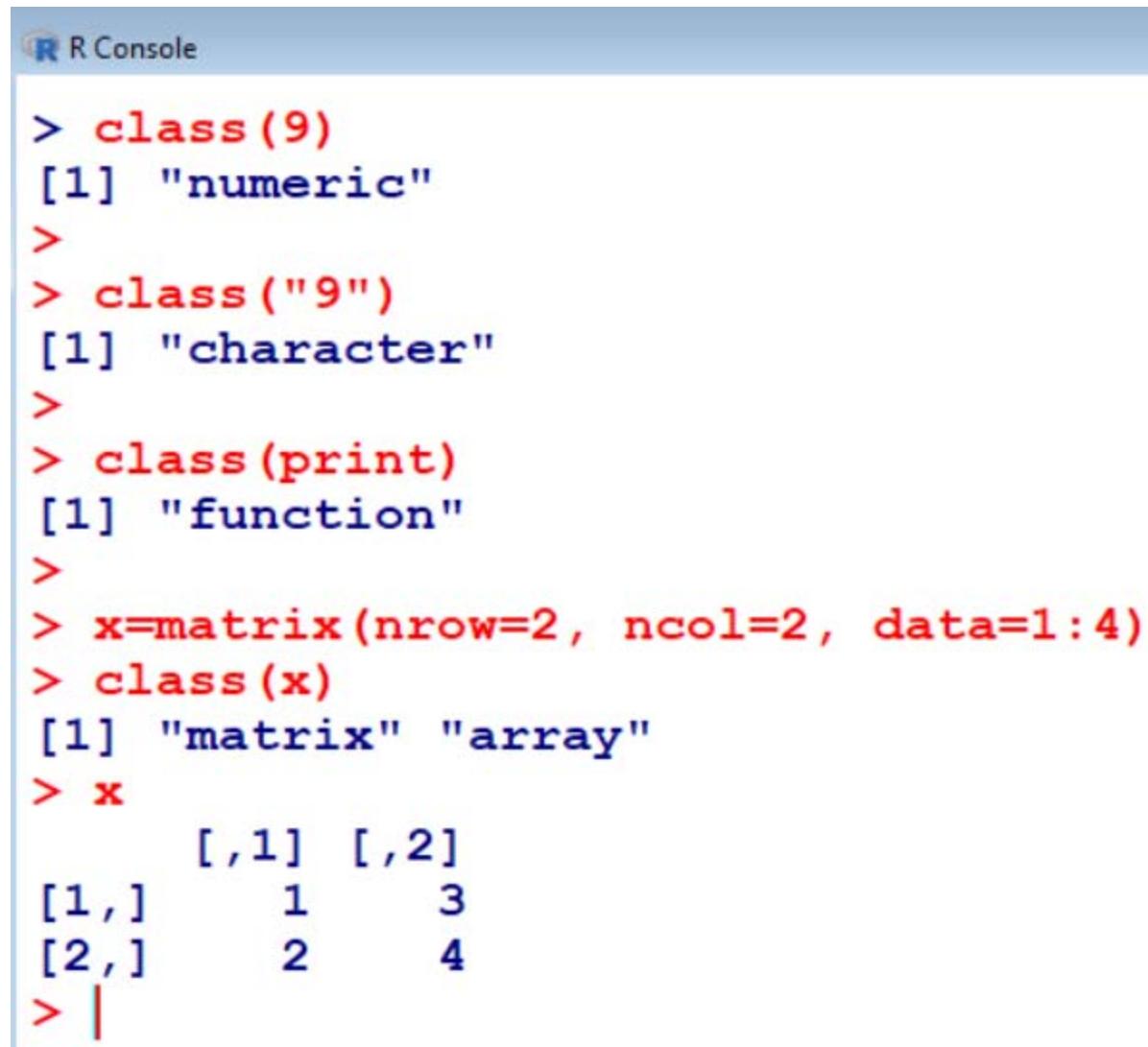
```
> class("9")
[1] "character"
```

```
> class(print)
[1] "function"
```

```
> x = matrix(nrow=2, ncol=2, data=1:4)
> class(x)
[1] "matrix" "array"
```

# Factors

**class** function :



R Console

```
> class(9)
[1] "numeric"
>
> class("9")
[1] "character"
>
> class(print)
[1] "function"
>
> x=matrix(nrow=2, ncol=2, data=1:4)
> class(x)
[1] "matrix" "array"
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> |
```

# Factors

**unclass** function

For example if an object has class "**data.frame**", it will be printed in a certain way, the **plot()** function will display it graphically in a certain way etc.

**unclass()** is used to temporarily remove the effects of class.

Use **help("unclass")** to get more information.

# Factors

Examples: `unclass` function

Consider a group of strings and store it as factors

```
> brands = c("A", "A", "B", "B", "B", "B", "C")
```

```
> brands
```

```
[1] "A" "A" "B" "B" "B" "B" "C"
```

```
> brands_fac = factor(brands)
```

```
> brands_fac
```

```
[1] A A B B B C
```

Levels: A B C

# Factors

Examples: `unclass` function

Apply `unclass` to this group of factors.

`unclass` will convert the factors to their numbers, like

```
> unclass(brands_fac)
```

```
[1] 1 1 2 2 2 2 3
```

```
attr(,"levels")
```

```
[1] "A" "B" "C"
```

Suppose we have a vector of colors as

```
> colours = c("blue", "green", "red")
```

```
> colours
```

```
[1] "blue" "green" "red"
```

# Factors

Examples: `unclass` function

With the following

```
> unclass(brands_fac)
```

```
[1] 1 1 2 2 2 2 3
```

```
attr(,"levels")
```

```
[1] "A" "B" "C"
```

numbers you can convert the factors to colors, by doing:

```
> colours [unclass(brands_fac)]
```

```
[1] "blue"   "blue"   "green"  "green"  "green"
```

```
"green"  "red"
```

Recall `brands = c("A","A","B","B","B","B","C")`

# Factors

Examples: `unclass` function

```
R Console

> colours = c("blue", "green", "red" )
> colours
[1] "blue"  "green" "red"
>
> brands = c("A","A","B","B","B","B","C")
> brands
[1] "A"  "A"  "B"  "B"  "B"  "B"  "C"
>
> brands_fac = factor(brands)
> brands_fac
[1] A A B B B B C
Levels: A B C
>
> unclass(brands_fac)
[1] 1 1 2 2 2 2 3
attr(,"levels")
[1] "A"  "B"  "C"
>
> colours [unclass(brands_fac) ]
[1] "blue"  "blue"  "green" "green" "green" "green" "red"
> |
```

# Factors

Examples: `unclass` function

The command `unclass` shows, an integer is assigned to every factor level:

```
> x = factor( c( "lemonade", "lemonade",
  "juice", "lemonade", "water" ) )

> unclass(x)
[1] 2 2 1 2 3

attr(,"levels")
[1] "juice" "lemonade" "water"
```

# Factors

```
R Console

> x = factor( c( "lemonade", "lemonade", "juice", "lemonade", "water" ) )
> x
[1] lemonade lemonade juice      lemonade water
Levels: juice lemonade water
>
> unclass(x)
[1] 2 2 1 2 3
attr(,"levels")
[1] "juice"    "lemonade" "water"
>
```

# Factors

If a different assignment is desired, the parameter **levels** can be used:

```
> x = factor( c("lemonade", "lemonade",
  "juice", "lemonade", "water"),
  levels=c("water", "juice", "lemonade") )

> x
[1] lemonade lemonade juice      lemonade water
Levels: water juice lemonade
```

# Factors

```
> unclass(x)
[1] 3 3 2 3 1

attr(,"levels")
[1] "water"      "juice"       "lemonade"

> levels(x)
[1] "water"      "juice"       "lemonade"
```

# Factors

```
R Console

> x = factor( c("lemonade", "lemonade", "juice", "lemonade", "water"),
+ levels=c("water", "juice", "lemonade") )
> x
[1] lemonade lemonade juice      lemonade water
Levels: water juice lemonade
>
> unclass(x)
[1] 3 3 2 3 1
attr(,"levels")
[1] "water"    "juice"     "lemonade"
>
> levels(x)
[1] "water"    "juice"     "lemonade"
> |
```

# Factors

Example for an ordered factor:

```
> income = ordered(c("high", "high", "low",
"medium", "medium"), levels=c("low", "medium",
"high") )  
  
> income  
[1] high    high    low     medium medium  
Levels: low < medium < high  
  
> unclass(income)  
[1] 3 3 1 2 2  
attr(,"levels")  
[1] "low"    "medium" "high"
```

# Factors

```
R Console

> income = ordered(c("high", "high", "low", "medium", "medium"),
+ levels=c("low", "medium", "high") )
> income
[1] high   high   low    medium medium
Levels: low < medium < high
>
> unclass(income)
[1] 3 3 1 2 2
attr(,"levels")
[1] "low"    "medium" "high"
> |
```

# **Foundations of R Software**

**Lecture 35**

## **Strings – Display and Formatting**

**⋮⋮⋮**

## **Print and Format Function**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# **Strings**

- **Formatting and Display of Strings**
- **Operations with Strings**

**We need formatting and display of strings to obtain the results of specific operations in required format.**

# Formatting and Display of Strings

Important commands regarding formatting and display are `print`, `format`, `cat`, and `paste`.

`print` function prints its argument.

Usage

`print()`

`print()` is a generic command that is available for every object class.

# Formatting and Display of Strings

## Examples: Print numbers

```
> print( sqrt(2) )
```

```
[1] 1.414214
```

```
> print( sqrt(2), digits=5 )
```

```
[1] 1.4142
```

```
> print( sqrt(2), digits=10 )
```

```
[1] 1.414213562
```



```
> print( sqrt(2) )
[1] 1.414214
> print( sqrt(2), digits=5 )
[1] 1.4142
> print( sqrt(2), digits=10 )
[1] 1.414213562
> |
```

# Formatting and Display of Strings

## Examples: Print characters

```
> print("apple")
```

```
[1] "apple"
```

```
> print(c("apple", "banana"))
```

```
[1] "apple" "banana"
```

```
> print(c("apple", "banana", 6, 10))
```

```
[1] "apple" "banana" "6" "10"
```



The screenshot shows the R console interface. The title bar says "R Console". Below it, there are three lines of R code and their corresponding output. The first line is "print("apple")" followed by "[1] "apple"". The second line is "print(c("apple", "banana"))" followed by "[1] "apple" "banana"". The third line is "print(c("apple", "banana", 6, 10))" followed by "[1] "apple" "banana" "6" "10"". The numbers 6 and 10 are in red, while the strings are in blue.

```
> print("apple")
[1] "apple"
> print(c("apple", "banana"))
[1] "apple" "banana"
> print(c("apple", "banana", 6, 10))
[1] "apple" "banana" "6" "10"
```

# Formatting and Display of Strings

Format an R object for pretty printing.

Usage

`format(x, ...)`

**x** is any R object; typically numeric.

# Formatting and Display of Strings

## Usage

```
format(x, trim = FALSE, digits = NULL, nsmall  
= 0L, justify = c("left", "right", "centre",  
"none"), width = NULL, ...)
```

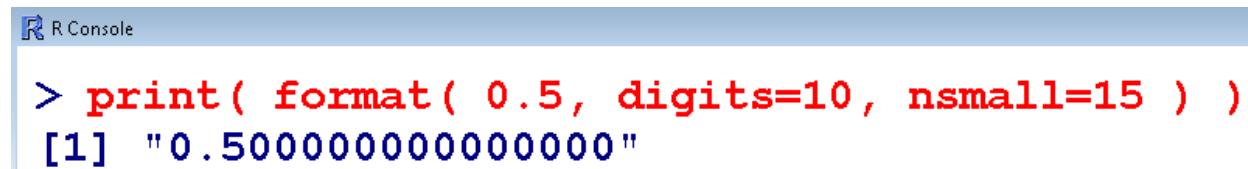
**digits** shows how many significant digits are to be used

**nsmall** shows the minimum number of digits to the right of  
the decimal point

**justify** provides left-justified (the default), right-justified, or  
centred.

# Formatting and Display of Strings

```
> print( format( 0.5, digits=10, nsmall=15 ) )  
[1] "0.500000000000000"
```



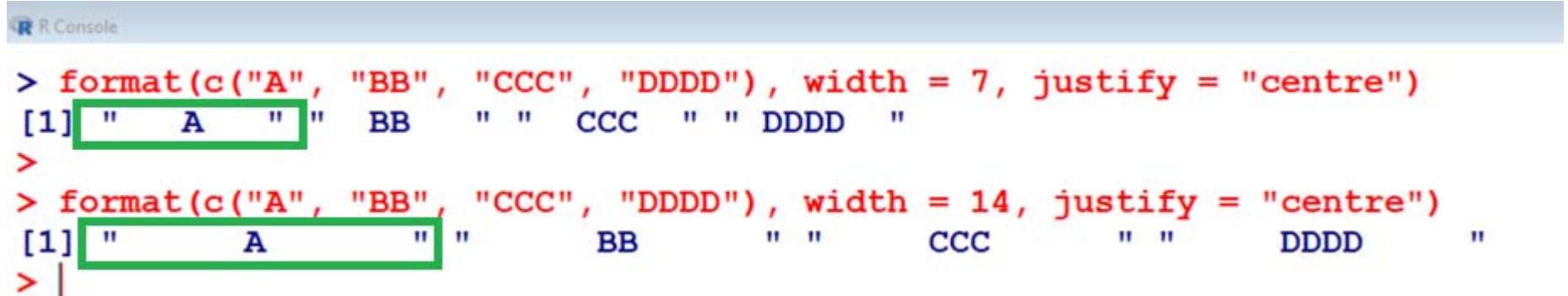
A screenshot of an R console window titled "R Console". The window contains the same R code and output as the previous text block, demonstrating the use of the `format` function to control the display of floating-point numbers.

```
> print( format( 0.5, digits=10, nsmall=15 ) )  
[1] "0.500000000000000"
```

# Formatting and Display of Strings: Use of width

```
> format(c("A", "BB", "CCC", "DDDD"),  
width = 7, justify = "centre")  
[1] " A " " BB " " CCC " " DDDD "
```

```
> format(c("A", "BB", "CCC", "DDDD"),  
width = 14, justify = "centre")  
[1] " A " " BB " " CCC  
" " DDDD "
```



The screenshot shows the R console interface with a light blue header bar containing the R logo and the word "Console". Below the header, there are two code snippets demonstrating string formatting.

```
R Console  
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "centre")  
[1] " A " " BB " " CCC " " DDDD "  
>  
> format(c("A", "BB", "CCC", "DDDD"), width = 14, justify = "centre")  
[1] " A " " BB " " CCC " " DDDD "
```

# Formatting and Display of Strings: Use of `justify`

```
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify  
= "centre")  
[1] "    A    " "  BB   " " CCC  " " DDDD "
```

```
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify  
= "left")  
[1] "A"      "BB"     "CCC"    "DDDD"  "
```

```
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify  
= "right")  
[1] "        A" "      BB" "    CCC" "  DDDD"
```

```
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify  
= "none")  
[1] "A"      "BB"     "CCC"    "DDDD"
```

# Formatting and Display of Strings: Use of justify

```
R Console

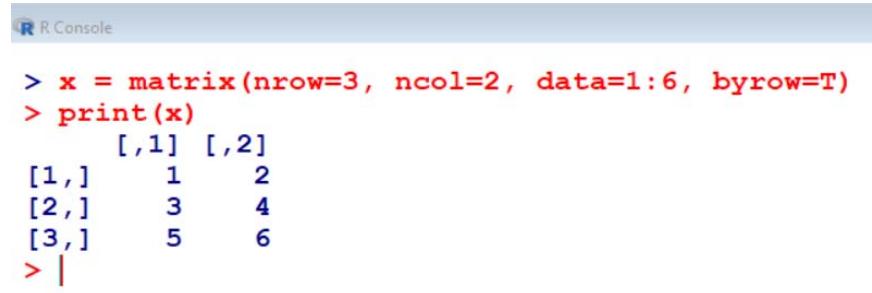
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "centre")
[1] " A   " " BB  " " CCC " " DDDD "
>
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "left")
[1] "A      " "BB     " "CCC    " "DDDD   "
>
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "right")
[1] "        A" "      BB" "     CCC" "    DDDD"
>
> format(c("A", "BB", "CCC", "DDDD"), width = 7, justify = "none")
[1] "A"     "BB"    "CCC"   "DDDD"
> |
```

# Formatting and Display of Strings

Example:

```
> x = matrix(nrow=3, ncol=2, data=1:6, byrow=T)

> print(x)
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```



The screenshot shows the R console window with the title "R Console". Inside, the R code is displayed in red, followed by the resulting matrix output in black. The matrix has 3 rows and 2 columns, containing the values 1 through 6.

Here, a matrix is displayed in the R command window.

One can specify the desired number of digits with the option **digits**.

# Formatting and Display of Large Quantities

Use command **big.mark** for printing large quantities with **format**.

Example, we can print a number with sequences separated by a comma ","

```
> format(1234567, big.mark = ",")
```

```
[1] "1,234,567"
```

```
> format(12345678, big.mark = ",")
```

```
[1] "12,345,678"
```

```
> format(123456789, big.mark = ",")
```

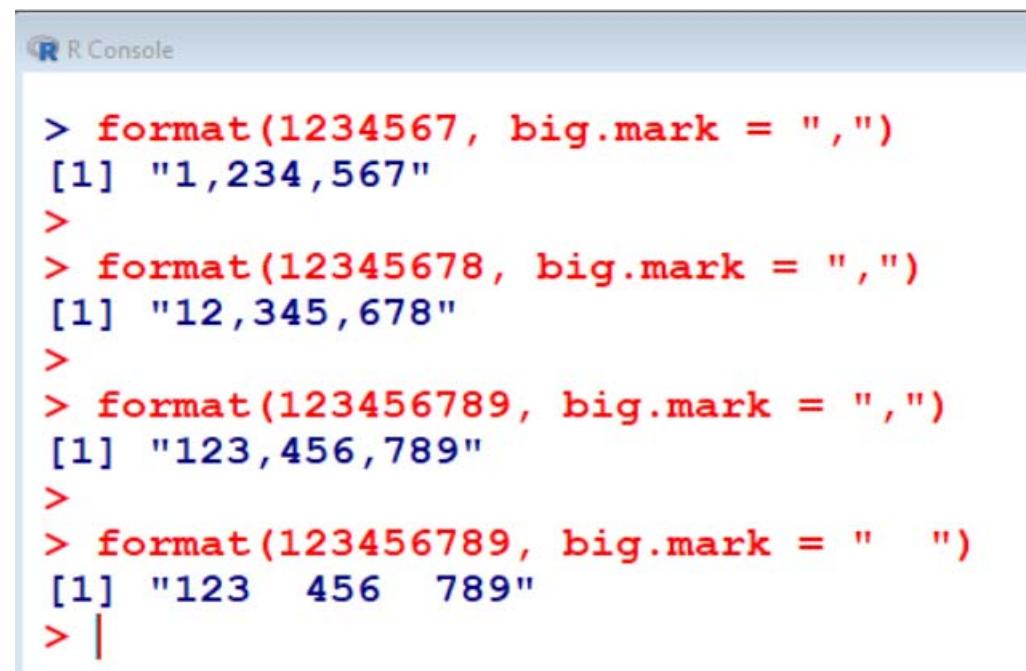
```
[1] "123,456,789"
```

# Formatting and Display of Large Quantities

Example, we can print a number with sequences separated by spaces

```
" "
```

```
> format(123456789, big.mark = " ")  
[1] "123 456 789"
```



A screenshot of an R console window titled "R Console". The window shows several lines of R code being executed. The code consists of the command `format(123456789, big.mark = ",")` followed by its output [1] "1,234,567". This is followed by several blank lines and then the command `format(12345678, big.mark = ",")` with its output [1] "12,345,678". Another blank line follows, then the command `format(123456789, big.mark = ",")` with its output [1] "123,456,789". Another blank line follows, then the command `format(123456789, big.mark = " ")` with its output [1] "123 456 789". A cursor is visible at the end of the last line.

# **Foundations of R Software**

**Lecture 36**

## **Strings – Display and Formatting**

**---**

## **Print and Format with Concatenate**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Formatting and Display of Strings

Suppose we want to print

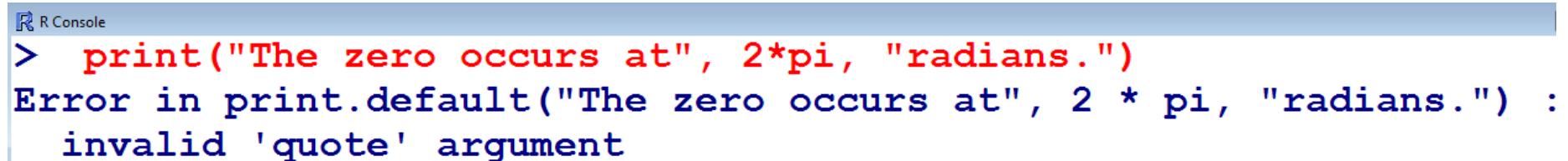
The zero occurs at  $2\pi$  radians.

# Formatting and Display of Strings

The `print` function has a significant limitation that it prints only one object at a time.

Trying to print multiple items gives error message:

```
> print("The zero occurs at", 2*pi, "radians.")  
Error in print.default("The zero occurs at", 2 *  
pi, "radians.") : invalid 'quote' argument
```



A screenshot of an R console window titled "R Console". The window shows a single line of R code: `> print("The zero occurs at", 2*pi, "radians.")`. Below the code, an error message is displayed in red text: `Error in print.default("The zero occurs at", 2 * pi, "radians.") : invalid 'quote' argument`.

# Formatting and Display of Strings

The only way to print multiple items is to print them one at a time

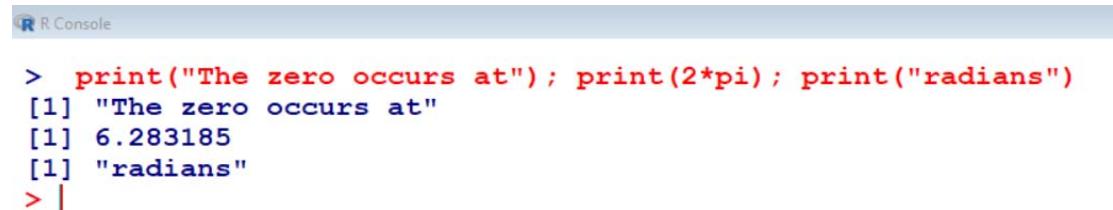
```
> print("The zero occurs at"); print(2*pi);
```

```
print("radians")
```

```
[1] "The zero occurs at"
```

```
[1] 6.283185
```

```
[1] "radians"
```



R Console

```
> print("The zero occurs at"); print(2*pi); print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
> |
```

The `cat` function is an alternative to `print` that lets you combine multiple items into a continuous output.

## Formatting and Display of Strings : cat function

- The function **cat( )** concatenate (link in the same sequence) and print the arguments in strings, concatenates them and prints the entire string in the command window.
- **cat** is useful for producing output in user-defined functions.
- It converts its arguments to character vectors, link them together in the same sequence to a single character vector, appends the given **sep = string(s)** to each element and then outputs them.

# Formatting and Display of Strings : cat function

## Usage

```
cat(...)  
cat(..., file = "", sep = " ", fill = FALSE,  
labels = NULL, append = FALSE)
```

**cat** puts a space between each item by default.

One must provide a newline character (`\n`) (newline) to terminate the line.

## Formatting and Display of Strings : cat function

**fill** : a logical or (positive) numeric controlling how the output is broken into successive lines. If **FALSE** (default), only newlines created explicitly by "**\n**" are printed. Otherwise, the output is broken into lines with print width equal to the option width if fill is **TRUE**, or the value of fill if this is numeric. Non-positive fill values are ignored, with a warning.

**labels** : character vector of labels for the lines printed. Ignored if fill is **FALSE**.

**append** : logical. Only used if the argument file is the name of file. If **TRUE** output will be appended to file; otherwise, it will overwrite the contents of file.

# Formatting and Display of Strings: print vs cat

The only way to print multiple items is to print them one at a time

```
> print("The zero occurs at"); print(2*pi);
print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
```

The **cat** function is an alternative to print that lets you combine multiple items into a continuous output:

```
> cat("The zero occurs at", 2*pi, "radians.",
"\n")
The zero occurs at 6.283185 radians.
```

# Formatting and Display of Strings

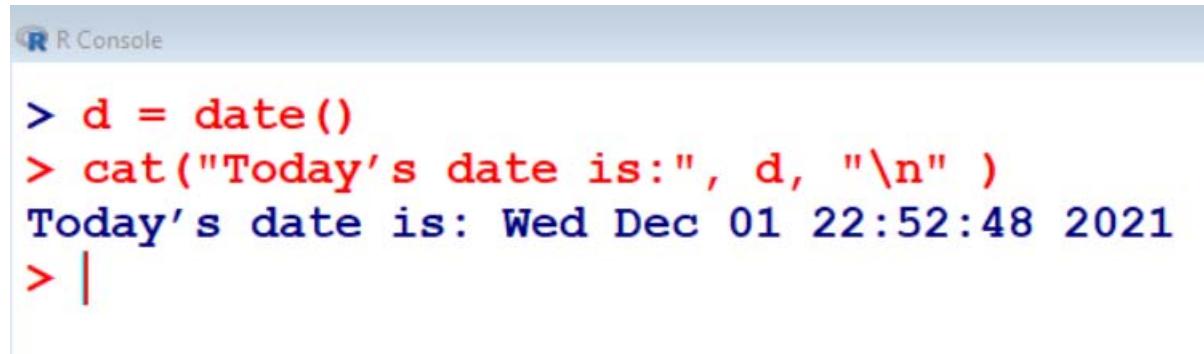
```
R Console
> print("The zero occurs at"); print(2*pi); print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
>
> cat("The zero occurs at", 2*pi, "radians.", "\n")
The zero occurs at 6.283185 radians.
```

# Formatting and Display of Strings

```
> d = date()
```

```
> cat("Today's date is:", d, "\n" )
```

```
Today's date is: Wed Dec 01 22:52:48 2021
```



The screenshot shows the R Console interface. The title bar says "R Console". The main area contains the following R code and its output:

```
> d = date()
> cat("Today's date is:", d, "\n" )
Today's date is: Wed Dec 01 22:52:48 2021
> |
```

[Please note this is the time at the time or preparation of slide]

# Formatting and Display of Strings

```
> x = 1:10  
> x  
[1] 1 2 3 4 5 6 7 8 9 10  
  
> cat(x, sep = " ++ ")  
1 ++ 2 ++ 3 ++ 4 ++ 5 ++ 6 ++ 7 ++ 8 ++ 9 ++ 10  
  
> cat("\n")      # This is used to change the line  
  
> cat(x, sep = " / ")  
1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 10
```

# Formatting and Display of Strings

R Console

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> cat(x, sep = " ++ ")
1 ++ 2 ++ 3 ++ 4 ++ 5 ++ 6 ++ 7 ++ 8 ++ 9 ++ 10>
> cat("\n")

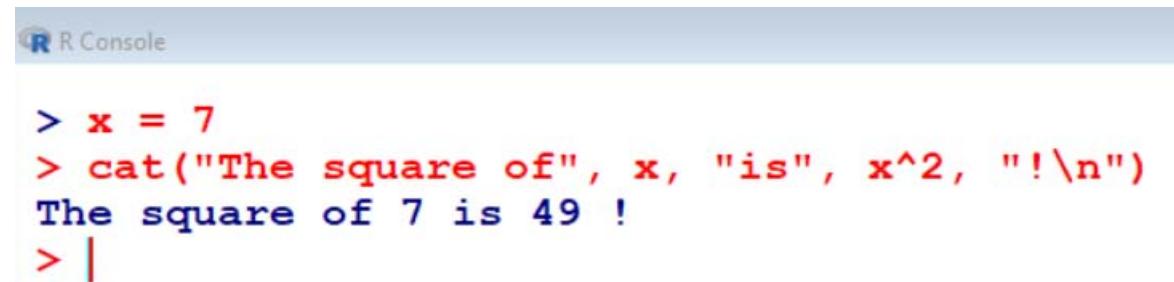
>
> cat(x, sep = " / ")
1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 10> |
```

# Formatting and Display of Strings

```
> x = 7
```

```
> cat("The square of", x, "is", x^2, "!\\n")
```

The square of 7 is 49 !



A screenshot of an R console window. The title bar says "R Console". The main area contains the following text:

```
> x = 7
> cat("The square of", x, "is", x^2, "!\\n")
The square of 7 is 49 !
> |
```

# Formatting and Display of Strings

```
> cat("The square root of",x,"is  
approximately", format(sqrt(x),digits=3),"\n")
```

The square root of 7 is approximately 2.65

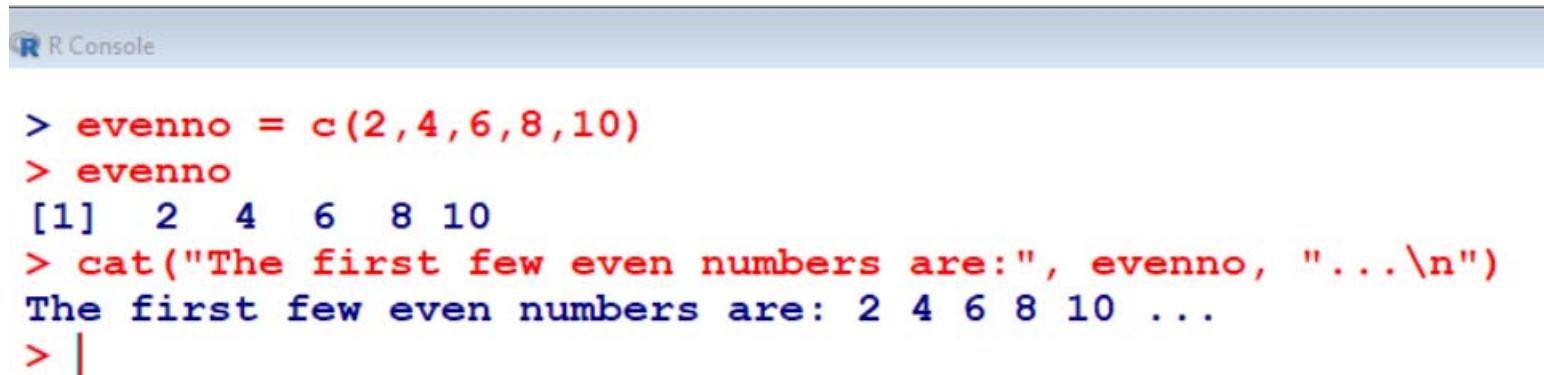
```
R Console  
> cat("The square root of",x,"is approximately", format(sqrt(x),digits=3),"\n")  
The square root of 7 is approximately 2.65
```

# Formatting and Display of Strings

The `cat` function can also print simple vectors

```
> evenno = c(2,4,6,8,10)
> evenno
[1] 2 4 6 8 10
```

```
> cat("The first few even numbers are:",
evenno, "...\\n")
The first few even numbers are: 2 4 6 8 10 ...
```



A screenshot of the R console interface. The title bar says "R Console". The main area contains the following R code and its output:

```
> evenno = c(2,4,6,8,10)
> evenno
[1] 2 4 6 8 10
> cat("The first few even numbers are:", evenno, "...\\n")
The first few even numbers are: 2 4 6 8 10 ...
> |
```

# Formatting and Display of Strings

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10

> cat(x, fill = 2, labels = paste("( ", letters[1:10], "):"))
( a ): 1
( b ): 2
( c ): 3
( d ): 4
( e ): 5
( f ): 6
( g ): 7
( h ): 8
( i ): 9
( j ): 10
```

# **Foundations of R Software**

**Lecture 37**

## **Strings – Display and Formatting**

**---**

### **Paste Function**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Formatting and Display of Strings: `paste`

- The `paste()` function concatenates several strings together.
- It creates a new string by joining the given strings end to end.
- The result of `paste()` can be assigned to a variable  
(in contrast to the function `cat()`).

# Formatting and Display of Strings

## Usage

```
paste (..., sep = " ", collapse = NULL).
```

**collapse** is an optional character string to separate the results.

- The parameter **sep** is a string that serves as a separation between the strings that are given as input.
- **paste** inserts a single space between pairs of strings.
- A desired line break can be achieved with "**\n**" (newline).

## Formatting and Display of Strings: `paste`

- `paste` converts its arguments to character strings (via `as.character`) , and concatenates them (separating them by the string given by `sep`).
- If the arguments are vectors, they are concatenated term-by-term to give a character vector result.
- If a value is specified for `collapse`, the values in the result are then concatenated into a single string, with the elements being separated by the value of `collapse`.

# Formatting and Display of Strings: `paste`

- The `paste()` function concatenates several strings together.
- It creates a new string by joining the given strings end to end.
- The result of `paste()` can be assigned to a variable (in contrast to the function `cat()`).

`paste0(..., collapse)`

is equivalent to `paste(..., sep = "", collapse)`,  
slightly more efficiently.

# Formatting and Display of Strings

Example: `paste` converts its arguments to character strings (via `as.character`), and concatenates them

```
> paste(1:12)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
"9"   "10"  "11"  "12"
```

```
> as.character(1:12) #Alternative to paste
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
"9"   "10"  "11"  "12"
```



The screenshot shows the R console interface. The title bar says "R Console". The main area contains two lines of R code and their corresponding outputs. The first line is red and reads "`> paste(1:12)`". The second line is red and reads "`> as.character(1:12)`". Both lines have blue outputs below them: the first output is "[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12\"", and the second output is "[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12\". A cursor is visible at the end of the second line of code.

```
> paste(1:12)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
"9"   "10"  "11"  "12"
>
> as.character(1:12)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
"9"   "10"  "11"  "12"
> |
```

# Formatting and Display of Strings

```
> paste("Everybody", "loves", "R Programming.")
```

```
[1] "Everybody loves R Programming."
```

```
> paste("Everybody", "loves", "R Programming.",  
sep="*")
```

```
[1] "Everybody*loves*R Programming."
```

```
> paste("Everybody", "loves", "R Programming.",  
sep="====")
```

```
[1] "Everybody====loves====R Programming."
```

# Formatting and Display of Strings

```
R Console
> paste("Everybody", "loves", "R Programming.")
[1] "Everybody loves R Programming."
>
> paste("Everybody", "loves", "R Programming.", sep="*")
[1] "Everybody*loves*R Programming."
>
> paste("Everybody", "loves", "R Programming.", sep="====")
[1] "Everybody====loves====R Programming."
```

# Formatting and Display of Strings

If one or more arguments are vectors of strings, `paste` will generate all combinations of the arguments:

```
> names = c("Prof. Singh", "Mr. Venkat", "Dr. Jha")
```

```
> names  
[1] "Prof. Singh" "Mr. Venkat" "Dr. Jha"
```

```
> paste(names, "is", "a good", "person.")  
[1] "Prof. Singh is a good person." "Mr. Venkat is a  
good person." "Dr. Jha is a good person."
```

```
R Console  
> names = c("Prof. Singh", "Mr. Venkat", "Dr. Jha")  
> names  
[1] "Prof. Singh" "Mr. Venkat" "Dr. Jha"  
> paste(names, "is", "a good", "person.")  
[1] "Prof. Singh is a good person." "Mr. Venkat is a good person." "Dr. Jha is a good person."  
> |
```

# Formatting and Display of Strings

When we want to join even those combinations into one, big string.

The `collapse` parameter defines a top-level separator and instructs `paste` to concatenate the generated strings using that separator:

```
> names = c("Prof. Singh", "Mr. Venkat", "Dr.  
Jha")  
  
> paste(names, "is", "a good", "person.",  
collapse=", and ")  
[1] "Prof. Singh is a good person., and Mr.  
Venkat is a good person., and Dr. Jha is a good  
person."
```

# Formatting and Display of Strings

R Console

```
> names = c("Prof. Singh", "Mr. Venkat", "Dr. Jha")
>
> paste(names, "is", "a good", "person.", collapse=", and ")
[1] "Prof. Singh is a good person., and Mr. Venkat is a good person., and Dr. Jha is a good person."
> |
```

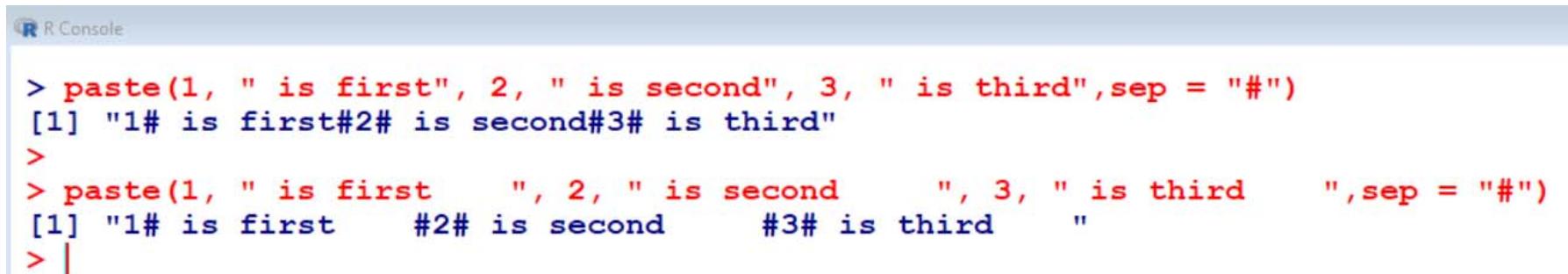
# Operations with Strings

Example:

```
> paste(1, " is first", 2, " is second", 3, " is third",
sep = "#")
[1] "1# is first#2# is second#3# is third"
```

Observe the role of blank space inside double quotes

```
> paste(1, " is first      ", 2, " is second      ", 3, " is
third      ", sep = "#")
[1] "1# is first      #2# is second      #3# is third      "
```



The screenshot shows the R Console window with two examples of the `paste()` function. The first example uses single quotes for all components and a double quote for the separator, resulting in three separate strings with '#' separators. The second example uses double quotes for all components and a single quote for the separator, resulting in a single string where the components are separated by the single quote.

```
R Console

> paste(1, " is first", 2, " is second", 3, " is third",sep = "#")
[1] "1# is first#2# is second#3# is third"
>
> paste(1, " is first      ", 2, " is second      ", 3, " is third      ",sep = "#")
[1] "1# is first      #2# is second      #3# is third      "
```

# Operations with Strings

Example:

```
> x = paste("Ex", 1:5, sep="_")
>x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"

> x[1]
[1] "Ex_1"

> x[2]
[1] "Ex_2"

> x[3]
[1] "Ex_3"

> x[5]
[1] "Ex_5"
```

R Console

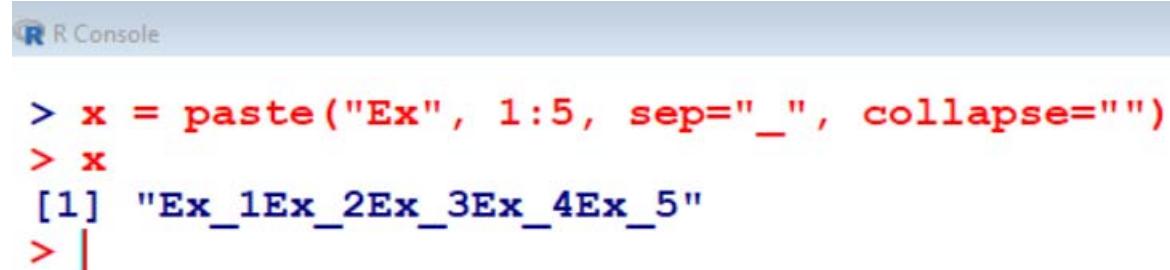
```
> x = paste("Ex", 1:5, sep="_")
>x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
>x[1]
[1] "Ex_1"
>x[2]
[1] "Ex_2"
>x[3]
[1] "Ex_3"
>x[4]
[1] "Ex_4"
>x[5]
[1] "Ex_5"
> |
```

# Operations with Strings

**x** is a vector of strings.

If we use the parameter **collapse**, a single string, instead of a vector of strings, is created:

```
> x = paste("Ex", 1:5, sep="_", collapse="")
> x[1]
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
```



The screenshot shows the R console interface. The title bar says "R Console". The main area contains the R code and its output. The code is identical to the one shown above: `> x = paste("Ex", 1:5, sep="_", collapse="")`, `> x`, and `[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"`. A cursor is visible at the end of the third line of code.

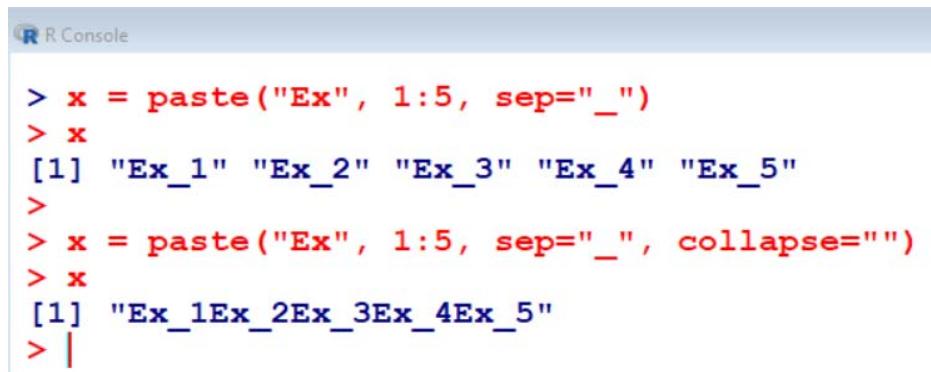
# Operations with Strings

Note the difference between

```
x = paste("Ex", 1:5, sep="_")
> x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
```

and

```
x = paste("Ex", 1:5, sep="_", collapse="")
> x
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
```



A screenshot of an R console window titled "R Console". It displays two separate R code sessions. The first session shows the use of the `sep` argument without `collapse`, resulting in a vector of five strings separated by underscores. The second session shows the use of both `sep` and `collapse`, resulting in a single string where all five original strings are concatenated together.

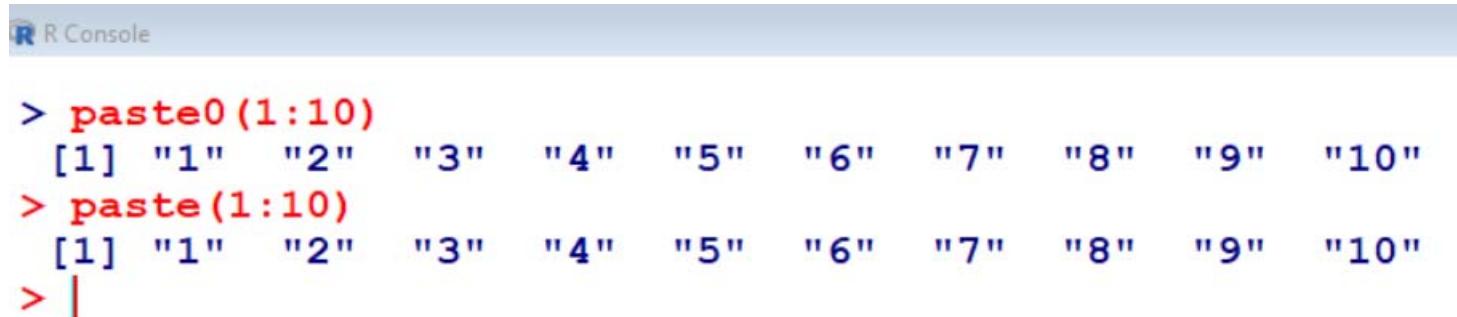
```
> x = paste("Ex", 1:5, sep="_")
> x
[1] "Ex_1" "Ex_2" "Ex_3" "Ex_4" "Ex_5"
>
> x = paste("Ex", 1:5, sep="_", collapse="")
> x
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
> |
```

# Operations with Strings

When using a single vector, `paste0` and `paste` have the same outcome and they work the same `as.character`.

```
> paste0(1:10)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
"9"   "10"
```

```
> paste(1:10)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
"9"   "10"
```



R Console

```
> paste0(1:10)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
> paste(1:10)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
> |
```

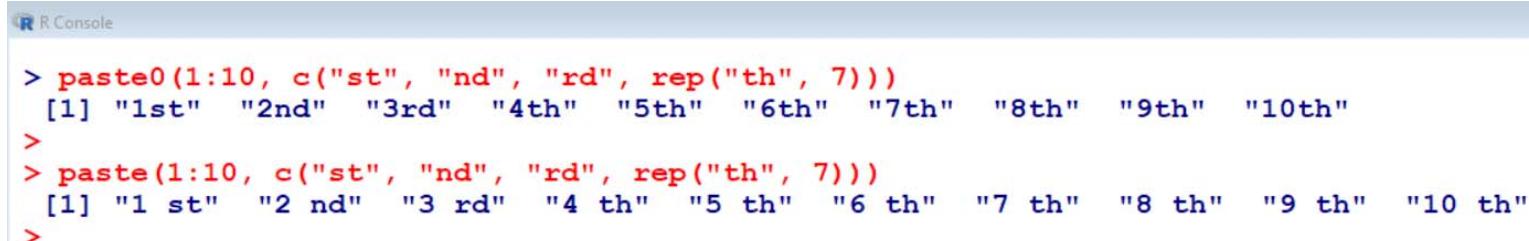
# Operations with Strings

When we use more than one vectors to `paste0` then they concatenate in a vectorized way.

```
> paste0(1:10, c("st", "nd", "rd", rep("th", 7)))  
[1] "1st"   "2nd"   "3rd"   "4th"   "5th"   "6th"   "7th"  
"8th"   "9th"   "10th"
```

Observe the role of blank space

```
> paste(1:10, c("st", "nd", "rd", rep("th", 7)))  
[1] "1 st"  "2 nd"  "3 rd"  "4 th"  "5 th"  "6 th"  
"7 th"  "8 th"  "9 th"  "10 th"
```



```
R Console  
> paste0(1:10, c("st", "nd", "rd", rep("th", 7)))  
[1] "1st"   "2nd"   "3rd"   "4th"   "5th"   "6th"   "7th"  
"8th"   "9th"   "10th"  
>  
> paste(1:10, c("st", "nd", "rd", rep("th", 7)))  
[1] "1 st"  "2 nd"  "3 rd"  "4 th"  "5 th"  "6 th"  
"7 th"  "8 th"  "9 th"  "10 th"  
>
```

# **Foundations of R Software**

**Lecture 38**

## **Strings – Display and Formatting**

---

### **String Splitting**

Shalabh

Department of Mathematics and Statistics

Indian Institute of Technology Kanpur

# Operations with Strings

Command `strsplit`, split the elements of a character vector.

"Split" can be a single character, or a character string:

Usage

```
strsplit(x, split, fixed = FALSE, ...)
```

Arguments

**x** character vector, each element of which is to be split.

**split** character vector containing regular expression(s)  
(unless `fixed = TRUE`) to use for splitting.

**fixed** logical. If `TRUE` match split exactly, otherwise use regular expressions.

# Operations with Strings

- With a command `strsplit`, we can split a string in pieces.

```
> x = "The&!syntax&!of&!paste&!is!&available!
&inthe online-help"
```

```
> x
[1] "The&!syntax&!of&!paste&!is!&available!
&inthe online-help"
```

```
> strsplit(x, split="!")
[[1]]
[1] "The&"           "syntax&"      "of&"
[4] "paste&"          "is"           "&available"
[7] "&inthe online-help"
```

# Operations with Strings

```
R Console

> x = "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
> x
[1] "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
> strsplit(x, split="!")
[[1]]
[1] "The&"           "syntax&"          "of&"
[4] "paste&"          "is"              "&available"
[7] "&inthe online-help"
```

# Operations with Strings

```
> x = "The&!syntax&!of&!paste&!is!&available!
&inthe online-help"

> x

[1] "The&!syntax&!of&!paste&!is!&available!
&inthe online-help"

> strsplit(x, split = "&!")

[[1]]

[1] "The"
[2] "syntax"
[3] "of"
[4] "paste"
[5] "is!&available! &inthe online-help"
```

# Operations with Strings

```
R R Console

> x = "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
> x
[1] "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
> strsplit(x, split = "&!")
[[1]]
[1] "The"
[2] "syntax"
[3] "of"
[4] "paste"
[5] "is!&available! &inthe online-help"

. . .
```

# Operations with Strings

```
> x = "The&!syntax&!of&!paste&!is!&available!
&inthe online-help"

> x
[1] "The&!syntax&!of&!paste&!is!&available!
&inthe online-help"

> y = strsplit(x, split = "!&")
[1] "The&!syntax&!of&!paste&!is"      "available!
&inthe online-help"
```

# Operations with Strings

Notice the access to single components:

```
> y[[1]][1]
```

```
[1] "The&!syntax&!of&!paste&!is"
```

```
> y[[1]][2]
```

```
[1] "available! &inthe online-help"
```

```
> y[[1]][3]
```

```
[1] NA
```

# Operations with Strings

```
R R Console

> x = "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
> x
[1] "The&!syntax&!of&!paste&!is!&available! &inthe online-help"
> y = strsplit(x, split = "!&")
> y
[[1]]
[1] "The&!syntax&!of&!paste&!is"      "available! &inthe online-help"

> y[[1]][1]
[1] "The&!syntax&!of&!paste&!is"
> y[[1]][2]
[1] "available! &inthe online-help"
> y[[1]][3]
[1] NA
> |
```

# Operations with Strings

Example: Convert to matrix

```
> dates = c("2020-07-24", "2021-08-25", "2022-  
09-26", "2023-10-27")
```

Split the dates

```
> datesplt = strsplit(dates, "-")  
> datesplt  
[[1]]  
[1] "2020" "07"     "24"  
  
[[2]]  
[1] "2021" "08"     "25"  
  
[[3]]  
[1] "2022" "09"     "26"  
  
[[4]]  
[1] "2023" "10"     "27"
```

# Operations with Strings

Example: Convert to matrix

Create matrix of dates and outcome is that the elements are character

```
> datemat = matrix(unlist(datesplt), nrow = 4,  
ncol=3, byrow=TRUE)  
  
> datemat  
      [,1]  [,2]  [,3]  
[1,] "2020" "07"  "24"  
[2,] "2021" "08"  "25"  
[3,] "2022" "09"  "26"  
[4,] "2023" "10"  "27"
```

# Operations with Strings

Example: Convert to matrix

Create matrix of dates and outcome is that the elements are numbers

```
> datematrix =  
matrix(as.numeric(unlist(datesplt))), nrow = 4,  
ncol=3, byrow=TRUE)  
  
> datematrix  
     [,1] [,2] [,3]  
[1,] 2020    7    24  
[2,] 2021    8    25  
[3,] 2022    9    26  
[4,] 2023   10    27
```

# Operations with Strings

## Example: Convert to matrix

```
R Console
```

```
> dates = c("2020-07-24", "2021-08-25", "2022-09-26", "2023-10-27")
> dates
[1] "2020-07-24" "2021-08-25" "2022-09-26" "2023-10-27"
> datesplt = strsplit(dates, "-")
> datesplt
[[1]]
[1] "2020" "07"   "24"

[[2]]
[1] "2021" "08"   "25"

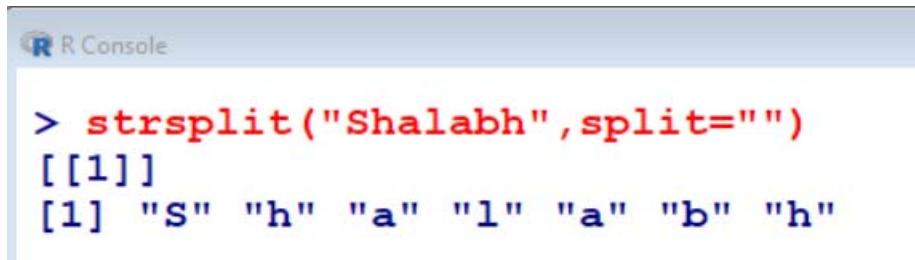
[[3]]
[1] "2022" "09"   "26"

[[4]]
[1] "2023" "10"   "27"

> datemat = matrix(unlist(datesplt), nrow = 4, ncol=3, byrow=TRUE)
> datemat
 [,1]  [,2]  [,3]
[1,] "2020" "07"  "24"
[2,] "2021" "08"  "25"
[3,] "2022" "09"  "26"
[4,] "2023" "10"  "27"
> datematrix = matrix(as.numeric(unlist(datesplt)), nrow = 4, ncol=3, byrow=TRUE)
> datematrix
 [,1] [,2] [,3]
[1,] 2020    7   24
[2,] 2021    8   25
[3,] 2022    9   26
[4,] 2023   10   27
> |
```

# Operations with Strings

```
> strsplit("Shalabh",split="")  
[[1]]  
[1] "S" "h" "a" "l" "a" "b" "h"
```



A screenshot of an R console window titled "R Console". The window shows the command `> strsplit("Shalabh",split="")` in red, followed by the output `[[1]]` and `[1] "S" "h" "a" "l" "a" "b" "h"`. The background of the console is light grey.

# **Foundations of R Software**

**Lecture 39**

## **Strings – Display and Formatting**

**:::**

## **Manipulations with Strings and Alphabets**

**Shalabh**

**Department of Mathematics and Statistics**

**Indian Institute of Technology Kanpur**

# Operations with Strings: Counting characters

**nchar** takes a character vector as an argument and returns a vector whose elements contain the sizes of the corresponding elements of **x**.

**nzchar** is a fast way to find out if elements of a character vector are non-empty strings.

# Operations with Strings: Counting characters

## Usage

```
nchar(x, type = "chars", allowNA = FALSE,
```

```
keepNA = NA)
```

```
nzchar(x, keepNA = FALSE)
```

## Arguments

**x** character vector, or a vector to be coerced to a character vector.

**type** character string: partial matching to one of `c("bytes", "chars", "width")`.

**keepNA** logical: should `NA` be returned when **x** is `NA`?

# Operations with Strings: Counting characters

There are a variety of commands that can be used for strings.

Example:

Count of number of characters:

```
> x = "R course 24.07.2022"  
  
> y = "Number of participants: 25"  
  
> nchar(x) #Count the Number of Characters in x  
[1] 19  
  
> nchar(y) #Count the Number of Characters in y  
[1] 26
```

# Operations with Strings: Counting characters

Example:

The `nzchar(x)` function takes the character vector `x` as a parameter. Its output is either `TRUE` or `FALSE`.

```
> x = "R course 24.07.2022"  
  
> y = "Number of participants: 25"  
  
> nzchar(x)  
[1] TRUE  
  
> nzchar(y)  
[1] TRUE
```

# Operations with Strings: Counting characters

```
R R Console
> x = "R course 24.07.2022"
> x
[1] "R course 24.07.2022"
> y = "Number of participants: 25"
> y
[1] "Number of participants: 25"
> nchar(x)
[1] 19
> nchar(y)
[1] 26
>
> nzchar(x)
[1] TRUE
> nzchar(y)
[1] TRUE
> |
```

# Operations with Strings: Counting characters

Example: `nzchar(x)`

```
> x = c("Apple", "Banana", "Cake")
```

```
> x
```

```
[1] "Apple"  "Banana" "Cake"
```

```
> nzchar(x)
```

```
[1] TRUE TRUE TRUE
```

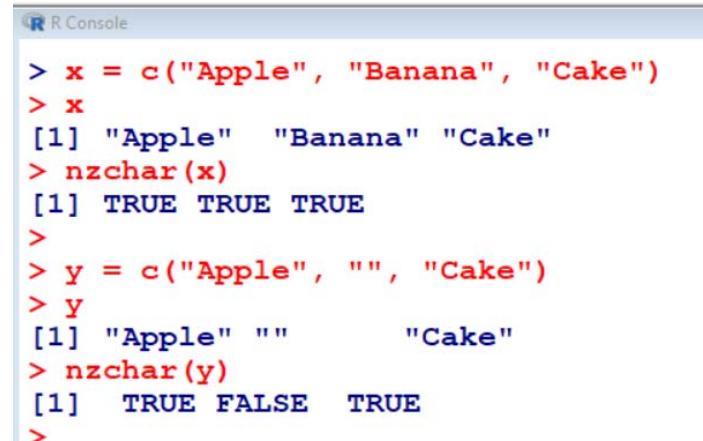
```
> y = c("Apple", "", "Cake")
```

```
> y
```

```
[1] "Apple"  ""          "Cake"
```

```
> nzchar(y)
```

```
[1] TRUE FALSE  TRUE
```



The screenshot shows an R console window with the title 'R Console'. It displays the following R session:

```
> x = c("Apple", "Banana", "Cake")
> x
[1] "Apple"  "Banana" "Cake"
> nzchar(x)
[1] TRUE TRUE TRUE
>
> y = c("Apple", "", "Cake")
> y
[1] "Apple"  ""          "Cake"
> nzchar(y)
[1] TRUE FALSE  TRUE
>
```

# Operations with Strings: Counting characters

Example:

```
> x = c("Apple", "Banana", "Cake")
```

```
> nchar(x)
```

```
[1] 5 6 4
```

```
> y = c(2, 4, 6)
```

```
> nchar(y)
```

```
[1] 1 1 1
```

```
> z = c(11, 222, 3333)
```

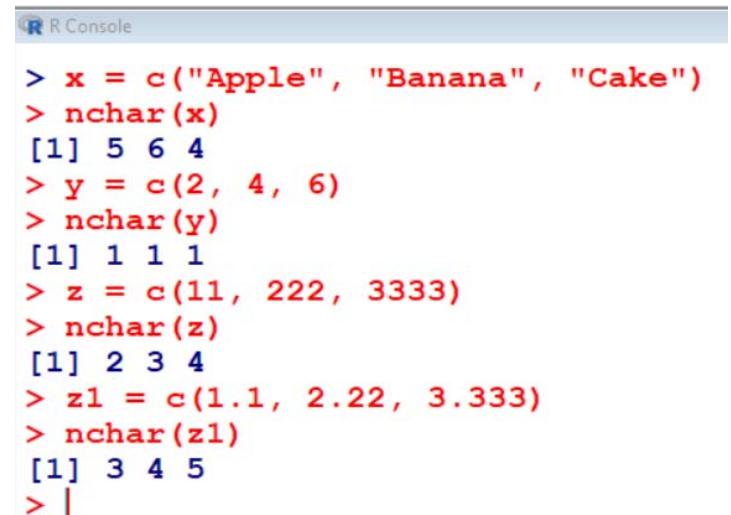
```
> nchar(z)
```

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

```
> z1 = c(1.1, 2.22, 3.333)
```

```
> nchar(z1)
```

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



```
R Console
> x = c("Apple", "Banana", "Cake")
> nchar(x)
[1] 5 6 4
> y = c(2, 4, 6)
> nchar(y)
[1] 1 1 1
> z = c(11, 222, 3333)
> nchar(z)
[1] 2 3 4
> z1 = c(1.1, 2.22, 3.333)
> nchar(z1)
[1] 3 4 5
>
```

# **Operations with Strings: Lower and upper cases**

**tolower(x) and toupper(x) Functions:**

**tolower(x)** and **toupper(x)** convert upper-case  
characters in a character vector to lower-case, or vice versa.

Non-alphabetic characters are left unchanged.

# Operations with Strings: Lower and upper cases

`tolower(x)` and `toupper(x)` Functions:

Examples:

```
> x = "R course will start from 24.07.2022"
```

```
> toupper(x)
```

```
[1] "R COURSE WILL START FROM 24.07.2022"
```

```
> z = "INDIAN INSTITUTE OF TECHNOLOGY"
```

```
> tolower(z)
```

```
[1] "indian institute of technology"
```

# Operations with Strings: Lower and upper cases

```
R R Console
> x = "R course will start from 24.07.2022"
> x
[1] "R course will start from 24.07.2022"
> toupper(x)
[1] "R COURSE WILL START FROM 24.07.2022"
>
> z = "INDIAN INSTITUTE OF TECHNOLOGY"
> z
[1] "INDIAN INSTITUTE OF TECHNOLOGY"
> tolower(z)
[1] "indian institute of technology"
> |
```

# **Foundations of R Software**

**Lecture 40**

## **Strings – Display and Formatting**

**⋮⋮⋮**

## **Substitution and Replacement of Strings**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Operations with Strings: Substitution

**sub** and **gsub** Functions:

Within a string, we want to replace one substring with another.

Use **sub** and **gsub** to replace the first instance of a substring:

**sub(old, new, string)**

The **sub** function finds the first instance of the old substring within string and replaces it with the new substring.

**gsub** does the same thing, but it replaces all instances of the substring (a global replace), not just the first.

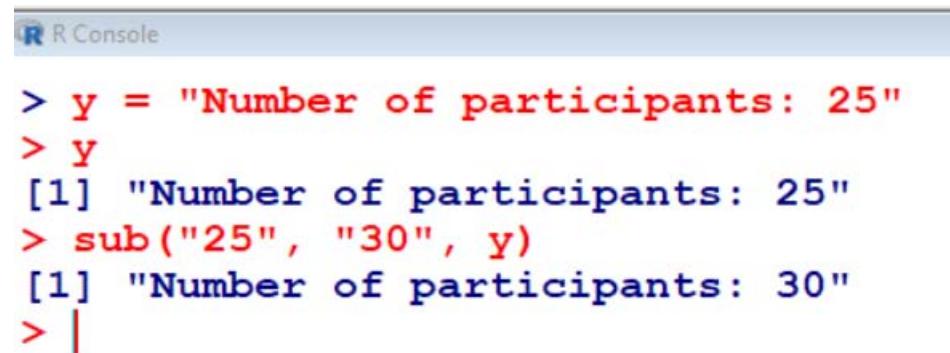
**gsub(old, new, string)**

# Operations with Strings: Substitution

Example:

```
> y = "Number of participants: 25"
```

```
> sub("25", "30", y)
[1] "Number of participants: 30"
```



A screenshot of an R console window titled "R Console". The window contains the following R code and its output:

```
> y = "Number of participants: 25"
> y
[1] "Number of participants: 25"
> sub("25", "30", y)
[1] "Number of participants: 30"
> |
```

# Operations with Strings: Substitution

Example:

```
> y = "Mr. Singh is the smart one. Mr. Singh  
is funny, too."
```

```
> y
```

```
[1] "Mr. Singh is the smart one. Mr. Singh is  
funny, too."
```

```
> sub("Mr. Singh", "Professor Jha", y)
```

```
[1] "Professor Jha is the smart one. Mr. Singh  
is funny, too."
```

# Operations with Strings: Substitution

```
R Console
> y = "Mr. Singh is the smart one. Mr. Singh is funny, too."
> y
[1] "Mr. Singh is the smart one. Mr. Singh is funny, too."
> sub("Mr. Singh","Professor Jha", y)
[1] "Professor Jha is the smart one. Mr. Singh is funny, too."
> |
```

# Operations with Strings: Substitution

Example:

```
> y = "Mr. Singh is the smart one. Mr. Singh  
is funny, too."  
  
> gsub("Mr. Singh", "Professor Jha", y)  
[1] "Professor Jha is the smart one. Professor  
Jha is funny, too."
```

Recall

```
> sub("Mr. Singh", "Professor Jha", y)  
[1] "Professor Jha is the smart one. Mr. Singh  
is funny, too."
```

# Operations with Strings: Substitution

```
R Console
> y = "Mr. Singh is the smart one. Mr. Singh is funny, too."
> y
[1] "Mr. Singh is the smart one. Mr. Singh is funny, too."
> gsub("Mr. Singh","Professor Jha", y)
[1] "Professor Jha is the smart one. Professor Jha is funny, too."
>
> sub("Mr. Singh","Professor Jha", y)
[1] "Professor Jha is the smart one. Mr. Singh is funny, too."
> |
```

# Operations with Strings

R has various functions for regular expression based match and replaces.

Some functions (e.g., `grep`, `grepl`, etc.) are used for searching for matches and functions whereas `sub` and `gsub` are used for performing replacement.

# Operations with Strings: grep

The `grep` function is used for searching the matches.

( `sub` and `gsub` are used for performing replacement. )

`grep` : Globally search regular expression and print it

`grep(pattern, x, ignore.case = FALSE)` search for matches to argument `pattern` within each element of a character vector `x`.

It returns an integer vector of the indices of the elements of `x` that yielded a match.

# Operations with Strings: grep

`ignore.case` : if `FALSE`, the pattern matching is case sensitive  
and if `TRUE`, case is ignored during matching.

`value` : if `FALSE`, a vector containing the (integer) indices of the  
matches determined by `grep` is returned, and if `TRUE`, a vector  
containing the matching elements themselves is returned.

# Operations with Strings: grep

`grep(pattern, x, value = TRUE)` returns a character vector containing the selected elements of x.

```
> str = c("R Course", "exercises", "include  
examples of R language")  
  
> grep("ex", str, value=T)  
[1] "exercises"  "include examples of R  
language"
```

# Operations with Strings: grep

`grep(pattern, x, value = FALSE)` returns an integer

vector of the indices of the elements of x that yielded a match

`value = FALSE` is default.

```
> str = c("R Course", "exercises", "include  
examples of R language")
```

```
> grep("ex", str, value=F)
```

```
[1] 2 3
```

# Operations with Strings: grep

```
R Console
> str = c("R Course", "exercises", "include examples of R language")
> grep("ex", str, value=T)
[1] "exercises"                  "include examples of R language"
>
> str = c("R Course", "exercises", "include examples of R language")
> grep("ex", str, value=F)
[1] 2 3
> |
```

# Operations with Strings: grep

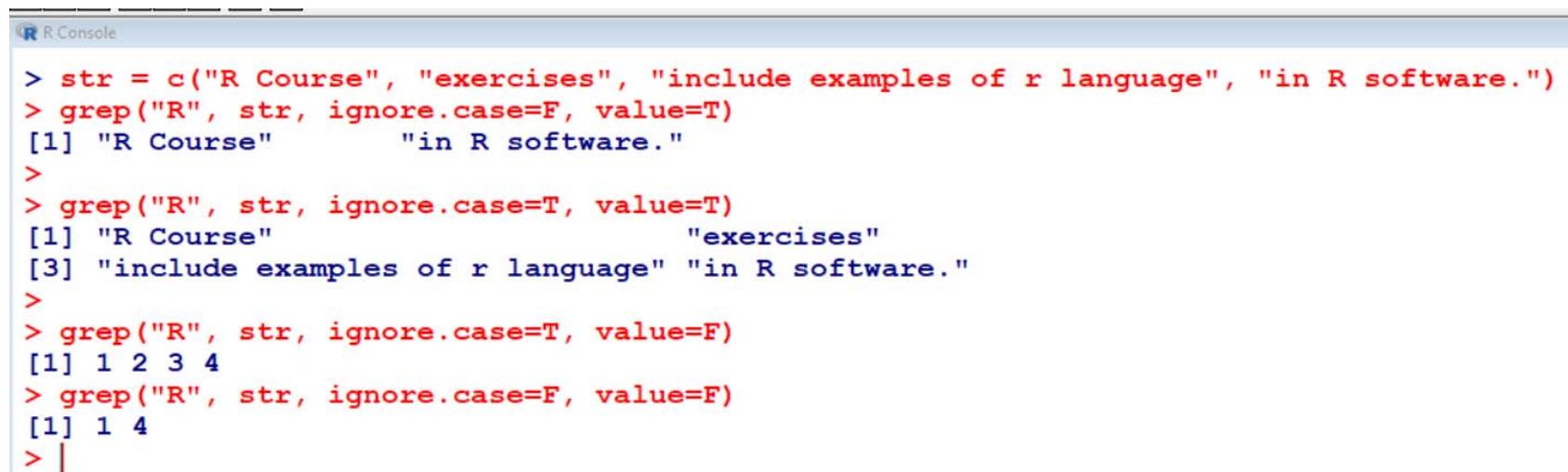
`grep(pattern, x, ignore.case = TRUE)` returns a character vector containing the selected elements of `x` ignoring the case.

```
> str = c("R Course", "exercises", "include  
examples of r language", "in R software.")  
  
> grep("R", str, ignore.case=F, value=T)  
[1] "R Course"           "in R software."  
  
> grep("R", str, ignore.case=T, value=T)  
[1] "R Course"           "exercises"  
[3] "include examples of r language" "in R software."
```

# Operations with Strings: grep

```
> grep("R", str, ignore.case=T, value=F)
[1] 1 2 3 4
```

```
> grep("R", str, ignore.case=F, value=F)
[1] 1 4
```



A screenshot of the R Console window. The title bar says "R Console". The console area contains the following R code and its output:

```
> str = c("R Course", "exercises", "include examples of r language", "in R software.")
> grep("R", str, ignore.case=F, value=T)
[1] "R Course"      "in R software."
>
> grep("R", str, ignore.case=T, value=T)
[1] "R Course"          "exercises"
[3] "include examples of r language" "in R software."
>
> grep("R", str, ignore.case=T, value=F)
[1] 1 2 3 4
> grep("R", str, ignore.case=F, value=F)
[1] 1 4
> |
```

# Operations with Strings: grep

Example:

```
> x = "R course 24.07.2021"  
  
> y = "Number of participants: 25"  
  
> c(x,y) # Combine the two strings  
[1] "R course 24.07.2021"  "Number of  
participants: 25"  
  
> grep("our", c(x,y) )  
[1] 1
```

"our" is in the 1st element (in the word "course"), therefore in x.  
There is no "our" in y.

# Operations with Strings: grep

**Example:**

```
x = "R course 24.07.2022"
y = "Number of participants: 50"

c(x,y) # Combine the two strings
[1] "R course 24.07.2021" "Number of
participants: 25"

> grep("Num", c(x,y) )
[1] 2
```

**"Num"** is in the 2nd element (in the word "**Number**"), therefore in y.

There is no "Num" in x.

# Operations with Strings: grep

grep function:

```
R R Console
> x = "R course 24.07.2022"
> y = "Number of participants: 50"
>
> c(x,y)
[1] "R course 24.07.2022"           "Number of participants: 50"
>
> grep("Num", c(x,y) )
[1] 2
~ |
```

# Operations with Strings: `grepl`

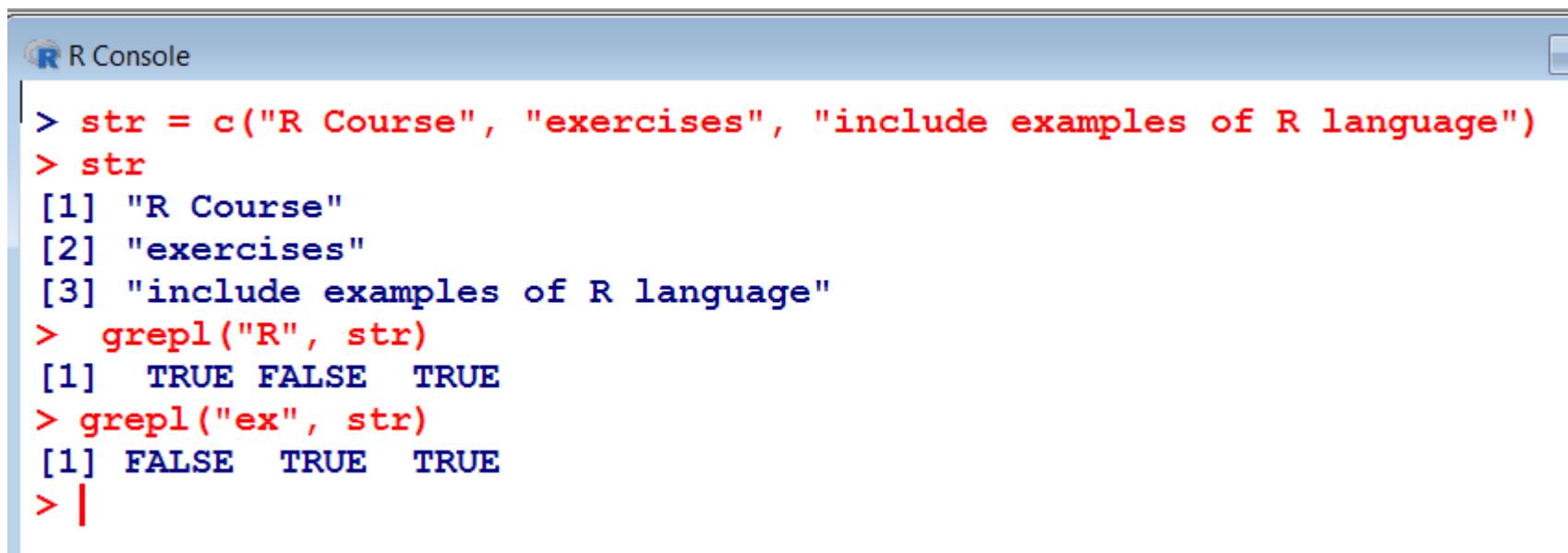
`grepl(pattern, x)` returns a character vector containing the selected elements of x and the outcome is in terms of `TRUE` and `FALSE`. Indicating if the matching is available or not.

```
> str = c("R Course", "exercises", "include  
examples of R language")  
  
> grepl("R", str)  
[1] TRUE FALSE  TRUE
```

# Operations with Strings: grepl

```
> str = c("R Course", "exercises", "include  
examples of R language")  
  
> grepl("ex", str, value=T)  
[1] FALSE  TRUE  TRUE
```

# Operations with Strings: grepl



R Console

```
> str = c("R Course", "exercises", "include examples of R language")
> str
[1] "R Course"
[2] "exercises"
[3] "include examples of R language"
> grepl("R", str)
[1] TRUE FALSE TRUE
> grepl("ex", str)
[1] FALSE TRUE TRUE
> |
```

# **Foundations of R Software**

## **Lecture 41**

## **Data Frames**

**Shalabh**

**Department of Mathematics and Statistics**  
**Indian Institute of Technology Kanpur**

# Data Frames

The commands `c`, `cbind`, `vector` and `matrix` functions combine data.

Another option is the data frame.

In a data frame, we can combine variables of equal length, with each row in the data frame containing observations on the same unit.

Hence, it is similar to the `matrix` or `cbind` functions.

Advantage is that one can make changes to the data without affecting the original data.

# Data Frames

One can also combine numerical variables, character strings as well as factors in data frame.

For example, `cbind` and `matrix` functions can not be used to combine different types of data

Data frames are special types of objects in R designed for data sets.

The data frame format is similar to a spreadsheet, where columns contain variables and observations are contained in rows.

# Data Frames

**Data frames contain complete data sets that are mostly created with other programs (spreadsheet-files, software SPSS-files, Excel-files etc.).**

**Variables in a data frame may be numeric (numbers) or categorical (characters or factors).**

# Data Frames

## Example:

Package “**MASS**” describes functions and datasets to support Venables and Ripley, ‘‘Modern Applied Statistics with S’’ (4<sup>th</sup> edition 2002)

# Data Frames

An example data frame `painters` is available in the library.

MASS (here only an excerpt of a data set):

```
> library(MASS)
```

```
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.

Here, the names of the painters serve as row identifications, i.e., every row is assigned to the name of the corresponding painter.

# Data Frames

R Console

```
> library(MASS)
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
	*	*	*	*	*
	*	*	*	*	*
	*	*	*	*	*
	*	*	*	*	*
Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H

# Data Frames

However, these names are not variables of the data set. Here a subset of these names:

```
> rownames(painters)
 [1] "Da Udine"          "Da Vinci"           "Del Piombo"
 [4] "Del Sarto"         "Fr. Penni"          "Guilio Romano"
 [7] "Michelangelo"      "Perino del Vaga"    "Perugino"
[10] "Raphael"           "F. Zuccaro"         "Fr. Salviata"
[13] "Parmigiano"        "Primaticcio"       "T. Zuccaro"
[16] "Volterra"          "Barocci"            "Cortona"
[19] "Josepin"            "L. Jordaeans"       "Testa"
[22] "Vanius"             "Bassano"            "Bellini"
[25] "Giorgione"          "Murillo"            "Palma Giovane"
[28] "Palma Vecchio"     "Pordenone"          "Tintoretto"
[31] "Titian"              "Veronese"           "Albani"
[34] "Caravaggio"         "Correggio"          "Domenichino"
[37] "Guercino"            "Lanfranco"          "The Carraci"
[40] "Durer"               "Holbein"            "Pourbus"
[43] "Van Leyden"          "Diepenbeck"         "J. Jordaeans"
[46] "Otho Venius"        "Rembrandt"          "Rubens"
[49] "Teniers"             "Van Dyck"            "Bourdon"
```

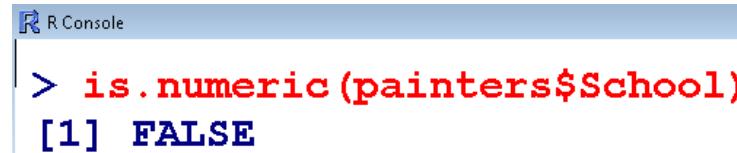
# Data Frames

```
R Console
> rownames(painters)
[1] "Da Udine"          "Da Vinci"           "Del Piombo"
[4] "Del Sarto"         "Fr. Penni"          "Guilio Romano"
[7] "Michelangelo"      "Perino del Vaga"   "Perugino"
[10] "Raphael"          "F. Zuccaro"        "Fr. Salviata"
[13] "Parmigiano"        "Primaticcio"       "T. Zuccaro"
[16] "Volterra"          "Barocci"            "Cortona"
[19] "Josepin"            "L. Jordaens"        "Testa"
[22] "Vanius"             "Bassano"            "Bellini"
[25] "Giorgione"          "Murillo"            "Palma Giovane"
[28] "Palma Vecchio"     "Pordenone"          "Tintoretto"
[31] "Titian"              "Veronese"           "Albani"
[34] "Caravaggio"          "Correggio"          "Domenichino"
[37] "Guercino"            "Lanfranco"          "The Carracci"
[40] "Durer"                "Holbein"            "Pourbus"
[43] "Van Leyden"          "Diepenbeck"         "J. Jordaens"
[46] "Otho Venius"         "Rembrandt"          "Rubens"
[49] "Teniers"              "Van Dyck"            "Bourdon"
```

# Data Frames

- The data set contains four numerical variables (Composition, Drawing, Colour and Expression), as well as one factor variable (School).

```
> is.numeric(painters$School)  
[1] FALSE
```



```
R Console  
> is.numeric(painters$School)  
[1] FALSE
```

Notice how we extract a variable (column) from data set.

```
> is.numeric(painters$Drawing)  
[1] TRUE
```

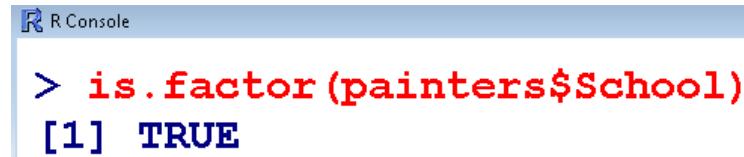


```
R Console  
> is.numeric(painters$Drawing)  
[1] TRUE
```

# Data Frames

- The data set contains four numerical variables (Composition, Drawing, Colour and Expression), as well as one factor variable (School).

```
> is.factor(painters$School)  
[1] TRUE
```



R Console  
> is.factor(painters\$School)  
[1] TRUE

A screenshot of an R console window titled "R Console". It shows the command "is.factor(painters\$School)" entered in red, followed by the output "[1] TRUE" in blue.

```
> is.factor(painters$Drawing)  
[1] FALSE
```



R Console  
> is.factor(painters\$Drawing)  
[1] FALSE

A screenshot of an R console window titled "R Console". It shows the command "is.factor(painters\$Drawing)" entered in red, followed by the output "[1] FALSE" in blue.

# Data Frames

```
> colnames(painters)
[1] "Composition" "Drawing" "Colour"
"Expression" "School"
```



A screenshot of an R console window titled "R Console". The window shows the command "colnames(painters)" entered in red, followed by its output in blue, which lists the column names: "Composition", "Drawing", "Colour", "Expression", and "School".

```
R Console
> colnames(painters)
[1] "Composition" "Drawing" "Colour"      "Expression"   "School"
```

# Data Frames

Using the **summary** function, we can get a quick overview of descriptive measures for each variable: (*We will learn later*).

```
> summary(painters)
```

Composition	Drawing	Colour	Expression	School
Min. : 0.00	Min. : 6.00	Min. : 0.00	Min. : 0.000	A : 10
1st Qu.: 8.25	1st Qu.:10.00	1st Qu.: 7.25	1st Qu.: 4.000	D : 10
Median :12.50	Median :13.50	Median :10.00	Median : 6.000	E : 7
Mean :11.56	Mean :12.46	Mean :10.94	Mean : 7.667	G : 7
3rd Qu.:15.00	3rd Qu.:15.00	3rd Qu.:16.00	3rd Qu.:11.500	B : 6
Max. :18.00	Max. :18.00	Max. :18.00	Max. :18.000	C : 6
				(Other): 8

The categories F and H, each present 4 times in the variable "School", are summed under the category Other as 8 with the corresponding frequency. i.e., only the 6 most frequent values are displayed.

# Data Frames

```
R R Console
> summary(painters)
      Composition      Drawing       Colour      Expression      School
Min.    : 0.00      Min.    : 6.00      Min.    : 0.00      Min.    : 0.000   A     :10
1st Qu.: 8.25      1st Qu.:10.00      1st Qu.: 7.25      1st Qu.: 4.000   D     :10
Median  :12.50      Median  :13.50      Median  :10.00      Median  : 6.000   E     : 7
Mean    :11.56      Mean    :12.46      Mean    :10.94      Mean    : 7.667   G     : 7
3rd Qu.:15.00      3rd Qu.:15.00      3rd Qu.:16.00      3rd Qu.:11.500   B     : 6
Max.    :18.00      Max.    :18.00      Max.    :18.00      Max.    :18.000   C     : 6
                                         (Other) : 8
```

# **Foundations of R Software**

**Lecture 42**

## **Data Frames: Creation and Operations**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Data Frames

An example data frame `painters` is available in the library MASS (here only an excerpt of a data set):

```
> library(MASS)
```

```
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.

Here, the names of the painters serve as row identifications, i.e., every row is assigned to the name of the corresponding painter.

# Data Frames

# **Foundations of R Software**

**Lecture 43**

## **Data Frames: Some More Operations**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Data Frames

An example data frame `painters` is available in the library MASS (here only an excerpt of a data set):

```
> library(MASS)
```

```
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.

Here, the names of the painters serve as row identifications, i.e., every row is assigned to the name of the corresponding painter.

# Data Frames

R Console

```
> library(MASS)
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
	*	*	*	*	*
	*	*	*	*	*
	*	*	*	*	*
	*	*	*	*	*
Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H

# Data Frames

The `summary` function for a categorical variable returns a detailed frequency table:

`summary` is a generic function used to produce result summaries of the results of various model fitting functions.

```
> summary(painters$School)
   A   B   C   D   E   F   G   H
10  6   6  10  7   4   7   4
```

R Console

```
> summary(painters$School)
   A   B   C   D   E   F   G   H
10  6   6  10  7   4   7   4
```

# Data Frames

## □ Attaching a data frame

With a command `attach( )` over the data frame, the variables can be referenced directly by name.

It can address the names of a data frame directly, without the prefix dollar sign operator, e.g. `painters$`.

### Example

```
> attach(painters)
```

Variable names are

- `Composition`,
- `Drawing`,
- `Colour`,
- `Expression`,
- `School`

# Data Frames

```
> summary(School) # Character variable
```

	A	B	C	D	E	F	G	H
10	6	6	10	7	4	7	4	

```
R Console  
> attach(painters)  
> summary(School)  
A B C D E F G H  
10 6 6 10 7 4 7 4
```

```
> summary(Composition) # Numeric variable
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.00	8.25	12.50	11.56	15.00	18.00	

```
R Console  
> summary(Composition)  
Min. 1st Qu. Median Mean 3rd Qu. Max.  
0.00 8.25 12.50 11.56 15.00 18.00
```

# Data Frames

- The command `detach()` recovers the default setting and then we have to use `painters$` again.

```
> detach(painters)

> summary(School)
Error in summary(School) : Object "School" not
found
```

```
R Console

> detach(painters)
> summary(School)
Error in summary(School) : object 'School' not found
```

# Data Frames

Subsets of a data frame can be obtained with `subset()` or with the second equivalent command:

```
> subset(painters, School=='F')
```

(# == means logical equal sign )

	Composition	Drawing	Colour	Expression	School
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
VanLeyden	8	6	6	4	F

# Data Frames

Similar outcome can be also obtained from

```
> painters[ painters[["School"]] == "F", ]
```

	Composition	Drawing	Colour	Expression	School
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
VanLeyden	8	6	6	4	F

```
R Console
> painters[ painters[["School"]] == "F", ]
      Composition Drawing Colour Expression School
Durer           8      10     10          8      F
Holbein         9      10     16         13      F
Pourbus         4      15      6          6      F
Van Leyden      8       6      6          4      F
```

# Data Frames

Subsets of a data frame can be obtained with `subset()` or with the second equivalent command:

```
> subset(painters, Composition <= 6)
```

```
R R Console
> subset(painters, Composition <= 6)
   Composition Drawing Colour Expression School
Fr. Penni          0      15     8          0      A
Perugino          4      12    10          4      A
Bassano           6      8    17          0      D
Bellini           4      6    14          0      D
Murillo           6      8    15          4      D
Palma Vecchio    5      6    16          0      D
Caravaggio        6      6    16          0      E
Pourbus          4      15     6          6      F
> █
```

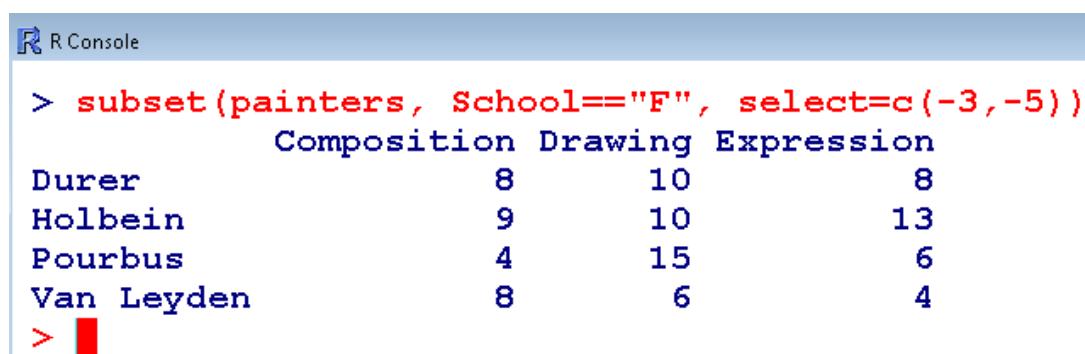
# Data Frames

- Uninteresting columns can be eliminated.

```
> subset(painters, School=="F", select=c(-3,-5))
```

	Composition	Drawing	Expression
Durer	8	10	8
Holbein	9	10	13
Pourbus	4	15	6
Van Leyden	8	6	4

The third and the fifth column (Colour and School) are not shown.



```
R Console
> subset(painters, School=="F", select=c(-3,-5))
      Composition Drawing Expression
Durer          8      10          8
Holbein        9      10         13
Pourbus        4      15          6
Van Leyden     8       6          4
> █
```

# Data Frames

- The command `split` partitions the data set by values of a specific variable. This should preferably be a factor variable.

**Example:** Following command splits `painters` with respect to `School` (A,B,C,... categories)

```
> splitted = split(painters, painters$School)
```

# Data Frames

> splitted

\$A

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
Michelangelo	8	17	4	8	A
Perino del Vaga	15	16	7	6	A
Perugino	4	12	10	4	A
Raphael	17	18	12	18	A

\$B

	Composition	Drawing	Colour	Expression	School
F. Zuccaro	10	13	8	8	B
Fr. Salviata	13	15	8	8	B
Parmigiano	10	15	6	6	B
Primaticcio	15	14	7	10	B
T. Zuccaro	13	14	10	9	B
Volterra	12	15	5	8	B

Contd...

# Data Frames

\$C

	Composition	Drawing	Colour	Expression	School
Barocci	14	15	6	10	C
Cortona	16	14	12	6	C
Josepin	10	10	6	2	C
L. Jordaens	13	12	9	6	C
Testa	11	15	0	6	C
Vanius	15	15	12	13	C

\$D

	Composition	Drawing	Colour	Expression	School
Bassano	6	8	17	0	D
Bellini	4	6	14	0	D
Giorgione	8	9	18	4	D
Murillo	6	8	15	4	D
Palma Giovane	12	9	14	6	D
Palma Vecchio	5	6	16	0	D
Pordenone	8	14	17	5	D
Tintoretto	15	14	16	4	D
Titian	12	15	18	6	D
Veronese	15	10	16	3	D

Contd...

# Data Frames

\$E

	Composition	Drawing	Colour	Expression	School
Albani	14	14	10	6	E
Caravaggio	6	6	16	0	E
Correggio	13	13	15	12	E
Domenichino	15	17	9	17	E
Guercino	18	10	10	4	E
Lanfranco	14	13	10	5	E
The Carraci	15	17	13	13	E

\$F

	Composition	Drawing	Colour	Expression	School
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F

Contd...

# Data Frames

\$G

	Composition	Drawing	Colour	Expression	School
Diepenbeck	11	10	14	6	G
J. Jordaens	10	8	16	6	G
Otho Venius	13	14	10	10	G
Rembrandt	15	6	17	12	G
Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G

\$H

	Composition	Drawing	Colour	Expression	School
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H
Le Sueur	15	15	4	15	H
Poussin	15	17	6	15	H

Remark: If the data set is not attached, we have to use  
`painters$School.`

# Data Frames

```
R Console
> splitted = split(painters, painters$School)
> splitted
$A
      Composition Drawing Colour Expression School
Da Udine          10     8    16      3     A
Da Vinci          15    16     4     14     A
Del Piombo         8    13    16      7     A
Del Sarto          12    16     9      8     A
Fr. Penni          0    15     8      0     A
Guilio Romano     15    16     4     14     A
Michelangelo       8    17     4      8     A
Perino del Vaga   15    16     7      6     A
Perugino           4    12    10      4     A
Raphael            17    18    12     18     A

$B
      Composition Drawing Colour Expression School
F. Zuccaro         10    13     8      8     B
Fr. Salviata       13    15     8      8     B
Parmigiano          10    15     6      6     B
Primaticcio        15    14     7     10     B
T. Zuccaro          13    14    10      9     B
Volterra            12    15     5      8     B

$C
      Composition Drawing Colour Expression School
Barocci             14    15     6     10     C
Cortona             16    14    12      6     C
Josepin              10    10     6      2     C
L. Jordaeans        13    12     9      6     C
Testa                11    15     0      6     C
Vanisus             15    15    12     13     C

$D
      Composition Drawing Colour Expression School
Bassano              6     8    17      0     D
Bellini              4     6    14      0     D
Giorgione            8     9    18      4     D
Murillo               6     8    15      4     D
Palma Giovane        12    9    14      6     D
Palma Vecchio        5     6    16      0     D
Pordenone             8    14    17      5     D
```

```
R Console
$E
      Composition Drawing Colour Expression School
Albani               14     14    10      6     E
Caravaggio            6      6    16      0     E
Corregio              13     13    15     12     E
Domenichino          15     17     9     17     E
Guercino              18     10    10      4     E
Lanfranco             14     13    10      5     E
The Carraci           15     17    13     13     E

$F
      Composition Drawing Colour Expression School
Durer                 8     10    10      8     F
Holbein               9     10    16     13     F
Pourbus                4     15     6      6     F
Van Leyden             8      6     6      4     F

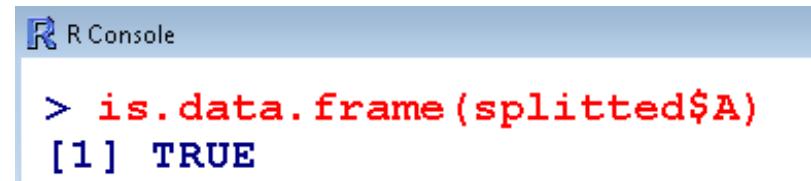
$G
      Composition Drawing Colour Expression School
Diepenbeck            11     10    14      6     G
J. Jordaeans          10      8    16      6     G
Otho Venius            13     14    10     10     G
Rembrandt              15      6    17     12     G
Rubens                 18     13    17     17     G
Teniers                15     12    13      6     G
Van Dyck                15     10    17     13     G

$H
      Composition Drawing Colour Expression School
Bourdon                10      8     8      4     H
Le Brun                 16     16     8     16     H
Le Sueur                15     15     4     15     H
Poussin                 15     17     6     15     H
```

# Data Frames

The objects  `splitted$A`  to  `splitted$H`  are themselves data frames:

```
> is.data.frame(splitted$A)  
[1] TRUE
```



R Console

```
> is.data.frame(splitted$A)  
[1] TRUE
```

# **Foundations of R Software**

**Lecture 44**

## **Data Frames: Combining and Merging**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

## Data Frames: Combining

- There are three main techniques :

`cbind( )` – combining the columns of two data frames side-by-side.

`merge( )` – joining two data frames using a common column.

`rbind( )` – stacking two data frames on top of each other,  
appending one to the other.

## Data Frames: Combining

- The command **cbind** horizontally merges two data frames side by side.

**Example:** Create two data frames as follows:

```
df1=data.frame(state=c("UP", "MP", "AP", "JK"),  
popnsize=c(1000,2000,3000,4000))  
  
df2=data.frame(state=c("UP", "MP", "AP", "JK"),  
samplesize=c(100,200,300,400),  
surveycompleted=c("Yes", "No", "Yes", "No"))
```

## Data Frames: Combining

```
> df1
```

	state	popnsize
1	UP	1000
2	MP	2000
3	AP	3000
4	JK	4000

```
> df2
```

	state	samplesize	surveycompleted
1	UP	100	Yes
2	MP	200	No
3	AP	300	Yes
4	JK	400	No

## Data Frames: Combining

```
> cbind(df1,df2)
```

	state	popnsize	state	samplesize	surveycompleted
1	UP	1000	UP	100	Yes
2	MP	2000	MP	200	No
3	AP	3000	AP	300	Yes
4	JK	4000	JK	400	No

# Data Frames: Combining

```
R Console

> df1
  state popysize
1   UP     1000
2   MP     2000
3   AP     3000
4   JK     4000
> df2
  state samplesize surveycompleted
1   UP       100           Yes
2   MP       200            No
3   AP       300           Yes
4   JK       400            No
> cbind(df1,df2)
  state popysize state samplesize surveycompleted
1   UP     1000     UP       100           Yes
2   MP     2000     MP       200            No
3   AP     3000     AP       300           Yes
4   JK     4000     JK       400            No
~ |
```

## Data Frames: Merging

- The command `merge` horizontally merges two data frames by common columns or row names.

**Example:** Create two data frames as follows:

```
df1=data.frame(state=c("UP", "MP", "AP", "JK"),  
popnsize=c(1000,2000,3000,4000))
```

```
df2=data.frame(state=c("UP", "MP", "AP", "JK"),  
samplesize=c(100,200,300,400),  
surveycompleted=c("Yes", "No", "Yes", "No"))
```

Variable “`state`” is common between the two data frames and we want to merge the two data frames with respect to `state`.

## Data Frames: Merging

- The command `merge` horizontally merges two data frames by common columns or row names.

Usage : `merge(x, y, ...)`

Arguments :

`x, y` : data frames, or objects to be coerced to one.

`by, by.x, by.y` : specifications of the columns used for merging.

`sort` : logical.

`no.dups` : logical indicating that suffixes are appended in more cases to avoid duplicated column names in the result.

## Data Frames: Merging

```
> df1
  state popnsize
1   UP     1000
2   MP     2000
3   AP     3000
4   JK     4000

> df2
  state samplesize surveycompleted
1   UP        100           Yes
2   MP        200            No
3   AP        300           Yes
4   JK        400            No

> merge(df1,df2,by="state")
  state popnsize samplesize surveycompleted
1   AP     3000        300           Yes
2   JK     4000        400            No
3   MP     2000        200            No
4   UP     1000        100           Yes
```

# Data Frames: Merging

```
R R Console

> df1
  state popnsize
1   UP     1000
2   MP     2000
3   AP     3000
4   JK     4000
>
> df2
  state samplesize surveycompleted
1   UP        100           Yes
2   MP        200            No
3   AP        300           Yes
4   JK        400            No
> merge(df1,df2,by="state")
  state popnsize samplesize surveycompleted
1   AP     3000        300           Yes
2   JK     4000        400            No
3   MP     2000        200            No
4   UP     1000        100           Yes
> |
```

## Data Frames: Combining vertically

- The command **rbind** stacks two data frames on top of each other, appending one to the other

**Example:** Create two data frames as follows:

```
df11=data.frame(state=c("UP", "MP", "AP",  
"JK"), popnsize=c(1000,2000,3000,4000))
```

```
df22=data.frame(state=c("Bihar", "Delhi",  
"Punjab"), popnsize =c(100,200,300))
```

## Data Frames: Combining vertically

```
> df11
```

	state	popnsize
1	UP	1000
2	MP	2000
3	AP	3000
4	JK	4000

```
> df22
```

	state	popnsize
1	Bihar	100
2	Delhi	200
3	Punjab	300

R Console

```
> df11
  state popnsize
  1   UP    1000
  2   MP    2000
  3   AP    3000
  4   JK    4000
> df22
  state popnsize
  1 Bihar    100
  2 Delhi    200
  3 Punjab   300
```

## Data Frames: Combining vertically

```
> rbind(df11,df22)
```

	state	popnsize
1	UP	1000
2	MP	2000
3	AP	3000
4	JK	4000
5	Bihar	100
6	Delhi	200
7	Punjab	300

```
R Console
```

```
> df11
  state popnsize
1   UP    1000
2   MP    2000
3   AP    3000
4   JK    4000
> df22
  state popnsize
1 Bihar     100
2 Delhi     200
3 Punjab    300
> rbind(df11,df22)
  state popnsize
1   UP    1000
2   MP    2000
3   AP    3000
4   JK    4000
5 Bihar     100
6 Delhi     200
7 Punjab    300
|
```

# **Foundations of R Software**

## **Lecture 45**

## **Data Handling**

::::

## **Importing and Reading CSV and Tabular Data Files**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Setting up directories:

- We can change the current working directory as follows:

```
> setwd("location of the dataset")
```

## Example:

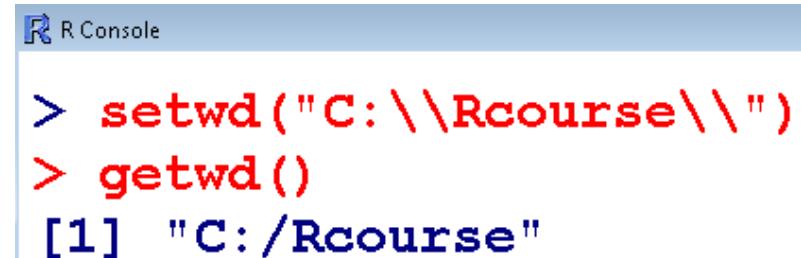
```
> setwd("C:/RCourse/")
```

or

```
> setwd("C:\\Rcourse\\")
```

- The following command returns the current working directory:

```
> getwd()
[1] "C:/RCourse/"
```



A screenshot of an R console window titled "R Console". It shows the following session:

```
> setwd("C:\\Rcourse\\")
> getwd()
[1] "C:/Rcourse"
```

# **Importing data files:**

**Suppose we have some data on our computer and we want to import it in R.**

**Different formats of files can be read in R**

- **comma-separated values (CSV) data file,**
- **table file (TXT),**
- **Spreadsheet (e.g., MS Excel) file,**
- **HTML table files**
- **files from other software like SPSS, Minitab etc.**

# Importing data files:

One can also read or upload the file from Internet site.

We can read the file containing rent index data from website:

<http://home.iitk.ac.in/~shalab/Rcourse/munichdata.asc>

as follows

```
datamunich = read.table(file=
"http://home.iitk.ac.in/~shalab/Rcourse/munichdata.asc", header=TRUE)
```

File name is **munichdata.asc**

# Importing data files:

## Comma-separated values (CSV) files

First set the working directory where the CSV file is located.

```
setwd("<location of your dataset>")
```

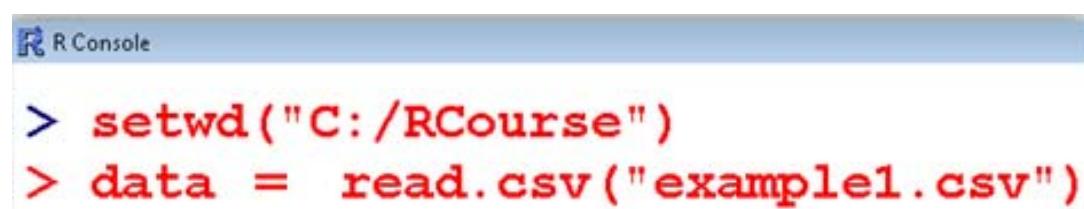
```
> setwd("C:/RCourse/")
```

To read a CSV file

Syntax: `read.csv("filename.csv")`

Example:

```
> data = read.csv("example1.csv")
```



A screenshot of an R console window titled "R Console". The window shows two lines of R code in red font: `> setwd("C:/RCourse")` and `> data = read.csv("example1.csv")`. The console has a light blue header bar and a white body.

# Importing data files:

Comma-separated values (CSV) files

`example1.csv`

The screenshot shows a Microsoft Excel spreadsheet titled "example1 - Excel". The ribbon menu is visible with tabs for File, Home, Insert, Page Layout, Formulas, Data, Review, and View. The "Home" tab is selected. The "Clipboard" group contains a Paste button with a dropdown arrow, a scissor icon, and a clipboard icon. The "Font" group includes Calibri font, size 11, bold (B), italic (I), underline (U), and font color (A). The "Alignment" group includes horizontal alignment icons (left, center, right) and vertical alignment icons (top, middle, bottom). The "Number" group includes a dropdown for "General" with options for currency, percentage, and scientific notation. The formula bar shows cell reference A1, a clear button, a checkmark button, a fx button, and the number 1. The main worksheet area displays a table with 6 rows and 4 columns. The first row (A1-D1) contains the values 1, 10, 100, and an empty cell. The second row (A2-D2) contains 2, 20, 200, and an empty cell. The third row (A3-D3) contains 3, 30, 300, and an empty cell. The fourth row (A4-D4) contains 4, 40, 400, and an empty cell. The fifth row (A5-D5) contains 5, 50, 500, and an empty cell. The sixth row (A6-D6) is entirely empty.

	A	B	C
1	1	10	100
2	2	20	200
3	3	30	300
4	4	40	400
5	5	50	500
6			

# Importing data files:

## Comma-separated values (CSV) files

### Example:

```
> data = read.csv("example1.csv")
```

```
> data
```

```
x1 x10 x100
```

```
1 2 20 200
```

```
2 3 30 300
```

```
3 4 40 400
```

```
4 5 50 500
```

	A	B	C	D
1	1	10	100	
2	2	20	200	
3	3	30	300	
4	4	40	400	
5	5	50	500	
6				

R Console

```
> setwd("C:/RCourse")
```

```
> data = read.csv("example1.csv")
```

```
> data
```

```
x1 x10 x100
```

```
1 2 20 200
```

```
2 3 30 300
```

```
3 4 40 400
```

```
4 5 50 500
```

Notice the difference in the first rows of excel file and output

# Importing data files:

## Comma-separated values (CSV) files

Notice the difference in the first rows of excel file and output

Example:

	A	B	C	D
1	1	10	100	
2	2	20	200	
3	3	30	300	
4	4	40	400	
5	5	50	500	
6				

```
R Console
> setwd("C:/RCourse")
> data = read.csv("example1.csv")
> data
  x1 x10 x100
1  2   20   200
2  3   30   300
3  4   40   400
4  5   50   500
```

# Importing data files:

## Comma-separated values (CSV) files

Data files have many formats and accordingly we have options for loading them.

If the data file does not have headers in the first row, then use

```
data = read.csv("datafile.csv", header=FALSE)
```

# Importing data files:

Comma-separated values (CSV) data

Example:

```
> data = read.csv("example1.csv", header=FALSE)
```

```
> data  
   v1  v2  v3  
1  1  10 100  
2  2  20 200  
3  3  30 300  
4  4  40 400  
5  5  50 500
```

# Importing data files:

Comma-separated values (CSV) data

Example:

	A	B	C	D
1	1	10	100	
2	2	20	200	
3	3	30	300	
4	4	40	400	
5	5	50	500	
6				

```
R Console
> data = read.csv("example1.csv", header=FALSE)
> data
  V1 V2 V3
1  1 10 100
2  2 20 200
3  3 30 300
4  4 40 400
5  5 50 500
```

# Importing data files:

Comma-separated values (CSV) files

The resulting data frame will have columns named V1, V2, ...

We can rename the header names manually:

```
> names(data) =  
c("Column1", "Column2", "Column3")
```

```
> data  
  Column1 Column2 Column3  
1      1      10     100  
2      2      20     200  
3      3      30     300  
4      4      40     400  
5      5      50     500
```

# Importing data files:

## Comma-separated values (CSV) files

```
R Console
> data = read.csv("example1.csv", header=FALSE)
> data
  V1 V2  V3
1  1 10 100
2  2 20 200
3  3 30 300
4  4 40 400
5  5 50 500
> names(data) = c("Column1", "Column2", "Column3")
> data
  Column1 Column2 Column3
1        1       10      100
2        2       20      200
3        3       30      300
4        4       40      400
5        5       50      500
```

# Importing data files:

## Comma-separated values (CSV) files

We can set the delimiter with `sep`.

If it is tab delimited, use `sep="\t"`.

```
data = read.csv("datafile.csv", sep = "\t")
```

If it is space-delimited, use `sep=" "`.

```
data = read.csv("datafile.csv", sep = " ")
```

# Importing data files:

## Reading Tabular Data Files

Tabular data files are text files with a simple format:

- Each line contains one record.
- Within each record, fields (items) are separated by a one-character delimiter, such as a space, tab, colon, or comma.
- Each record contains the same number of fields.

We want to read a text file that contains a table of data.

**read.table** function is used and it returns a data frame.

```
read.table("FileName")
```

# Importing data files:

## Reading Tabular Data Files

Data:

```
1 10 100  
2 20 200  
3 30 300  
4 40 400  
5 50 500
```

example3 - Notepad		
File	Edit	Format
View	Help	
1	10	100
2	20	200
3	30	300
4	40	400
5	50	500

Saved in example3.txt

```
> data = read.table("example3.txt", sep = " ")  
> data  
   V1  V2  V3  
1  1 10 100  
2  2 20 200  
3  3 30 300  
4  4 40 400  
5  5 50 500
```

# Importing data files:

## Reading Tabular Data Files

R R Console

```
> data = read.table("example3.txt", sep=" ")
> data
  V1  V2  V3
1  1  10 100
2  2  20 200
3  3  30 300
4  4  40 400
5  5  50 500
```

# **Foundations of R Software**

## **Lecture 46**

## **Data Handling**

::::

## **Importing and Reading EXCEL and other Data Files**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Importing data files from other software:

```
> setwd("C:/RCourse/")
```

## Spreadsheet (Excel) file data

The `readxl` package has the function `read_excel()` for reading Excel files.

This will read the first sheet of an Excel spreadsheet.

To read Excel files, we first need to install the package

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

```
library(readxl)
```

# Importing data files from other software:

Spreadsheet (Excel) file data

```
read_excel("datafile.xlsx")
```

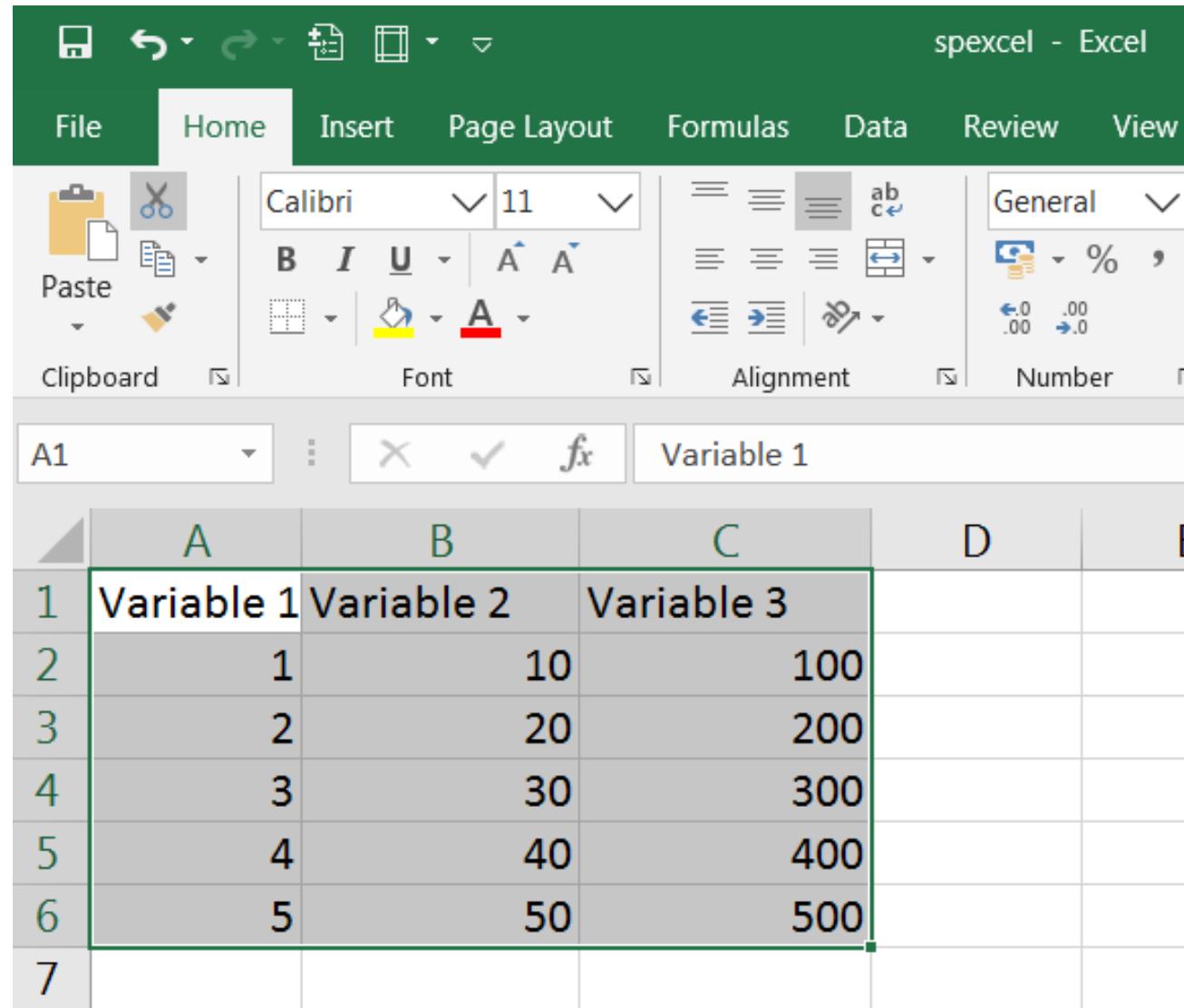
```
read_excel("datafile.xls")
```

# Specify sheet either by position or by name

```
read_excel(datasets, sheet_number)
```

```
read_excel(datasets, "sheet_name")
```

# Importing data files:



A screenshot of the Microsoft Excel interface. The ribbon at the top shows the tabs: File, Home, Insert, Page Layout, Formulas, Data, Review, and View. The Home tab is selected. The ribbon also includes sections for Clipboard, Font, Alignment, and Number. The active cell is A1, and the formula bar shows "Variable 1". The main area displays a data table with columns labeled A, B, C, D, and E. The first row contains the labels "Variable 1", "Variable 2", and "Variable 3". Rows 2 through 6 contain numerical values: (1, 10, 100), (2, 20, 200), (3, 30, 300), (4, 40, 400), and (5, 50, 500). Row 7 is empty.

	A	B	C	D	E
1	Variable 1	Variable 2	Variable 3		
2	1	10	100		
3	2	20	200		
4	3	30	300		
5	4	40	400		
6	5	50	500		
7					

# Importing data files:

To extract variable, write

`object_name$Variable_name`

**Example:**

```
> dataspexcel = read_excel("spexcel.xlsx",  
sheet=1)
```

```
> dataspexcel  
# A tibble: 5 x 3  
  `Variable 1` `Variable 2` `Variable 3`  
    <dbl>        <dbl>        <dbl>  
1      1          10         100  
2      2          20         200  
3      3          30         300  
4      4          40         400  
5      5          50         500
```

# Importing data files:

Excel data

Example:

```
> dataspexcel$`Variable 1`  
[1] 1 2 3 4 5
```

	A	B	C	D
1	Variable 1	Variable 2	Variable 3	
2	1	10	100	
3	2	20	200	
4	3	30	300	
5	4	40	400	
6	5	50	500	
7				

Sheet1 Sheet2 Sheet3

```
> dataspexcel$`Variable 2`  
[1] 10 20 30 40 50
```

```
> mean(dataspexcel$`Variable 1`)  
[1] 3
```

# Importing data files:

## Excel data Example:

```
R Console

> dataspexcel = read_excel("spexcel.xlsx", sheet=1)
> dataspexcel
# A tibble: 5 x 3
  `Variable 1` `Variable 2` `Variable 3`
  <dbl>        <dbl>        <dbl>
1      1          10         100
2      2          20         200
3      3          30         300
4      4          40         400
5      5          50         500
>
> dataspexcel$`Variable 1`
[1] 1 2 3 4 5
> dataspexcel$`Variable 2`
[1] 10 20 30 40 50
>
> mean(dataspexcel$`Variable 1`)
[1] 3
>
```

# Importing data files:

## Excel data

### Example:

```
> dataspexcel2 = read_excel("spexcel.xlsx",  
sheet=2)
```

```
> dataspexcel2  
# A tibble: 5 x 3  
  `Variable 4` `Variable 5` `Variable 6`  
    <dbl>        <dbl>        <dbl>  
1       6         110         110  
2       7         120         210  
3       8         130         310  
4       9         140         410  
5      10         150         510
```

# Importing data files:

Excel data

Example:

```
> dataspexcel2$`Variable 4`  
[1] 6 7 8 9 10
```

```
> dataspexcel2$`Variable 5`  
[1] 110 120 130 140 150
```

```
> dataspexcel2$`Variable 6`  
[1] 110 210 310 410 510
```

```
> mean(dataspexcel2$`Variable 6`)  
[1] 310
```

	A	B	C	D
1	Variable 4	Variable 5	Variable 6	
2	6	110	110	
3	7	120	210	
4	8	130	310	
5	9	140	410	
6	10	150	510	
7				

# Importing data files:

## Excel data Example:

```
R Console

> dataspexcel2 = read_excel("spexcel.xlsx", sheet=2)
> dataspexcel2
# A tibble: 5 x 3
`Variable 4` `Variable 5` `Variable 6`
  <dbl>       <dbl>       <dbl>
1       6        110        110
2       7        120        210
3       8        130        310
4       9        140        410
5      10        150        510
>
> dataspexcel2$`Variable 4`
[1] 6 7 8 9 10
>
> dataspexcel2$`Variable 5`
[1] 110 120 130 140 150
>
> dataspexcel2$`Variable 6`
[1] 110 210 310 410 510
>
> mean(dataspexcel2$`Variable 6`)
[1] 310
```

# Importing data files from other software:

Spreadsheet (Excel) file data

# Limit the number of data rows read

```
read_excel(datasets, n_max = 3)
```

# Read from an Excel range using A1 or R1C1 notation

```
read_excel(datasets, range = "C1:E7")
```

```
read_excel(datasets, range = "R1C2:R2C5")
```

R1C1 notation : Row-Column notation

R2C3 refers to the cell at the second row and third column

# Importing data files from other software:

## Spreadsheet (Excel) file data

```
# Limit the number of data rows read
```

```
read_excel(datasets, n_max = 3)
```

```
dataspexcel4 = read_excel("spexcel.xlsx",
n_max=3)
```

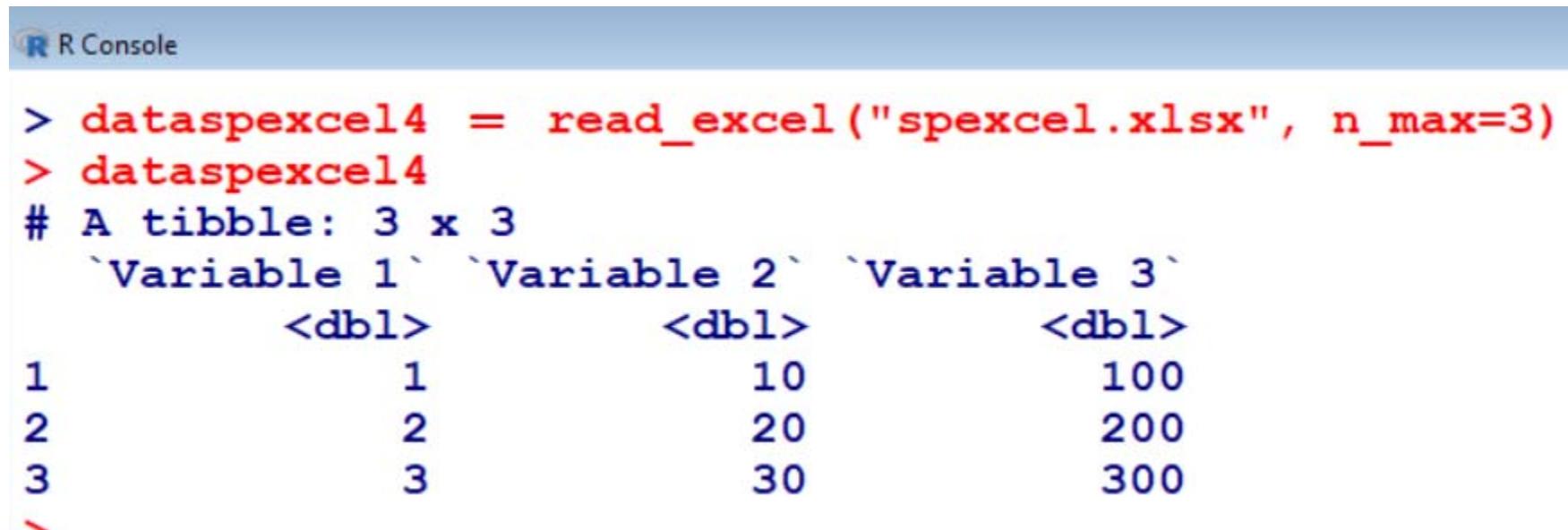
```
dataspexcel4
```

```
# A tibble: 3 x 3
```

	`Variable 1`	`Variable 2`	`Variable 3`
	<dbl>	<dbl>	<dbl>
1	1	10	100
2	2	20	200
3	3	30	300

# Importing data files from other software:

```
dataspexcel4 = read_excel("spexcel.xlsx",
n_max=3)
```



R Console

```
> dataspexcel4 = read_excel("spexcel.xlsx", n_max=3)
> dataspexcel4
# A tibble: 3 x 3
`Variable 1` `Variable 2` `Variable 3`
  <dbl>        <dbl>        <dbl>
1      1          10         100
2      2          20         200
3      3          30         300
```

# Importing data files from other software:

## Spreadsheet (Excel) file data

# Read from an Excel range using A1 or R1C1 notation

```
R Console

> dataspexcel5 = read_excel("spexcel.xlsx", range="A2:B3", sheet=1)
> dataspexcel5
# A tibble: 1 x 2
  `1`   `10` 
  <dbl> <dbl>
1     2     20
>
> dataspexcel6 = read_excel("spexcel.xlsx", range="A2:B3", sheet=2)
> dataspexcel6
# A tibble: 1 x 2
  `6`   `110` 
  <dbl> <dbl>
1     7     120
>
```

# Importing data files from other software:

## SPSS data file

For reading SPSS data files, use **foreign** package and **function**

**read.spss( )**

To read SPSS files, we first need to install the package

```
install.packages(" foreign ")  
library(foreign)  
data = read.spss("datafile.sav")
```

# Importing data files:

HTML data file

For reading HTML data files, use **XML** package and function

**readHTMLTable**

To read HTML data files, we first need to install the package

```
install.packages( "XML" )
```

```
library( XML )
```

```
data = readHTMLTable( "filename" )
```

# Importing data files:

## Other data files

The `foreign` package also includes functions to load from other formats, including:

- `read.octave("Path to file")`: Octave and MATLAB
- `read.systat("Path to file")`: SYSTAT
- `read.xport("Path to file")`: SAS XPORT
- `read.dta("Path to file")`: Stata

# **Foundations of R Software**

## **Lecture 47**

## **Data Handling**

::::

## **Saving and Writing Data Files**

Shalabh

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Saving and writing data files:

The `write` function can write the data (usually a matrix) `x` are written to file `file`.

If `x` is a two-dimensional matrix you need to transpose it to get the columns in file the same as those in the internal representation.

# Saving and writing data files:

```
write(x, file = "data", ncolumns ,  
      append = FALSE, sep = " ")
```

## Arguments

**x** the data to be written out, usually an atomic vector.

**file** a connection, or a character string naming the file to write to. If "", print to the standard output connection.

# Saving and writing data files:

## Arguments

### `ncolumns`

the number of columns to write the data in.

### `append`

if `TRUE` the data `x` are appended to the connection.

### `sep`

a string used to separate columns. Using `sep = "\t"` gives tab delimited output; default is `" "`.

# Saving and writing data files:

```
> x=c(1:100)
```

```
> x
```

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18  
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36  
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54  
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72  
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90  
[91] 91 92 93 94 95 96 97 98 99 100
```

```
> write(x, file="shalabh")
```

# Saving and writing data files:

The image shows a side-by-side comparison between an R console window and a Microsoft Notepad window.

**R Console (Left):**

```
> x=c(1:100)
> x
 [1]  1   2   3   4   5   6   7
 [8]  8   9   10  11  12  13  14
[15] 15  16  17  18  19  20  21
[22] 22  23  24  25  26  27  28
[29] 29  30  31  32  33  34  35
[36] 36  37  38  39  40  41  42
[43] 43  44  45  46  47  48  49
[50] 50  51  52  53  54  55  56
[57] 57  58  59  60  61  62  63
[64] 64  65  66  67  68  69  70
[71] 71  72  73  74  75  76  77
[78] 78  79  80  81  82  83  84
[85] 85  86  87  88  89  90  91
[92] 92  93  94  95  96  97  98
[99] 99  100
> write(x, file="shalabh")
```

**Notepad (Right):**

shalabh - Notepad

File	Edit	Format	View	Help
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40
41	42	43	44	45
46	47	48	49	50
51	52	53	54	55
56	57	58	59	60
61	62	63	64	65
66	67	68	69	70
71	72	73	74	75
76	77	78	79	80
81	82	83	84	85
86	87	88	89	90
91	92	93	94	95
96	97	98	99	100

## Saving and writing tabular and CSV data files:

The `write.csv` function can write tabular data to an ASCII file in CSV format. Each row of data creates one line in the file, with data items separated by commas (,):

```
write.csv(x, file = "", append = FALSE)

write.csv(x, file = "", append = FALSE, quote =
TRUE, sep = " ", eol = "\n", na = "NA", dec =
".", row.names = TRUE, col.names = TRUE,
qmethod = c("escape", "double"), fileEncoding =
"")
```

## Saving and writing tabular and CSV data files:

The `write.table` prints its required argument `x`.

```
write.table(x, file = "", append = FALSE)

write.table(x, file = "", append = FALSE, quote
= TRUE, sep = " ", eol = "\n", na = "NA", dec =
".", row.names = TRUE, col.names = TRUE,
qmethod = c("escape", "double"), fileEncoding =
"")
```

# Saving and writing tabular and CSV data files:

## Discussion

**x** the object to be written, preferably a matrix or data frame. If not, it is attempted to coerce **x** to a data frame.

**file** either a character string naming a file or a connection open for writing. "" indicates output to the console.

**append** logical. Only relevant if file is a character string. If **TRUE**, the output is appended to the file. If **FALSE**, any existing file of the name is destroyed.

## Saving and writing tabular and CSV data files:

**quote** a logical value (**TRUE** or **FALSE**) or a numeric vector. If **TRUE**, any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of columns to quote. If **FALSE**, nothing is quoted.

**sep** the field separator string. Values within each row of **x** are separated by this string.

**eol** the character(s) to print at the end of each line (row).

**na** the string to use for missing values in the data.

# **Foundations of R Software**

**Lecture 48**

## **Introduction to Statistical Functions**

**:::**

## **Introduction, Frequencies and Partition Values**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# **Descriptive statistics:**

**First hand tools which gives first hand information.**

- **Central tendency of data**
- **Variation in data**
- **Structure and shape of data tendency**
- **Relationship study**

**Graphical as well as analytical tools are used.**

## Absolute and relative frequencies:

Suppose there are 10 persons coded into two categories as male (M) and female (F).

M, F, M, F, M, M, M, F, M, M.

Use  $a_1$  and  $a_2$  to refer to male and female categories.

There are 7 male and 3 female persons,  
denoted as  $n_1 = 7$  and  $n_2 = 3$

The number of observations in a particular category is called the absolute frequency.

# Absolute and relative frequencies:

The relative frequencies of  $a_1$  and  $a_2$  are

$$f_1 = \frac{n_1}{n_1 + n_2} = \frac{7}{10} = 0.7 = 70\%$$

$$f_2 = \frac{n_2}{n_1 + n_2} = \frac{3}{10} = 0.3 = 30\%$$

This gives us information about the proportions of male and female persons.

# Absolute and relative frequencies:

`table(variable)` creates the absolute frequency of the `variable` of the data file.

Enter data as `x`

`table(x) # absolute frequencies`

`table(x)/length(x) # relative frequencies`

# Absolute and relative frequencies:

Example: Code the 10 persons by using, say 1 for male (M) and 2 for female (F).

M, F, M, F, M, M, M, F, M, M  
1, 2, 1, 2, 1, 1, 1, 2, 1, 1

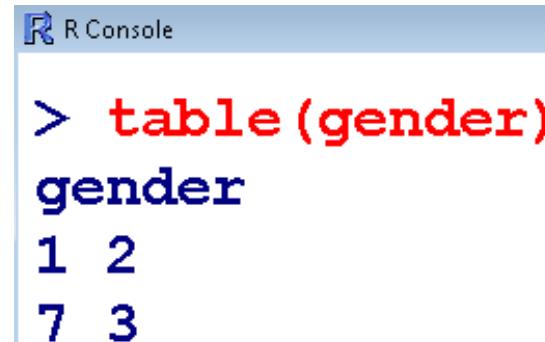
```
> gender <- c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)  
  
> gender  
[1] 1 2 1 2 1 1 1 2 1 1
```

```
R R Console  
  
> gender <- c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)  
> gender  
[1] 1 2 1 2 1 1 1 2 1 1
```

## Absolute and relative frequencies:

```
> table(gender) # Absolute frequencies  
gender
```

```
1 2  
7 3
```

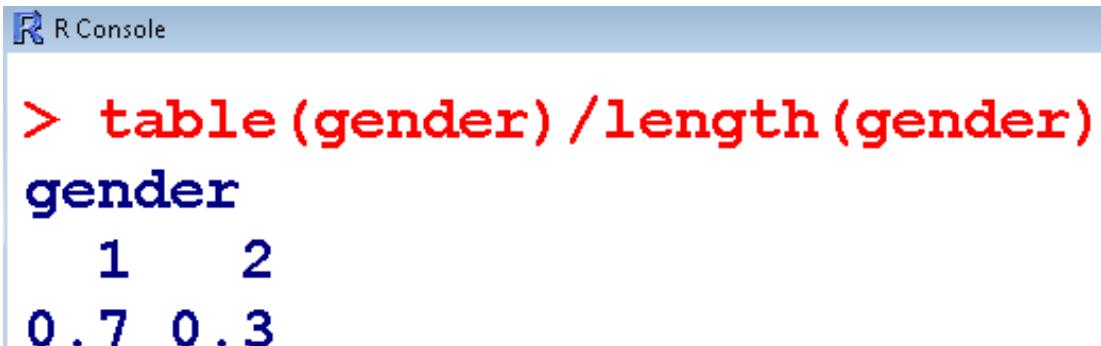


R Console

```
> table(gender)  
gender  
1 2  
7 3
```

```
> table(gender)/length(gender) #Relative freq.
```

```
gender  
1 2  
0.7 0.3
```



R Console

```
> table(gender)/length(gender)  
gender  
1 2  
0.7 0.3
```

# Absolute and relative frequencies:

**Example:**

Consider the following data on pizza home delivery.

- There are three branches (East - coded as 1, West - coded as 2, Central - coded as 3) of the restaurant.

The 100 values from code Directions are as follows:

```
direction =  
c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2  
,1,3,3,3,2,2,2,2,1,2,2,1,1,1,3,2,2,1,2,3,2,2,1,  
2,3,3,2,1,2,2,3,1,1,2,1,2,3,2,3,2,2,3,1,2,3,3,3  
,2,1,1,1,2,1,1,2,1,2,3,3,1,2,3,3,2,1,2,3,2,1,3,  
2,2,2,2,3,2,2)
```

# Absolute and relative frequencies:

Example:

```
> table(direction) # Absolute frequencies  
direction  
 1  2  3  
28 43 29  
  
> table(direction)/length(direction) # Relative  
                                         frequencies  
  
direction  
 1      2      3  
0.28  0.43  0.29
```

# Absolute and relative frequencies:

Example:

```
R Console
> direction = c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2,1,3,3,2,2,2,2
> direction
[1] 1 1 2 1 2 3 2 2 3 3 3 1 2 3 2 2 3 1 1 3 3 1 2 1 3 3 2 2 2 2
[32] 1 2 2 1 1 1 3 2 2 1 2 3 2 2 1 2 3 3 2 1 2 2 3 1 1 2 1 2 3 2 3
[63] 2 2 3 1 2 3 3 3 2 1 1 1 2 1 1 2 1 2 3 3 1 2 3 3 2 1 2 3 2 1 3
[94] 2 2 2 2 3 2 2
> table(direction)
direction
 1  2  3
28 43 29
> table(direction)/length(direction)
direction
 1    2    3
0.28 0.43 0.29
>
```

## **Partition values:**

Such values divides the total frequency given data into required number of partitions.

**Quartile:** Divides the data into 4 equal parts.

**Decile:** Divides the data into 10 equal parts.

**Percentile:** Divides the data into 100 equal parts.

## Partition values:

`quantile` function computes quantiles corresponding to the given probabilities.

The smallest observation corresponds to a probability of 0 and the largest to a probability of 1.

`quantile(x, ...)`

`quantile(x, probs = seq(0, 1, 0.25), ...)`

### Arguments

- x** numeric vector whose sample quantiles are wanted,
- probs** numeric vector of probabilities with values in [0, 1].

## Partition values:

**Example:** Marks of 15 students are

```
> marks = c(68, 82, 63, 86, 34, 96, 41, 89, 29,  
51, 75, 77, 56, 59, 42)  
  
> quantile(marks)  
0%   25%   50%   75% 100%  
29.0 46.5 63.0 79.5 96.0
```

**Default values**

```
> quantile(marks, probs=c(0,0.25,0.5,0.75,1))  
0%   25%   50%   75% 100%  
29.0 46.5 63.0 79.5 96.0
```

# Partition values:

**Example:** Marks of 15 students are

Default values

```
> quantile(marks, probs=c(0,0.25,0.5,0.75,1))  
 0% 25% 50% 75% 100%  
29.0 46.5 63.0 79.5 96.0
```

Defining probabilities

```
> quantile(marks, probs=c(0,0.20,0.4,0.6,0.8,1))  
 0% 20% 40% 60% 80% 100%  
29.0 41.8 57.8 70.8 82.8 96.0
```

# Partition values:

Example: Marks of 15 students are

```
R Console
> marks = c(68, 82, 63, 86, 34, 96, 41, 89, 29, 51, 75, 77, 56, 59)
> marks
[1] 68 82 63 86 34 96 41 89 29 51 75 77 56 59 42
>
> quantile(marks)
 0% 25% 50% 75% 100%
29.0 46.5 63.0 79.5 96.0
>
> quantile(marks, probs=c(0,0.25,0.5,0.75,1))
 0% 25% 50% 75% 100%
29.0 46.5 63.0 79.5 96.0
>
> quantile(marks, probs=c(0,0.20,0.4,0.6,0.8,1))
 0% 20% 40% 60% 80% 100%
29.0 41.8 57.8 70.8 82.8 96.0
> |
```

# **Foundations of R Software**

## **Lecture 49**

## **Graphics**

⋮

## **Scatter Plot and Bar Plots**

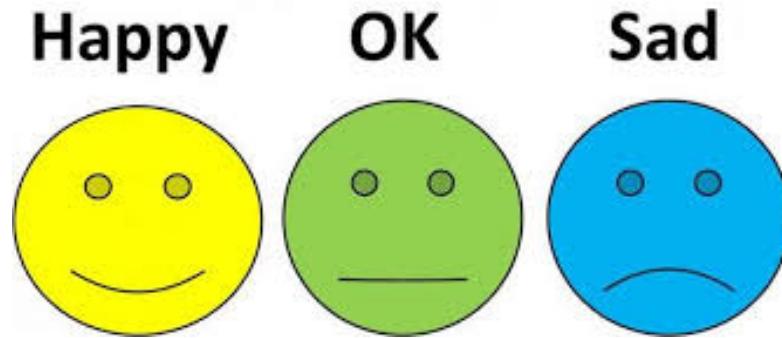
**Shalabh**

**Department of Mathematics and Statistics**  
**Indian Institute of Technology Kanpur**

# **Graphics:**

**Graphics summarize the information contained in a data.**

**For example, the mood of a person may be conveyed very easily by the smilies:**



**They have an advantage that they convey the information hidden inside the data more compactly**

**Appropriate number and choice of plots in analysis provides better inferences.**

# **Graphics:**

**Graphical tools- various type of plots**

- 2D & 3D plots,
- scatter diagram
- Pie diagram
- Histogram
- Bar plot
- Stem and leaf plot
- Box plot ...

**Appropriate number and choice of plots in analysis provides better inferences.**

# **Graphics:**

**In R, Such graphics can be easily created and saved in various formats.**

- **Bar plot**
- **Pie chart**
- **Box plot**
- **Grouped box plot**
- **Scatter plot**
- **Histogram**
- **Various 3 dimensional plots**
- ...

# Scatter Plot

Plot command for one variable:

**x:** Data vector

**plot(x)**

# Scatter Plot

## Example

Height of 50 persons are recorded in centimeters as follows:

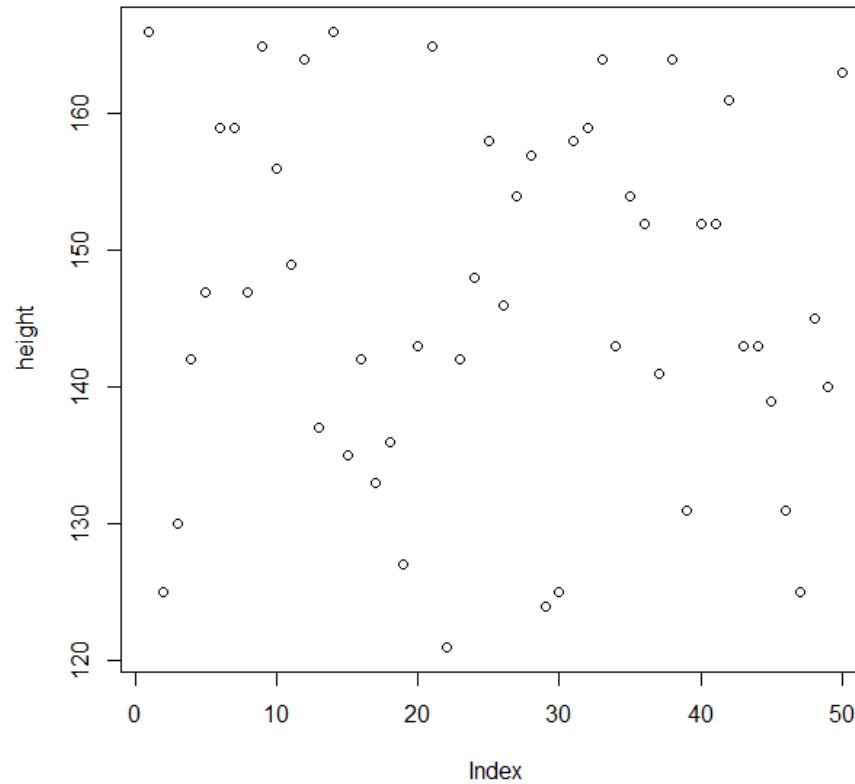
166,125,130,142,147,159,159,147,165,156,149,164,137,166,135,142,  
133,136,127,143,165,121,142,148,158,146,154,157,124,125,158,159,  
164,143,154,152,141,164,131,152,152,161,143,143,139,131,125,145,  
140,163

```
> height = c(166,125,130,142,147,159,159,147,  
165,156,149,164,137,166,135,142,133,136,127,143,  
165,121,142,148,158,146,154,157,124,125,158,159,  
164,143,154,152,141,164,131,152,152,161,143,143,  
139,131,125,145,140,163)
```

# Scatter Plot

## Example

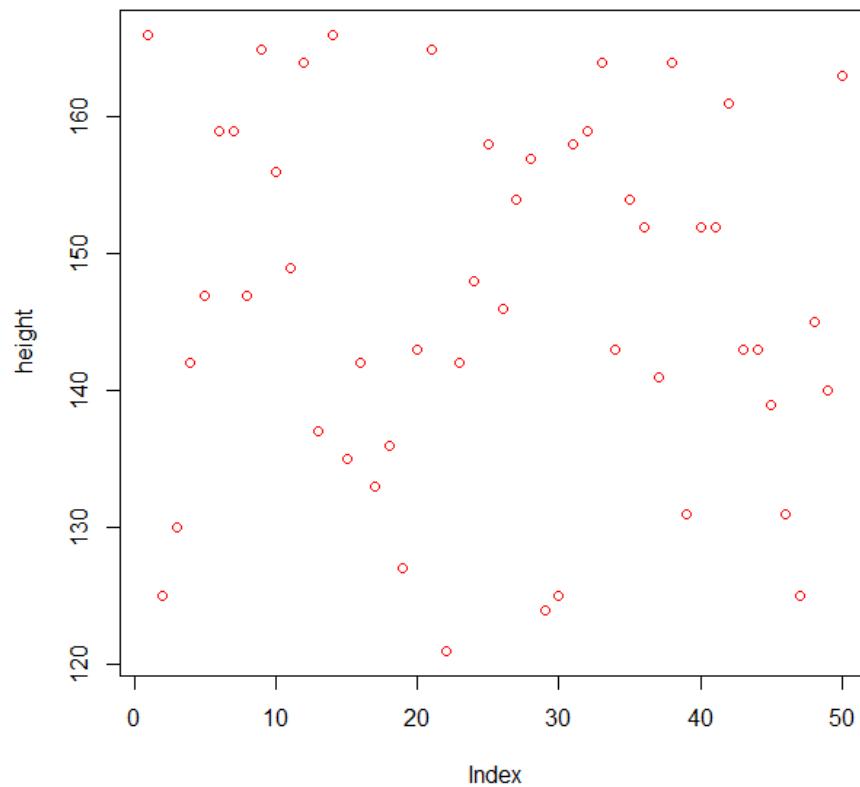
```
plot(height)
```



# Scatter Plot

## Example

```
plot(height, col = "red")
```



# **Bar plots:**

**Visualize the relative or absolute frequencies of observed values of a variable.**

**Used for categorical variables only.**

**It consists of one bar for each category.**

**The height of each bar is determined by either the absolute frequency or the relative frequency of the respective category and is shown on the *y-axis*.**

# Bar plots:

Visualize the relative or absolute frequencies of observed values of a variable.

```
barplot(x, width = 1, space = NULL,...)
```

Bar plot with absolute frequencies

```
barplot(table(x)) # Absolute frequencies
```

Bar plot with relative frequencies

```
barplot(table(x)/length(x))
```

## Bar plots:

```
> help("barplot")
```

```
barplot(height, width = 1, space = NULL,  
names.arg = NULL, legend.text = NULL, beside  
= FALSE, horiz = FALSE, density = NULL, angle  
= 45, col = NULL, border = par("fg"), main =  
NULL, sub = NULL, xlab = NULL, ylab = NULL,  
xlim = NULL, ylim = NULL, xpd = TRUE, log =  
"", axes = TRUE, axisnames = TRUE, cex.axis =  
par("cex.axis"), cex.names = par("cex.axis"),  
inside = TRUE, plot = TRUE, axis.lty = 0,  
offset = 0, add = FALSE, args.legend = NULL,  
...)
```

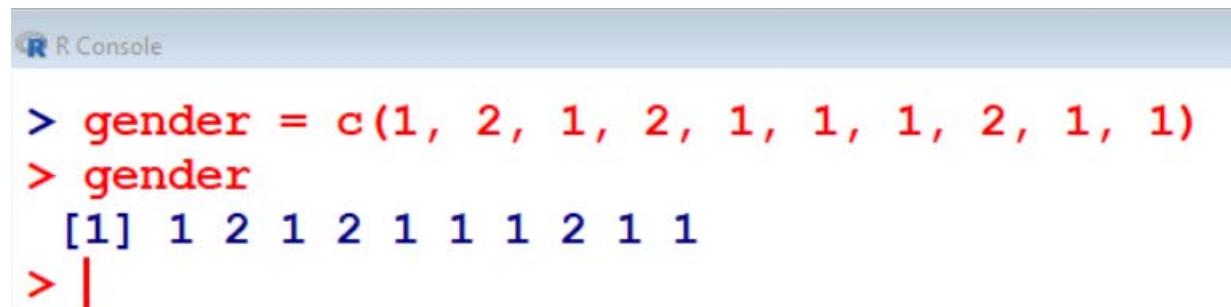
# Bar plots:

Example:

Code the 10 persons by using, say 1 for male (M) and 2 for female (F).

M, F, M, F, M, M, M, F, M, M  
1, 2, 1, 2, 1, 1, 1, 2, 1, 1

```
> gender = c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)  
  
> gender  
[1] 1 2 1 2 1 1 1 2 1 1
```



R Console

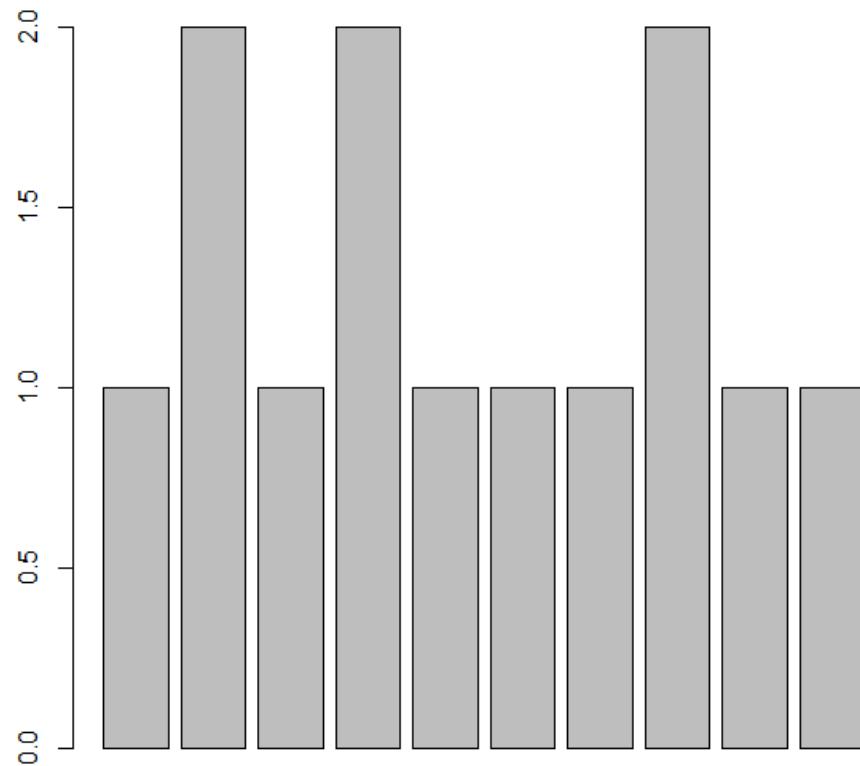
```
> gender = c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)  
> gender  
[1] 1 2 1 2 1 1 1 2 1 1  
> |
```

# Bar plots:

Example:

```
> barplot(gender)
```

Do you want this?



# Bar plots:

Example:

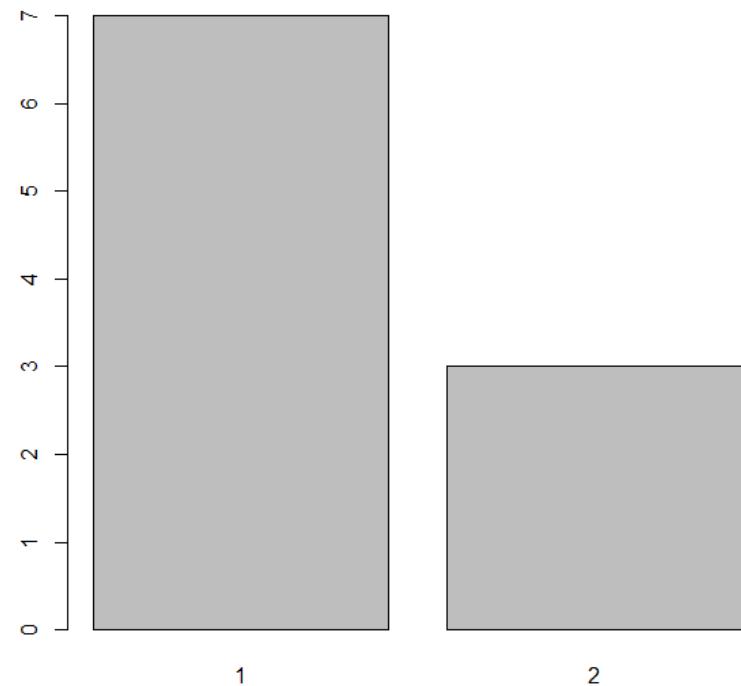
```
> table(gender)
```

```
gender
```

```
1 2
```

```
7 3
```

```
> barplot(table(gender))
```



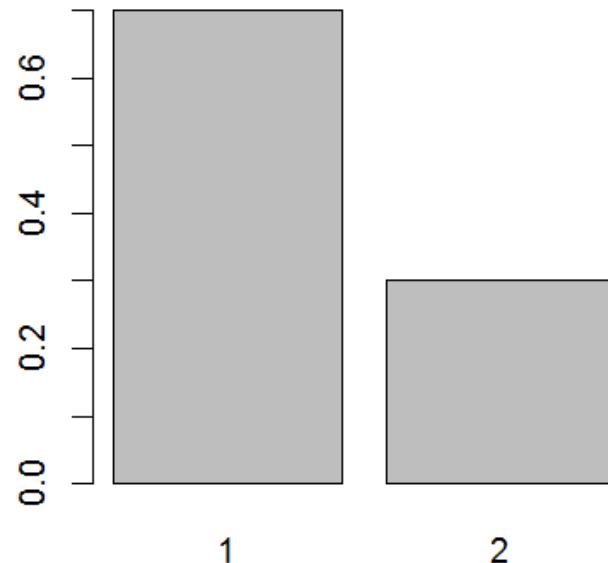
## Bar plots:

Example:

```
> table(gender)/length(gender)
```

```
gender
```

1	2
0.7	0.3



```
> barplot(table(gender)/length(gender))
```

## Bar plots:

### Example:

Consider the following data on pizza home delivery. There are three branches (East - coded as 1, West - coded as 2, Central - coded as 3) of the restaurant.

The 100 values from code Directions are as follows:

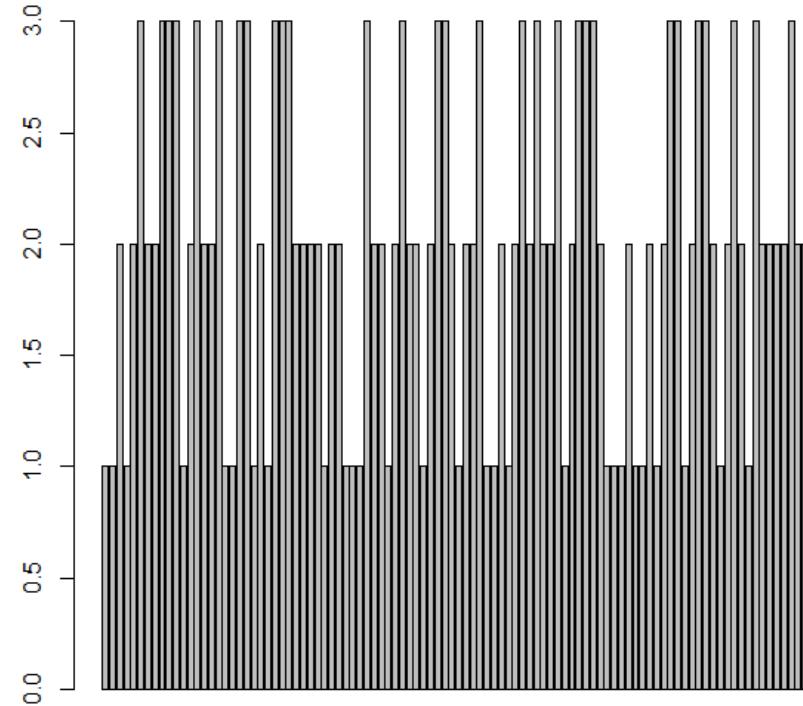
```
direction =  
c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2  
,1,3,3,3,2,2,2,2,1,2,2,1,1,1,3,2,2,1,2,3,2,2,1,  
2,3,3,2,1,2,2,3,1,1,2,1,2,3,2,3,2,2,3,1,2,3,3,3  
,2,1,1,1,2,1,1,2,1,2,3,3,1,2,3,3,2,1,2,3,2,1,3,  
2,2,2,2,3,2,2)
```

# Bar plots:

Example:

```
barplot(direction)
```

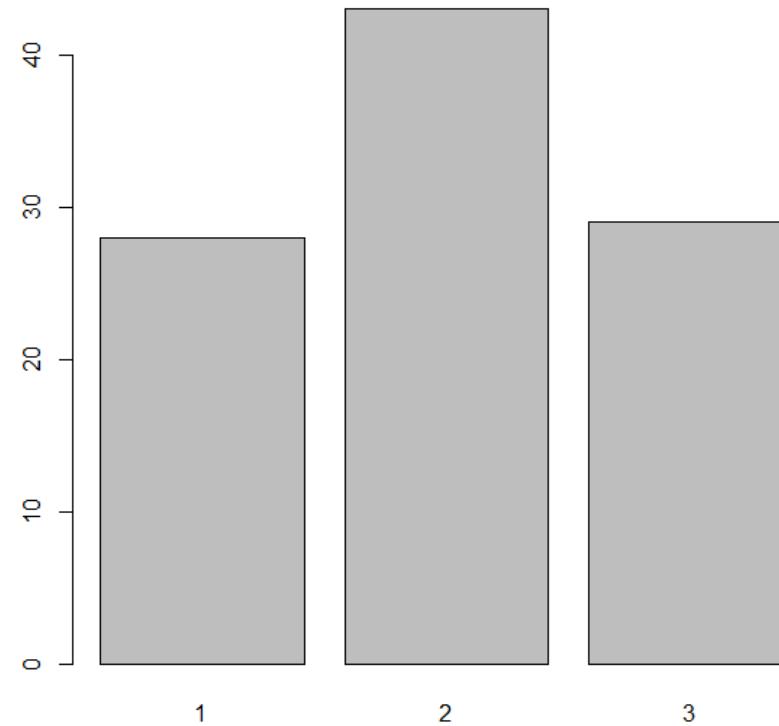
Do you want this?



# Bar plots:

Example

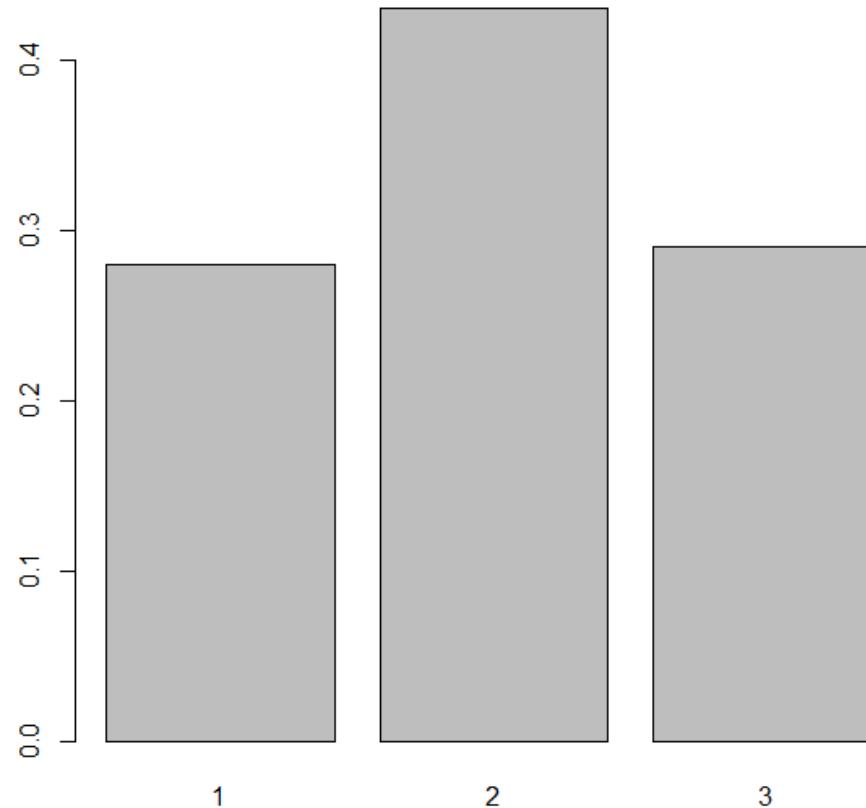
```
barplot(table(direction))
```



# Bar plots:

## Example

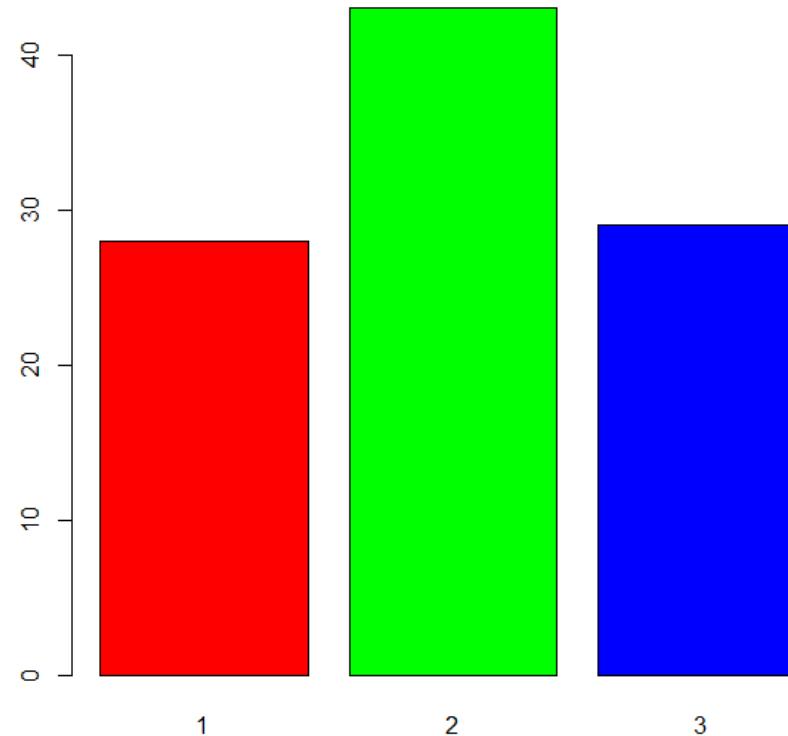
```
barplot(table(direction)/length(direction))
```



# Bar plots:

Example: Adding colours

```
barplot(table(direction), col=c("red", "green",  
"blue") )
```



# Bar plots:

Example: Adding title

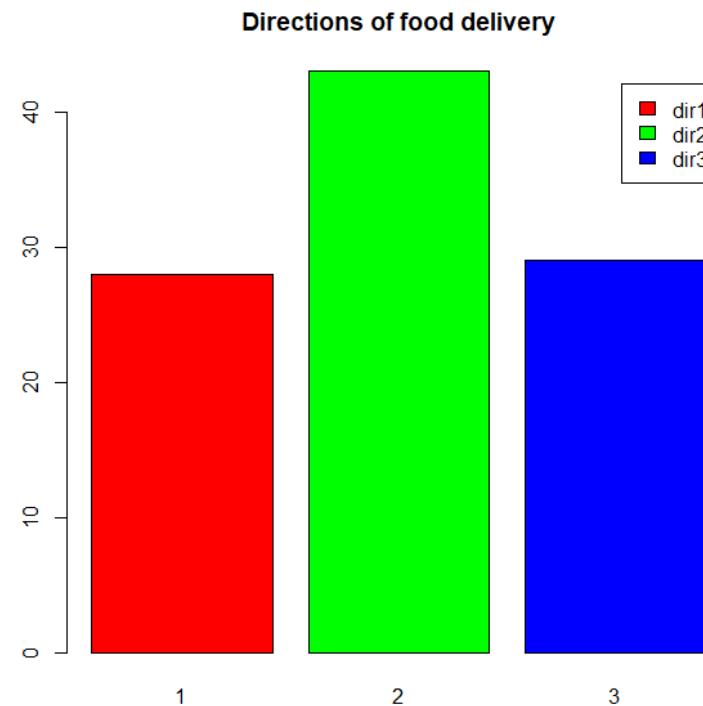
```
barplot(table(direction), col=c("red", "green",  
"blue"), main="Directions of food delivery" )
```



# Bar plots:

Example: Adding legends

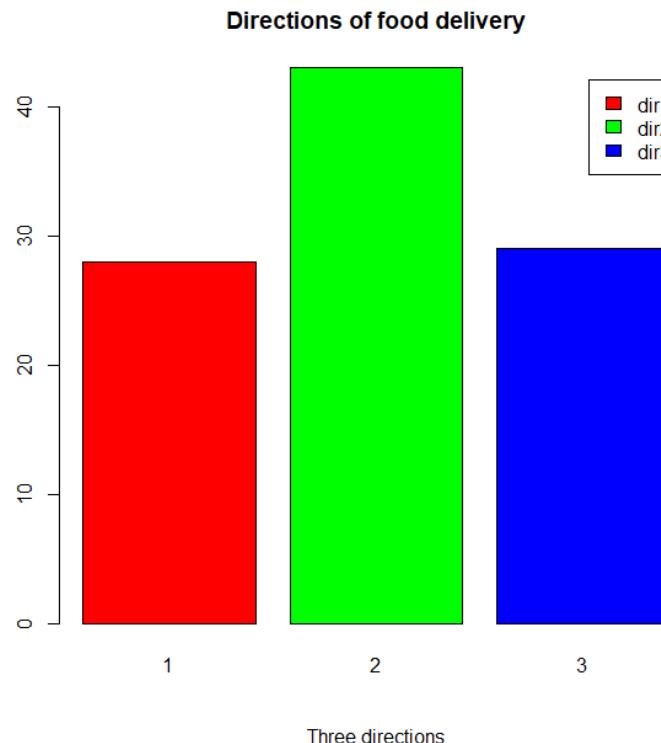
```
barplot(table(direction), col=c("red", "green",  
"blue"), main="Directions of food delivery",  
legend.text=c("dir1", "dir2", "dir3") )
```



# Bar plots:

## Example: Adding Subtitle

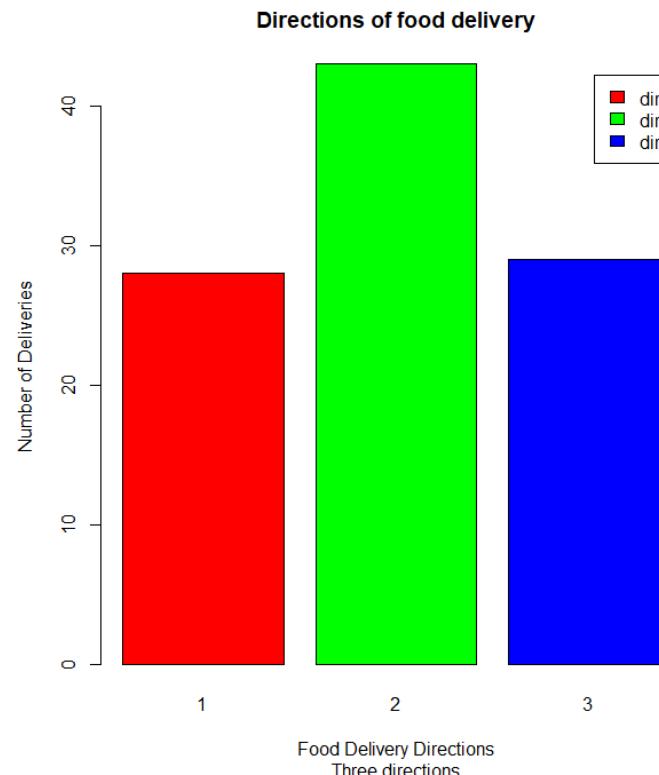
```
barplot(table(direction), col=c("red", "green",  
"blue"), main="Directions of food delivery",  
legend.text=c("dir1", "dir2", "dir3"),  
sub="Three directions" )
```



# Bar plots:

## Example: Adding Subtitle

```
barplot(table(direction), col=c("red", "green",  
"blue"), main="Directions of food delivery",  
legend.text=c("dir1", "dir2", "dir3"),  
sub="Three directions", xlab="Food Delivery  
Directions", ylab="Number of Deliveries" )
```



# **Foundations of R Software**

## **Lecture 50**

## **Graphics**

⋮

## **Sub-Divided Bar Plots and Pie Diagram**

Shalabh

**Department of Mathematics and Statistics**  
**Indian Institute of Technology Kanpur**

# **Subdivided or component bar diagram**

**Subdivided or component bar diagram divides the total magnitude of variables into various parts.**

# Subdivided or component bar diagram

## Example

The data on the number of customers visiting 3 shops during 10-11 AM on 4 consecutive days is as follows:

No. of customers	Shop 1	Shop 2	Shop 3
Day 1	2	20	30
Day 2	26	53	40
Day 3	42	15	25
Day 4	30	75	100

```
cust = matrix(nrow=4, ncol=3, data =c(2,20, 30,26,53,40,42,15,25,30,75,100), byrow = T)
```

```
> cust
      [,1] [,2] [,3]
[1,]    2   20   30
[2,]   26   53   40
[3,]   42   15   25
[4,]   30   75  100
```

# **Subdivided or component bar diagram**

**Usage**

```
barplot(variable in matrix format)
```

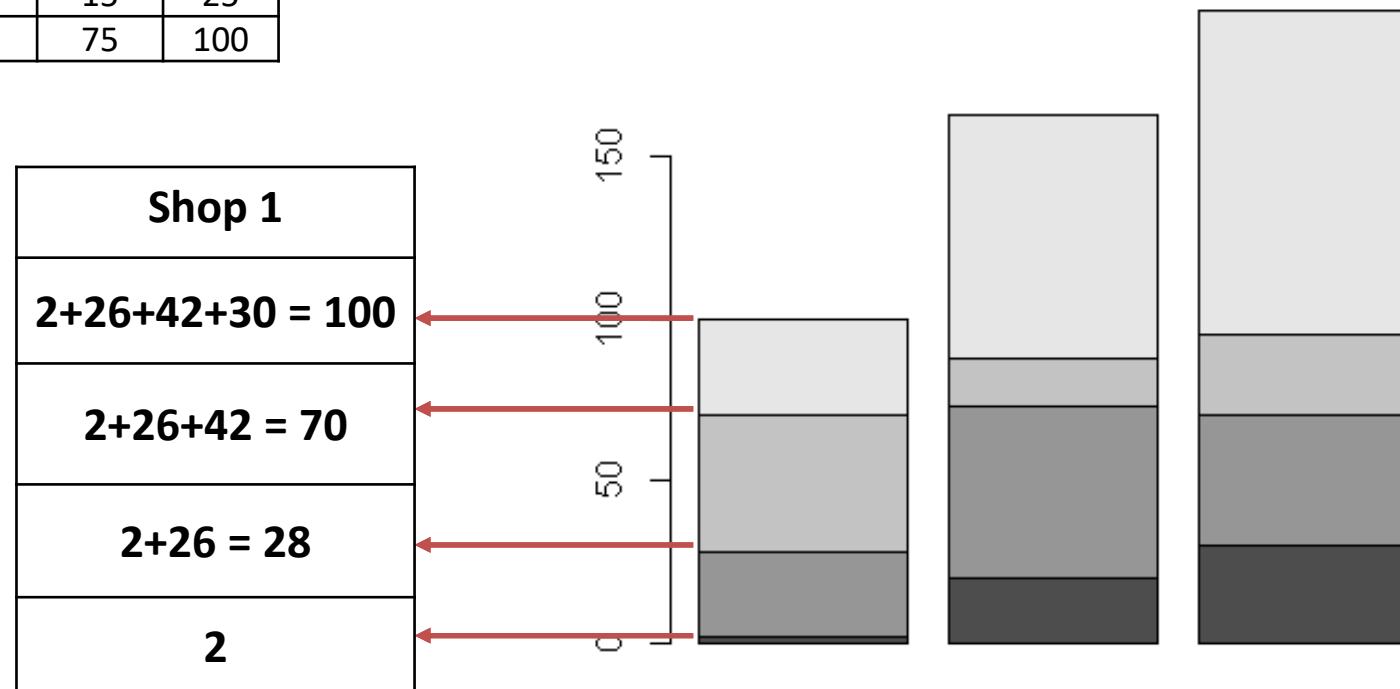
**will create a subdivided or component bar diagram with columns of matrix as bars.**

**Sections inside bars indicate the values in cumulative form.**

# Subdivided or component bar diagram

> `barplot(cust)`

No. of customers	Shop 1	Shop 2	Shop 3
Day 1	2	20	30
Day 2	26	53	40
Day 3	42	15	25
Day 4	30	75	100

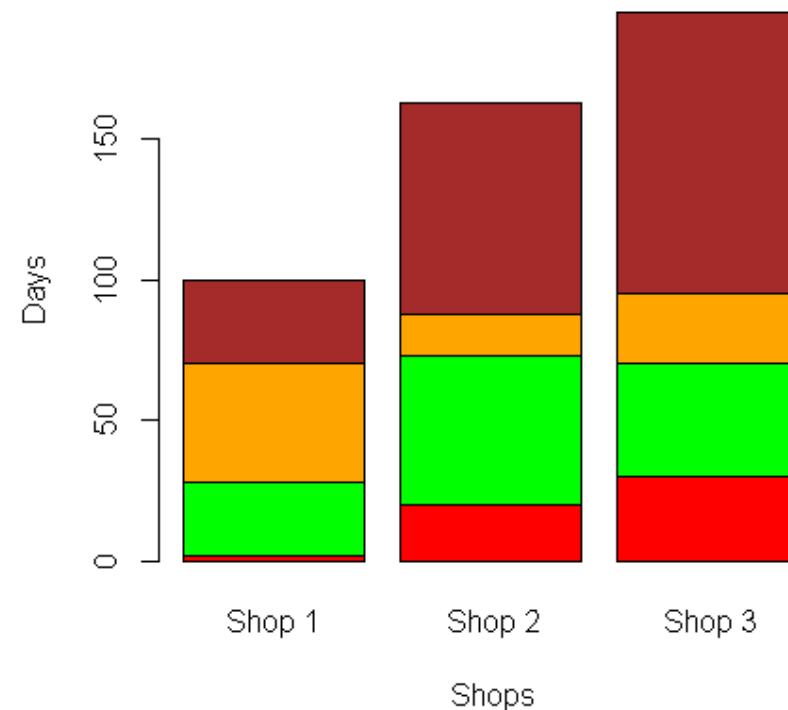


# Subdivided or component bar diagram

Adding labels and colours

```
> barplot(cust, names.arg=c("Shop 1", "Shop 2", "Shop 3"), xlab = "Shops", ylab = "Days", col= c("red","green","orange","brown"))
```

No. of customers	Shop 1	Shop 2	Shop 3
Day 1	2	20	30
Day 2	26	53	40
Day 3	42	15	25
Day 4	30	75	100



# Pie diagram:

Pie charts visualize the absolute and relative frequencies.

A pie chart is a circle partitioned into segments where each of the segments represents a category.

The size of each segment depends upon the relative frequency and is determined by the angle (frequency  $\times$  360°).

```
pie(x, labels = names(x), ...)
```

# Pie diagram:

Example:

Code the 10 persons by using, say 1 for male (M) and 2 for female (F).

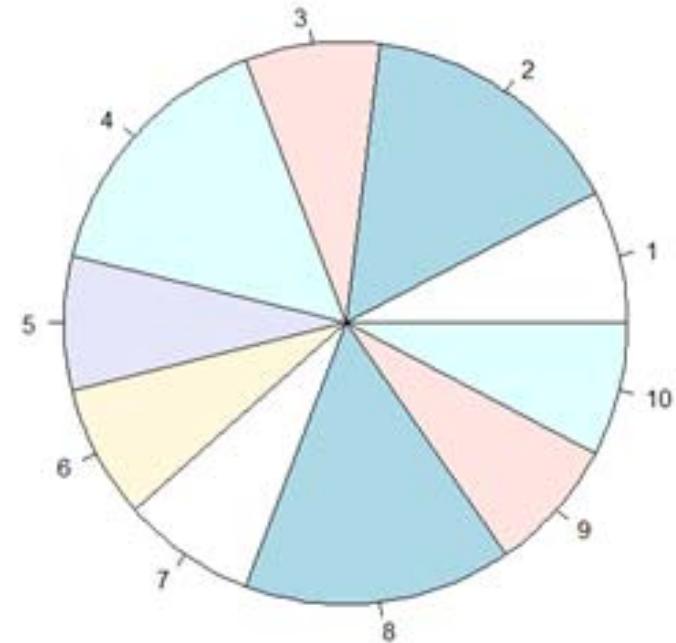
M, F, M, F, M, M, M, F, M, M  
1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1

```
> gender = c(1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1)
```

```
> gender  
[1] 1 2 1 2 1 1 1 1 2 1 1
```

```
> pie(gender)
```

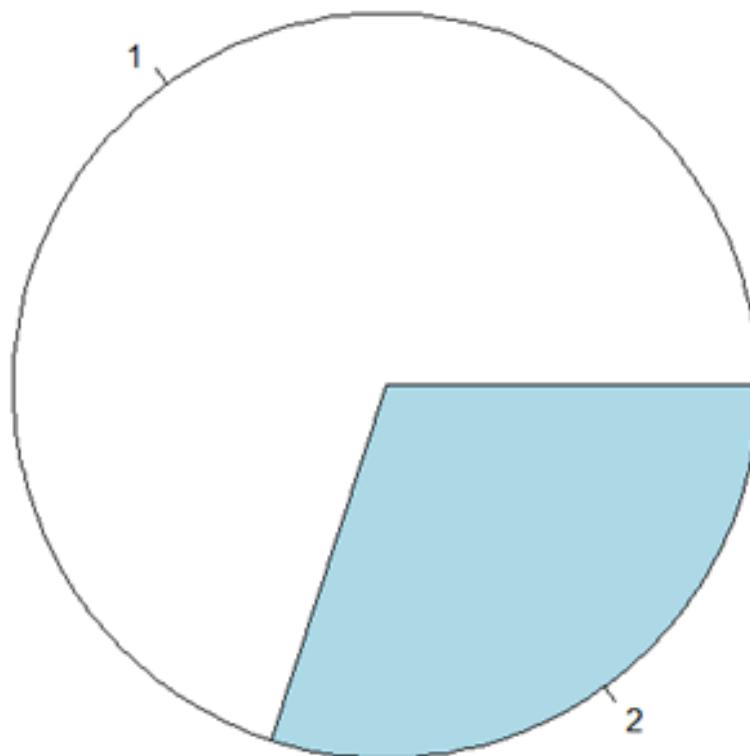
Do you want this?



# Pie diagram:

Example

```
pie(table(gender))
```



# Pie diagram:

## Example:

Consider the following data on pizza home delivery.

- There are three branches (East - coded as 1, West - coded as 2, Central - coded as 3) of the restaurant.

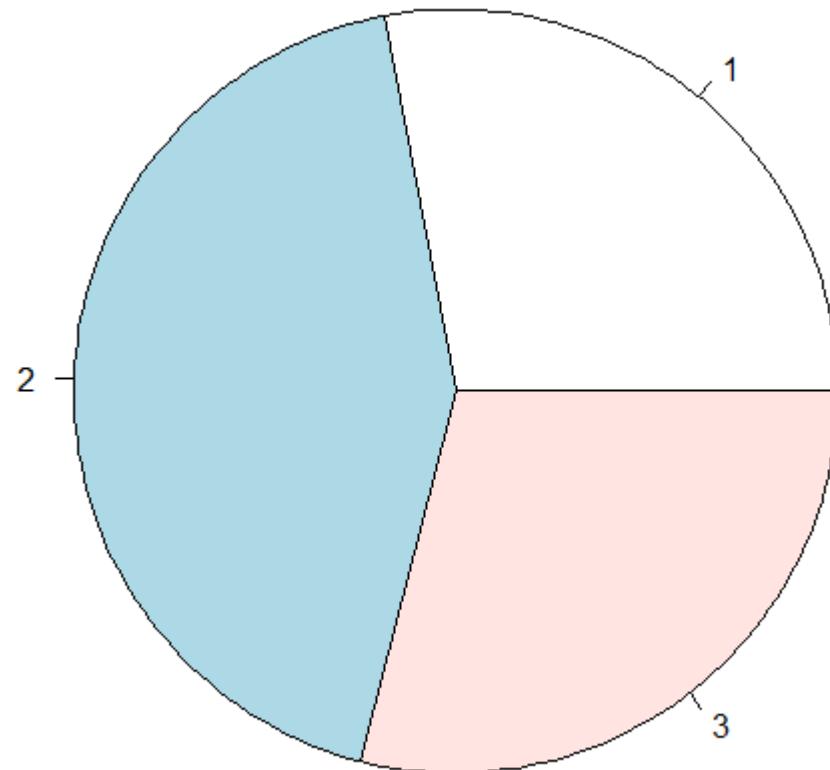
The 100 values from code Directions are as follows:

```
direction =  
c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2  
,1,3,3,3,2,2,2,2,1,2,2,1,1,1,3,2,2,1,2,3,2,2,1,  
2,3,3,2,1,2,2,3,1,1,2,1,2,3,2,3,2,2,3,1,2,3,3,3  
,2,1,1,1,2,1,1,2,1,2,3,3,1,2,3,3,2,1,2,3,2,1,3,  
2,2,2,2,3,2,2)
```

# Pie diagram:

Example

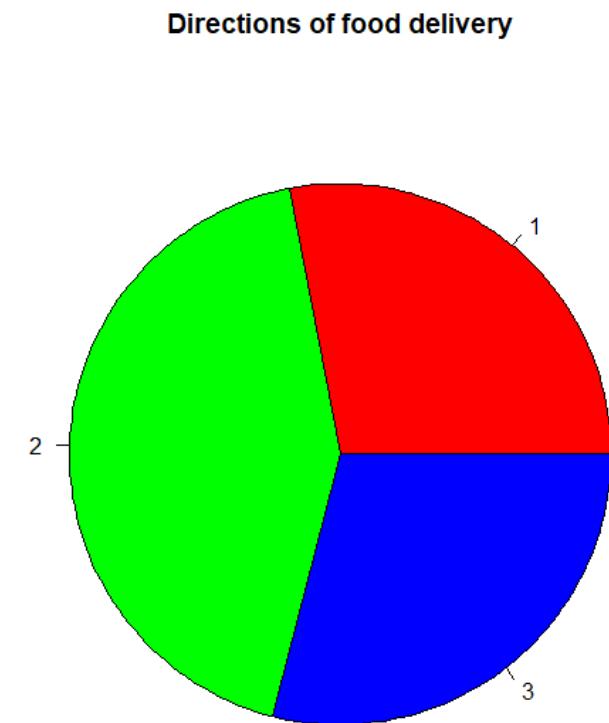
```
pie(table(direction))
```



# Pie diagram:

Example: Adding colours and main title

```
pie(table(direction), col=c("red", "green",  
"blue"), main="Directions of food delivery")
```



# Combining graphics:

Use command `par()` to put multiple graphs in a single plot.

Adjust the graphical parameters with the help of function.

```
par(mfrow=c(p,q))    # set the plotting area into a p*q array

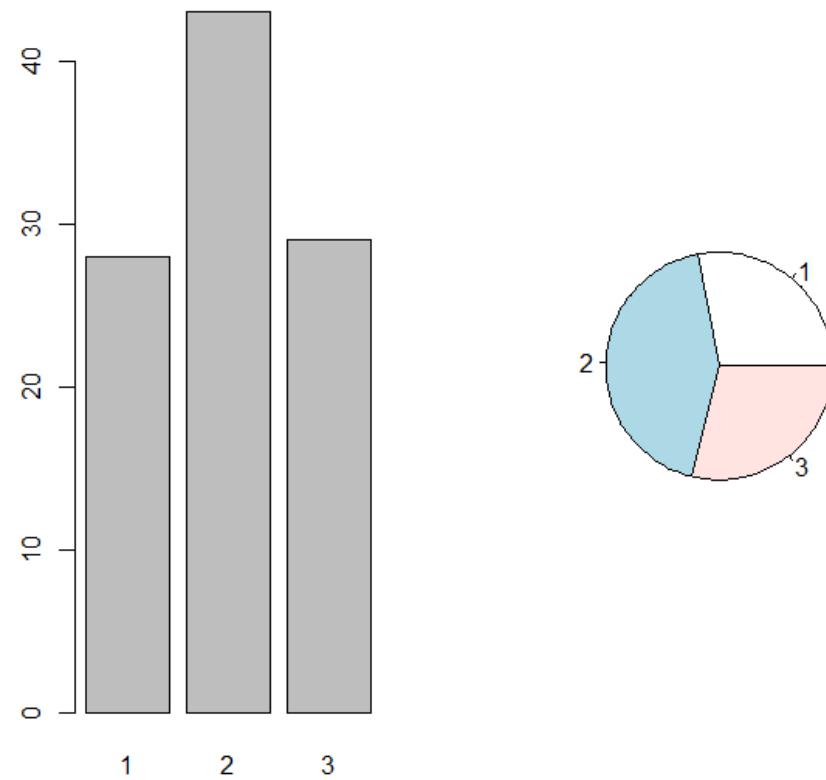
direction = c(1,1,2,1,2,3,2,2,3,3,3,1,2,3,2,2,3,1,1,3,3,1,2,
1,3,3,3,2,2,2,2,1,2,2,1,1,1,3,2,2,1,2,3,2,2,1,2,3,3,2,1,2,2,3
,1,1,2,1,2,3,2,3,2,2,3,1,2,3,3,3,2,1,1,1,2,1,1,2,1,2,3,3,1,2,
3,3,2,1,2,3,2,1,3,2,2,2,3,2,2)

par(mfrow=c(1,2))    # set the plotting area into a 1*2 array

barplot(table(direction))

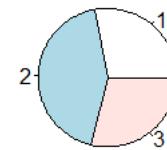
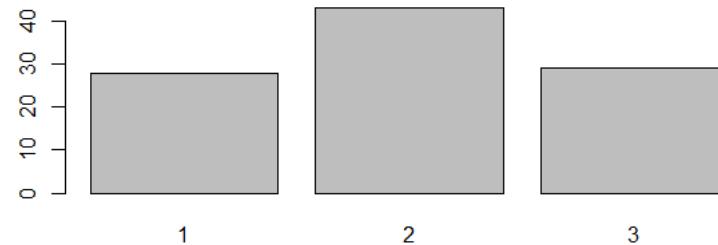
pie(table(direction))
```

# Combining graphics:



# Combining graphics:

```
par(mfrow=c(2,1)) # set the plotting area into a 2*1 array  
  
barplot(table(direction))  
  
pie(table(direction))
```



# **Foundations of R Software**

## **Lecture 51** **Graphics**

⋮

### **Histogram**

Shalabh

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# Histogram:

Histogram is based on the idea to categorize the data into different groups and plot the bars for each category with height.

Data is continuous.

The area of the bars (= height X width) is proportional to the frequency (or relative frequency).

So the widths of the bars need not necessarily to be the same

# Histogram:

```
hist(x) # show absolute frequencies
```

```
hist(x, freq=F) # show relative frequencies
```

# Histogram:

```
hist(x, main, col, xlab, xlim, ylim)
```

**x** : Vector containing numeric values used in histogram.

**main** : Title of the chart.

**col** : Set colour of the bars.

**xlab** : Description of x-axis.

**xlim** : Specifies the range of values on x-axis.

**ylim** : Specifies the range of values on y-axis.

See `help("hist")` for more details

# Histogram:

## Example

Height of 50 persons in centimetres are recorded as follow

166,125,130,142,147,159,159,147,165,156,149,164,137,166,135,142,  
133,136,127,143,165,121,142,148,158,146,154,157,124,125,158,159,  
164,143,154,152,141,164,131,152,152,161,143,143,139,131,125,145,  
140,163

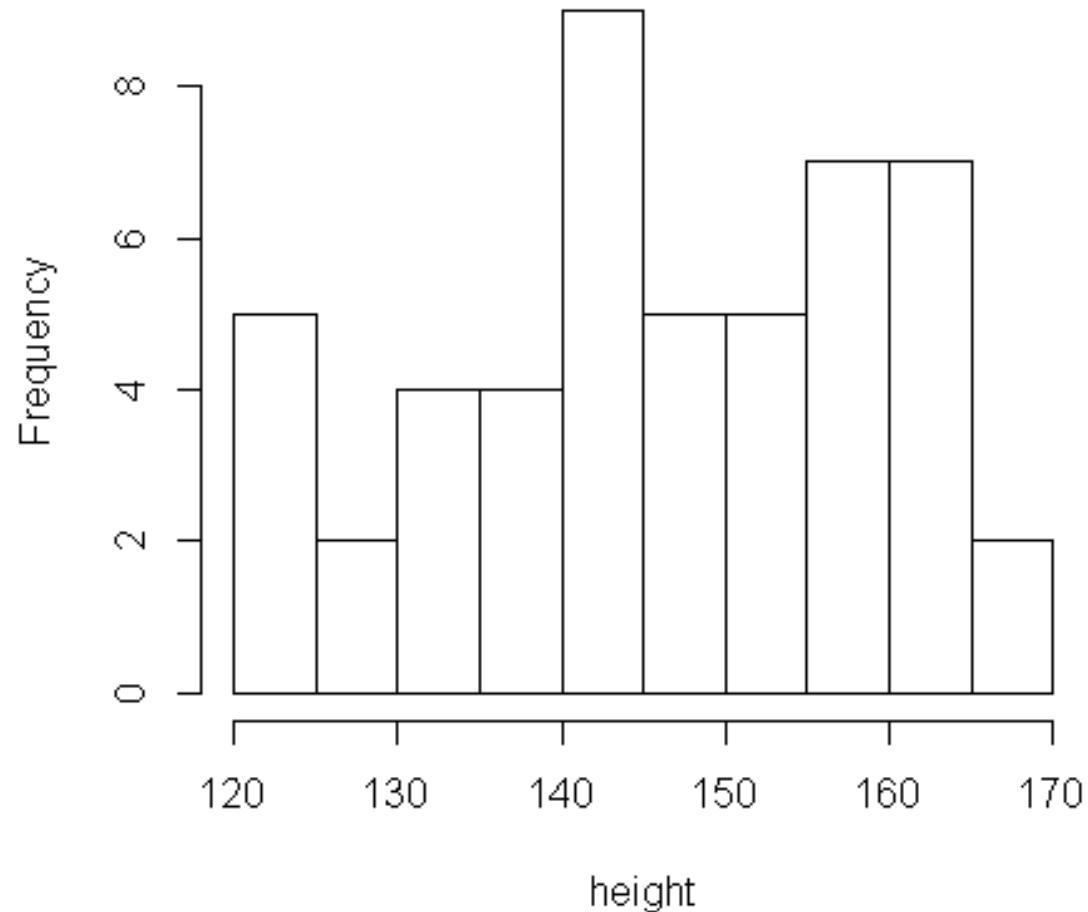
```
height = c(166,125,130,142,147,159,159,147,  
165,156,149,164,137,166,135,142,133,136,127,143,  
165,121,142,148,158,146,154,157,124,125,158,159,  
164,143,154,152,141,164,131,152,152,161,143,143,  
139,131,125,145,140,163)
```

# Histogram:

Example

```
hist(height)
```

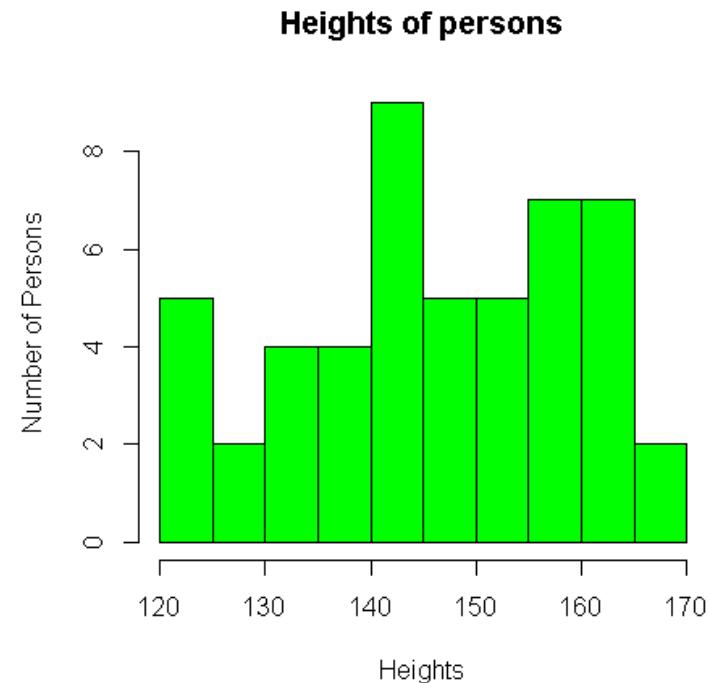
**Histogram of height**



# Histogram

**Example:** Adding colour to bars and titles on axis

```
hist(height, main = "Heights of persons", col  
= "green", xlab = "Heights", ylab = "Number  
of Persons")
```

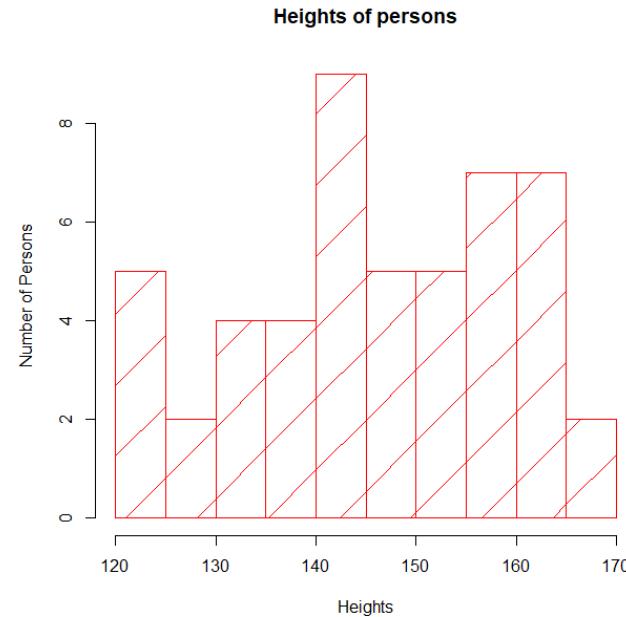


# Histogram

**Example:** Adding density

**density** : Density of shading lines, in lines per inch. Non-positive values of density also inhibit the drawing of shading lines.

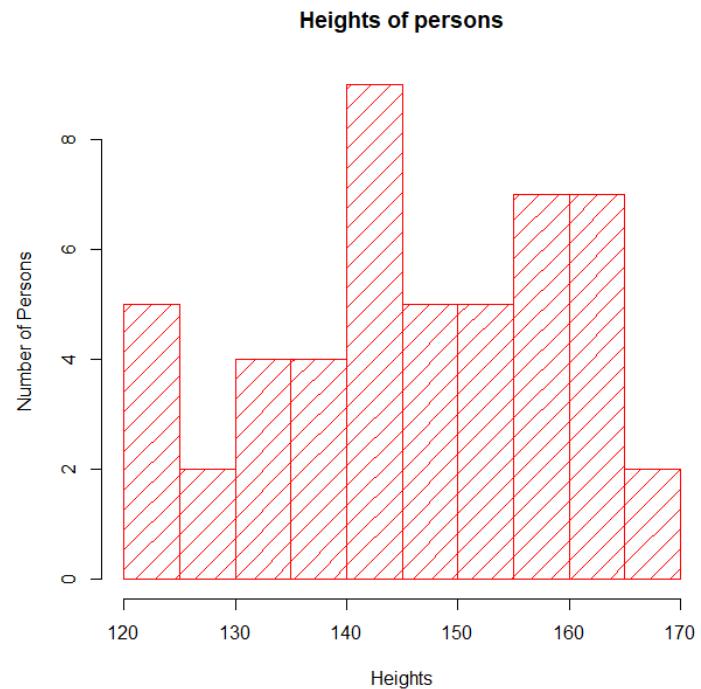
```
hist(height, main = "Heights of persons", col  
= "red", xlab = "Heights", ylab = "Number of  
Persons", density = 2)
```



# Histogram

**Example: Increasing density**

```
hist(height, main = "Heights of persons", col  
= "red", xlab = "Heights", ylab = "Number of  
Persons", density = 8)
```

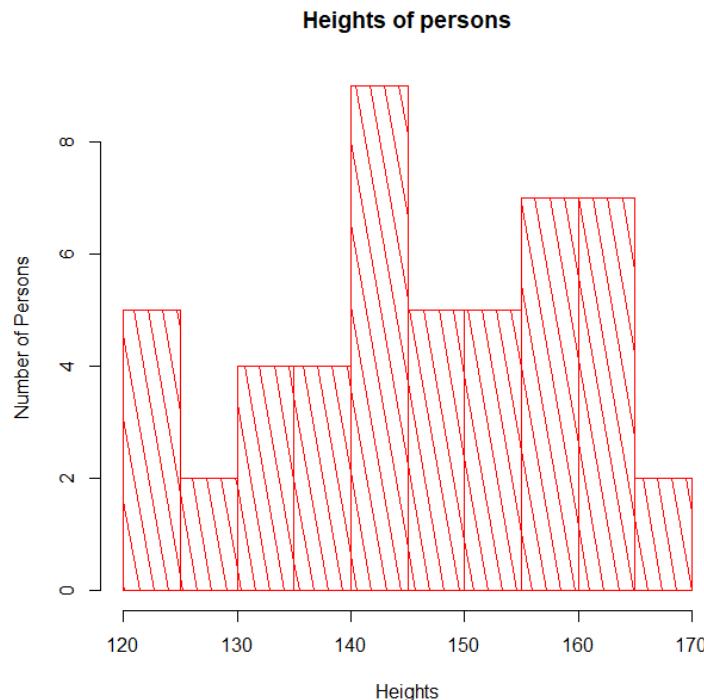


# Histogram

**Example:** Adding angle

**angle** : the slope of shading lines, given as an angle in degrees (counter-clockwise).

```
hist(height, main = "Heights of persons", col = "red", xlab = "Heights", ylab = "Number of Persons", density = 8, angle=100)
```



# **Foundations of R Software**

## **Lecture 52**

## **Graphics**

**:::**

## **Bivariate and Three Dimensional Scatter Plots**

**Shalabh**

**Department of Mathematics and Statistics**  
**Indian Institute of Technology Kanpur**

# **Association of two variables:**

## **Example**

- Number of hours of study affect the marks obtained in an examination.
- Electricity/power consumption increases when the weather temperature increases.
- Weight of infants and small children increases as their height increases under normal circumstances.

# **Association of two variables:**

The observations on both the variables are related to each other.

How to know the variables are related?

How to know the degree of relationship between the two variables?

**Graphical procedures** – Two dimensional plots, three dimensional plots etc.

**Quantitative procedures** – Correlation coefficients, contingency tables, Chi-square statistic, linear regression, nonlinear regression etc.

# **Association of two variables:**

**How to judge or graphically summarize the association of two variables?**

**$X, Y$  : Two variables**

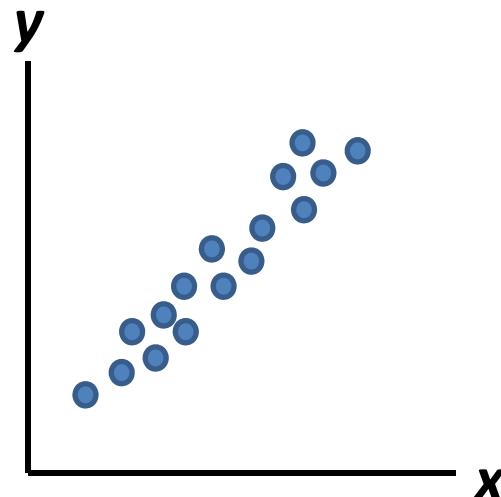
**$n$  pairs of observations are available as  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$**

# Scatter plots:

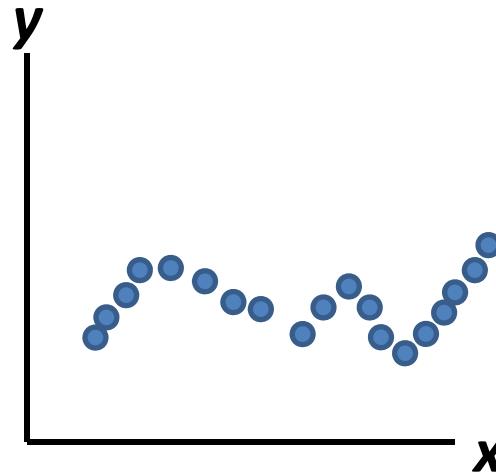
Plot the paired observations in a single graph, called as scatter plot.

Scatter plot reveals the nature and trend of possible relationship.

Relationships : Linear or nonlinear.



Linear relationship between  $X$  and  $Y$



Nonlinear relationship between  $X$  and  $Y$

# Strength and trend of relationships:

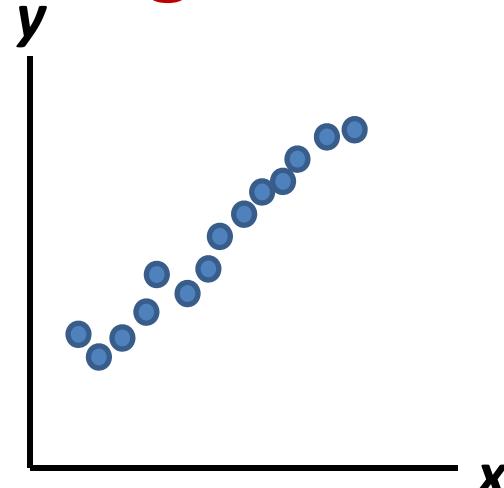


Fig. 1: Strong positive linear relationship

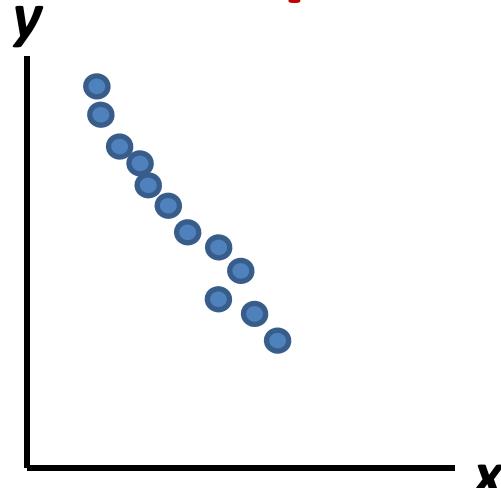


Fig. 2: Strong negative linear relationship

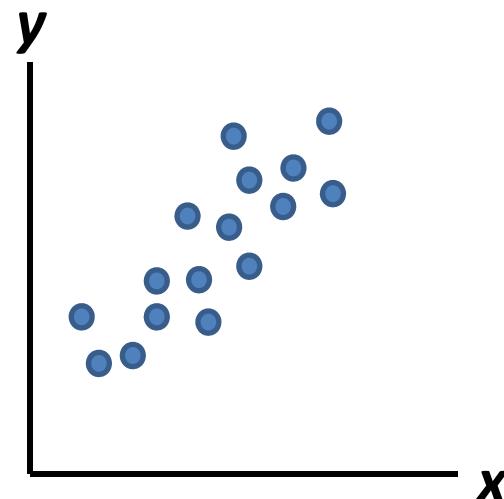


Fig. 3: Moderate positive linear relationship

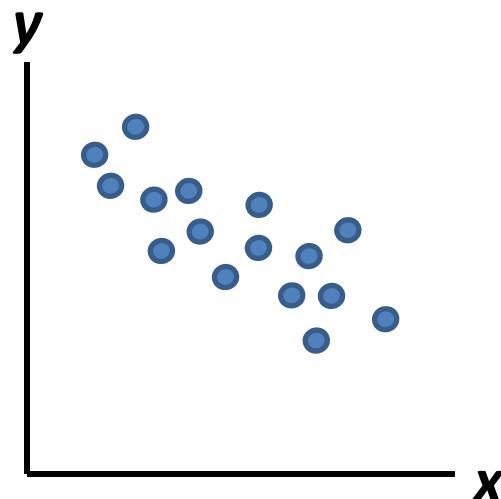


Fig. 4: Moderate negative linear relationship

# Strength and trend of relationships:

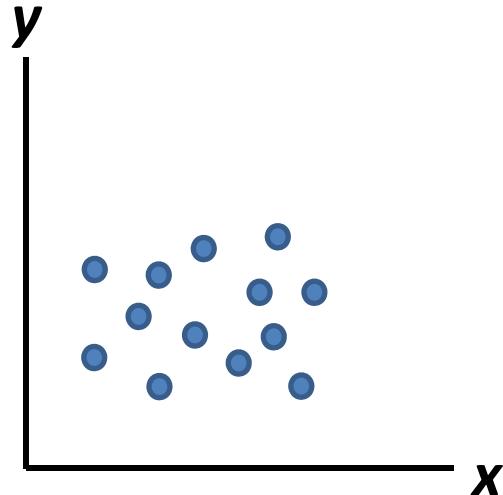


Fig. 5: No clear relationship

We will study about the direction and degree of linear relationships.

Two aspects – graphical and quantitative

## **Bivariate plots:**

**Provide first hand visual information about the nature and degree of relationship between two variables.**

**Relationship can be linear or nonlinear.**

**We discuss several types of plots through examples.**

# Bivariate scatter plots:

Plot command:

**x, y:** Two data vectors

`plot(x, y)`

`plot(x, y, type)`

type	
<b>"p"</b> for <u>points</u>	<b>"l"</b> for <u>lines</u>
<b>"b"</b> for <u>both</u>	<b>"c"</b> for the lines part alone of " <b>b</b> "
<b>"o"</b> for both ' <u>overplotted</u> '	<b>"s"</b> for <u>stair steps</u> .
<b>"h"</b> for ' <u>histogram</u> ' like (or ' <u>high-density</u> ') vertical lines	

# Bivariate scatter plots:

Plot command

**x, y:** Two data vectors

`plot(x, y)`

`plot(x, y, type)`

Get more details from help: `help("type")`

Other options:

**main** an overall title for the plot.

**suba** sub title for the plot.

**xlab** title for the x axis.

**ylab** title for the y axis.

**aspthe** y/x aspect ratio.

# Bivariate scatter plots:

## Example

Data on marks obtained by 20 students out of 500 marks and the number of hours they studied per week are recorded as follows:

We know from experience that marks obtained by students increase as the number of hours increase.

Marks	337	316	327	340	374	330	352	353	370	380
Number of hours per week	23	25	26	27	30	26	29	32	33	34

Marks	384	398	413	428	430	438	439	479	460	450
Number of hours per week	35	38	39	42	43	44	45	46	44	41

# Bivariate scatter plots:

## Example

```
marks =  
c(337,316,327,340,374,330,352,353,370,380,384,39  
8,413,428,430,438,439,479,460,450)  
  
hours =  
c(23,25,26,27,30,26,29,32,33,34,35,38,39,42,43,4  
4,45,46,44,41)
```

# Bivariate scatter plots:

Example

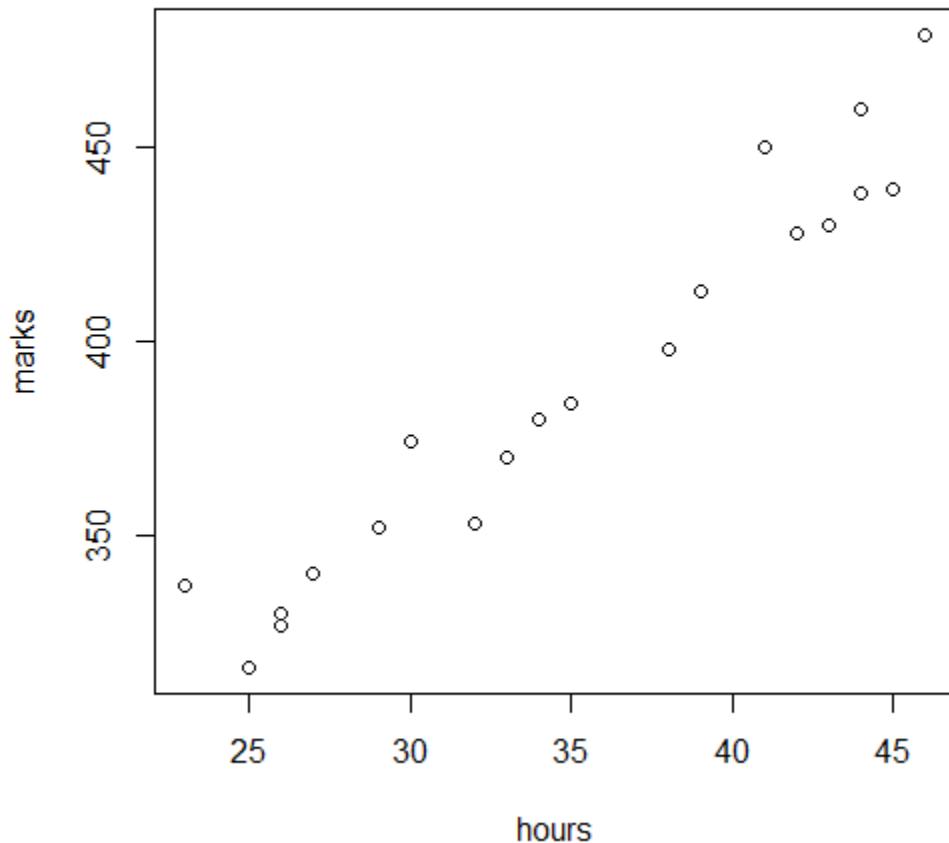
**plot command:**

**x, y:** Two data vectors

Various type of plots are possible to draw.

**plot(x, y)**

**plot(hours, marks)**

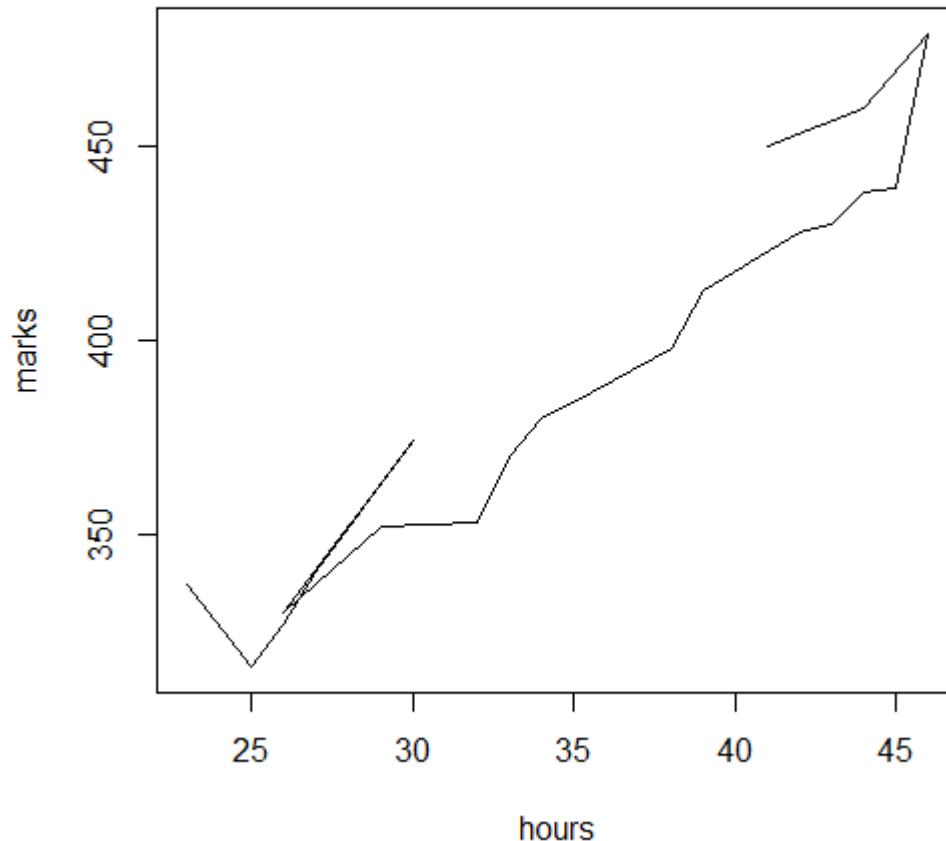


# Bivariate scatter plots:

## Example

```
plot(hours, marks, "l")
```

"l" for lines,

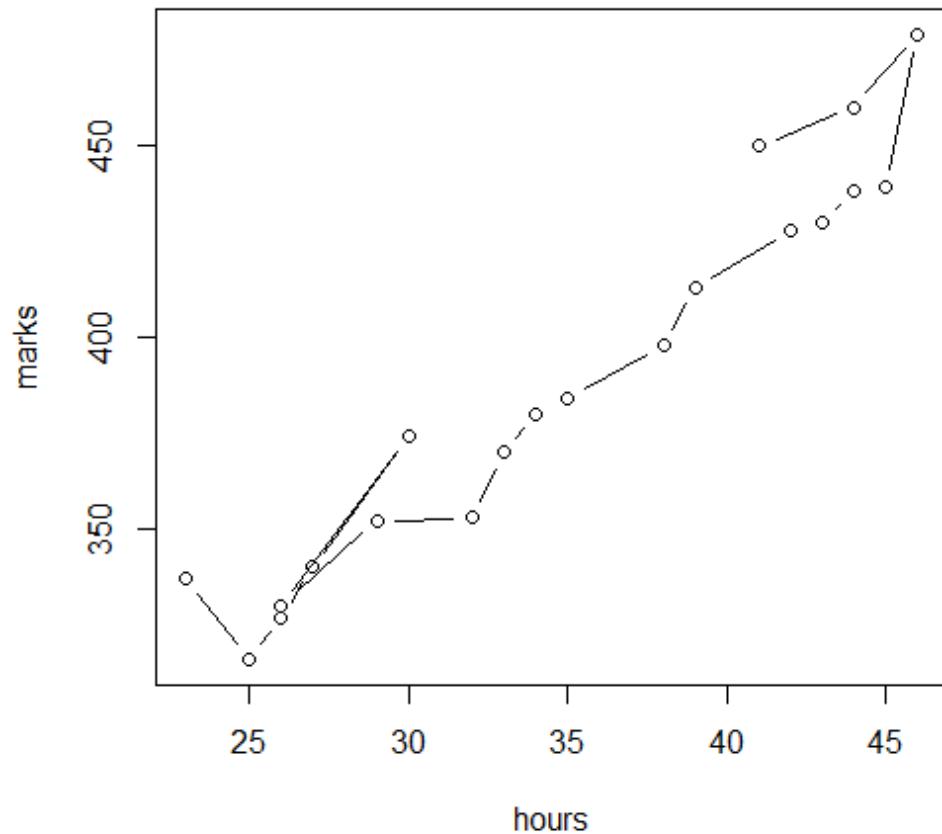


# Bivariate scatter plots:

## Example

```
plot(hours, marks, "b")
```

“b” for both – line and point

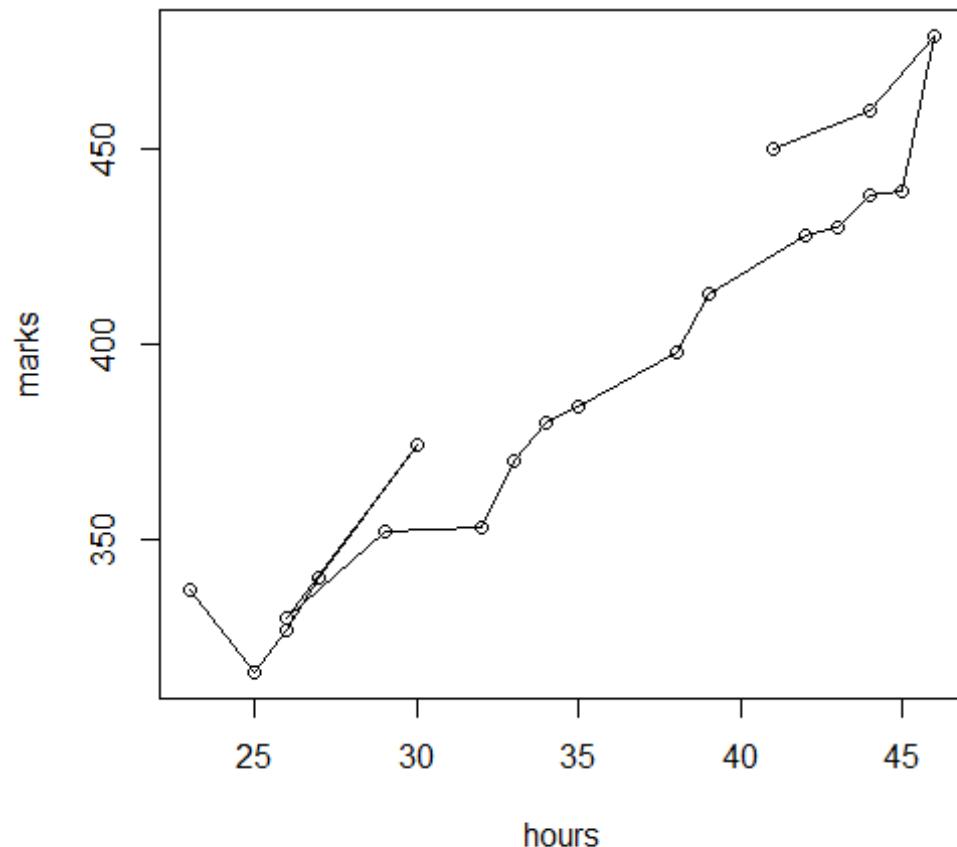


# Bivariate scatter plots:

Example

```
plot(hours, marks, "o")
```

“o” for both ‘overplotted’

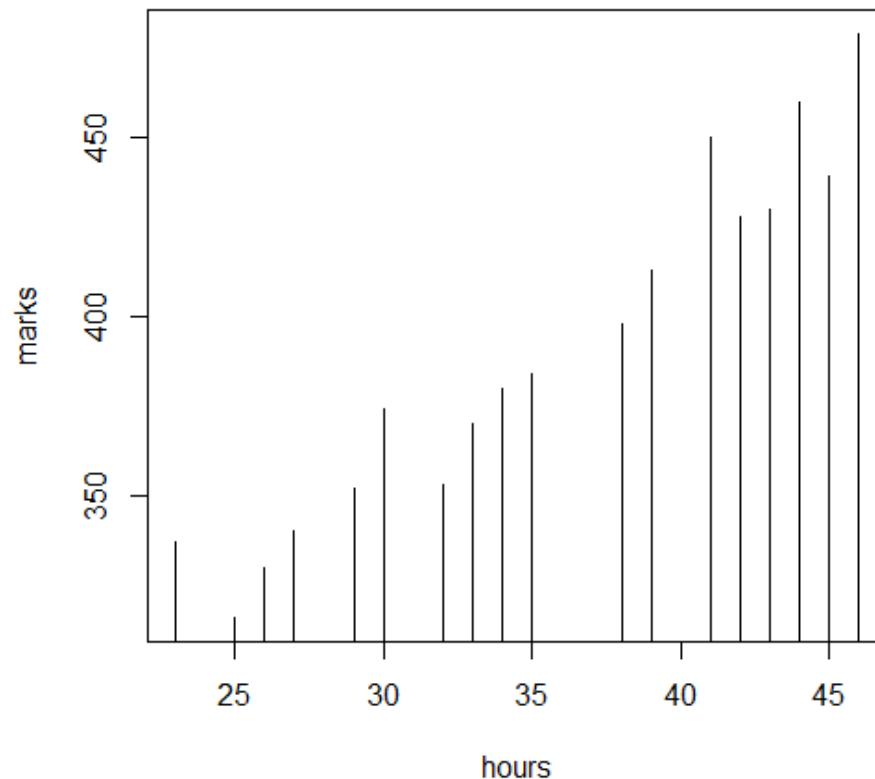


# Bivariate scatter plots:

## Example

```
plot(hours, marks, "h")
```

“h” for ‘histogram’ like (or ‘high-density’) vertical lines

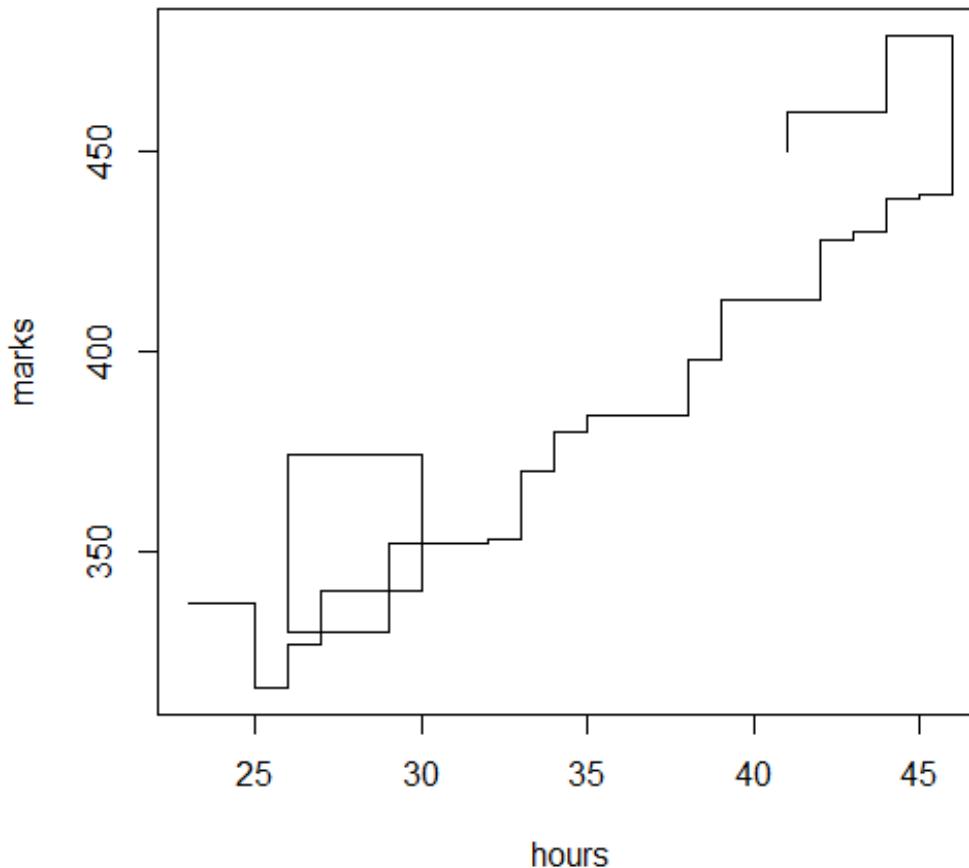


# Bivariate scatter plots:

## Example

```
plot(hours, marks, "s")
```

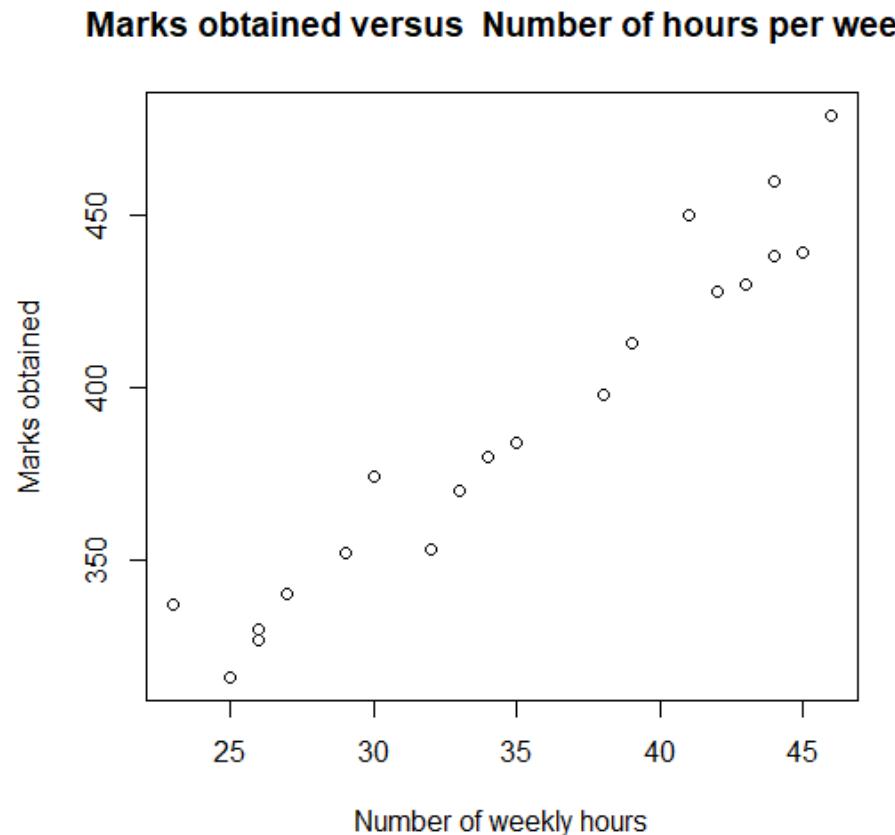
“s” for stair steps.



# Bivariate scatter plots:

## Example

```
plot(hours, marks, xlab="Number of weekly hours", ylab="Marks obtained", main="Marks obtained versus Number of hours per week")
```

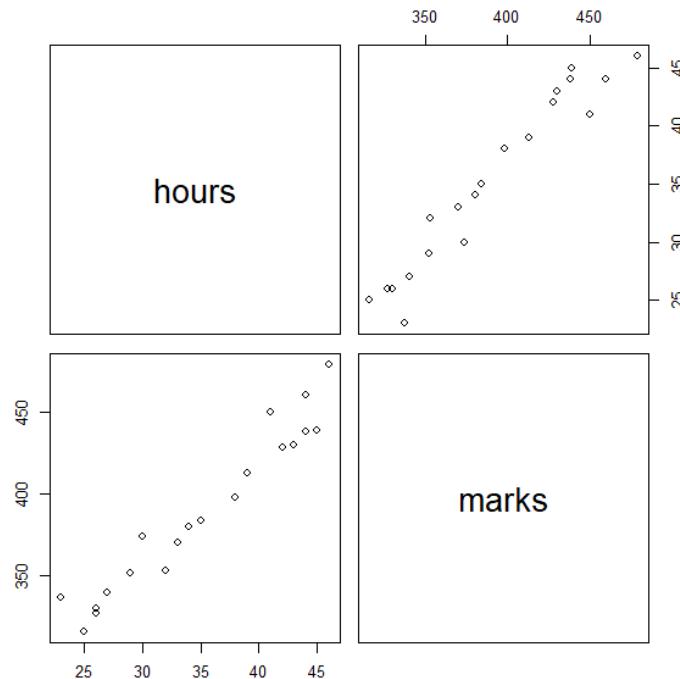


# Matrix Scatter plot:

The command `pairs( )` allows the simple creation of a matrix of scatter plots.

Example

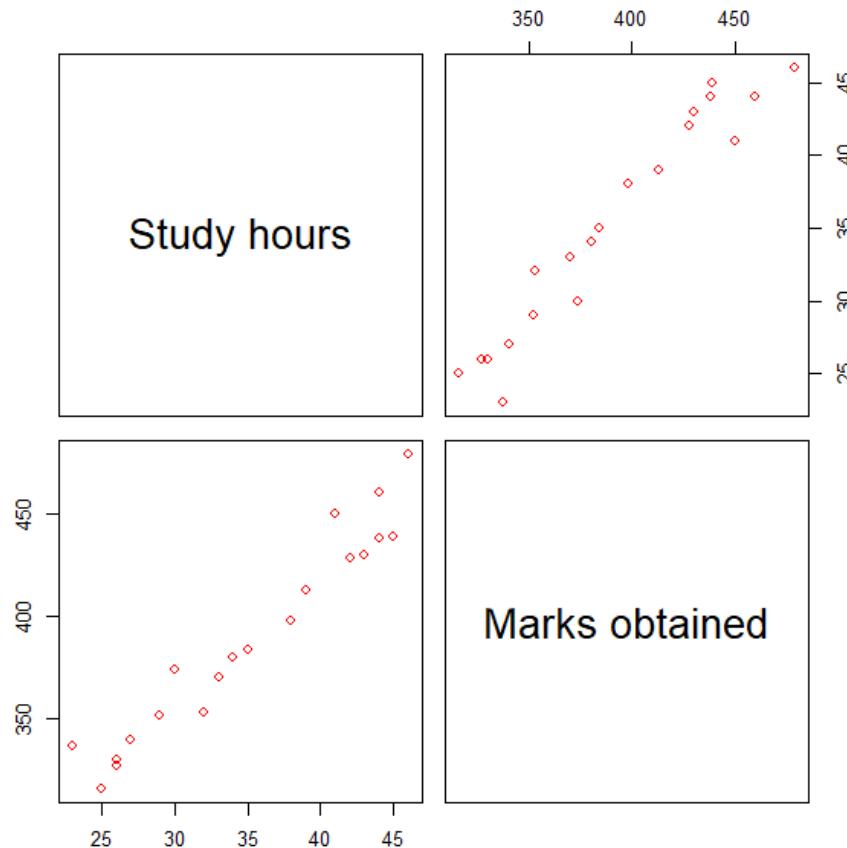
```
pairs(cbind(hours, marks))
```



# Matrix Scatter plot

Example: Adding labels and colour

```
pairs(cbind(hours, marks), labels=c("Study hours", "Marks obtained"), col="red" )
```



# **Scatter plots with smooth curve:**

Suppose two variables are related.

A scatter plot along with a fitted line will provide information on the trend or relationship between them.

**scatter.smooth** produces a scatter plot and adds a smooth curve to the scatter plot.

## **Scatter plots with smooth curve:**

`scatter.smooth` is based on the concept of LOESS which is a locally weighted scatterplot smoothing method.

LOESS is used for local polynomial regression fitting.

Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

Use `help("scatter.smooth")` to get more details.

## Scatter plots with smooth curve:

```
scatter.smooth(x, y = NULL, span = 2/3, degree  
= 1, family = c("symmetric", "gaussian"), xlab =  
NULL, ylab = NULL, ylim = range(y, pred$y,  
na.rm = TRUE),...)
```

<b>x, y</b>	<b>x</b> and <b>y</b> arguments provide the x and y coordinates for the plot.
<b>span</b>	smoothness parameter for LOESS.
<b>degree</b>	degree of local polynomial used.
<b>family</b>	if "gaussian" fitting is by least-squares, and if family = "symmetric" a re-descending M estimator is used.
<b>xlab</b>	label for x axis.
<b>ylab</b>	label for y axis.
<b>ylim</b>	the y limits of the plot.

# Scatter plots with smooth curve:

## Example

Data on marks obtained by 20 students out of 500 marks and the number of hours they studied per week are recorded as follows:

We know from experience that marks obtained by students increase as the number of hours increase.

Marks	337	316	327	340	374	330	352	353	370	380
Number of hours per week	23	25	26	27	30	26	29	32	33	34

Marks	384	398	413	428	430	438	439	479	460	450
Number of hours per week	35	38	39	42	43	44	45	46	44	41

# Scatter plots with smooth curve:

Example

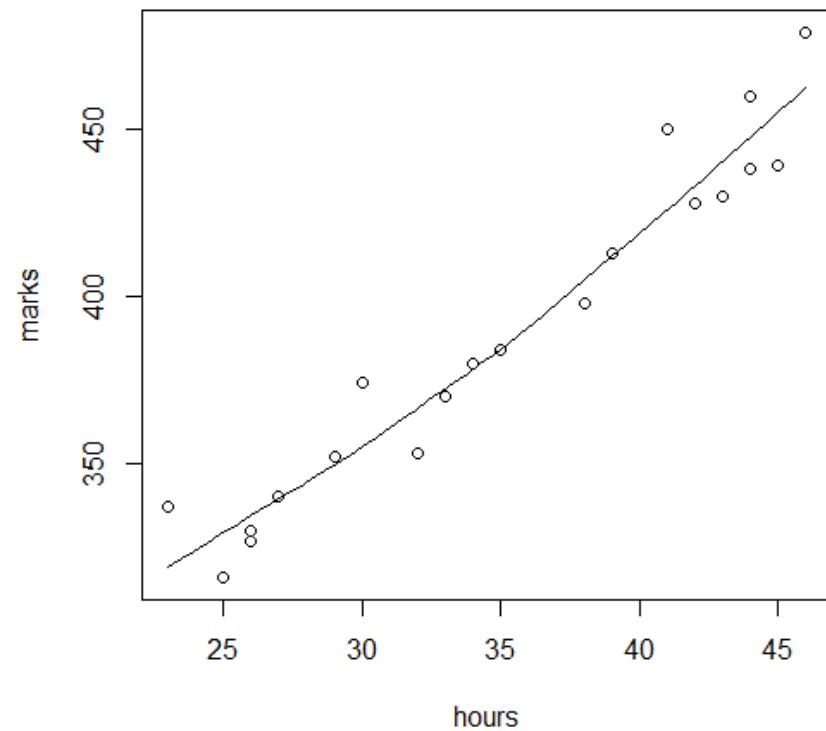
```
marks =  
c(337,316,327,340,374,330,352,353,370,380,384,39  
8,413,428,430,438,439,479,460,450)  
  
hours =  
c(23,25,26,27,30,26,29,32,33,34,35,38,39,42,43,4  
4,45,46,44,41)
```

# Scatter plots with smooth curve:

## Example

`scatter.smooth(x, y)` provides scatter plot with smooth curve

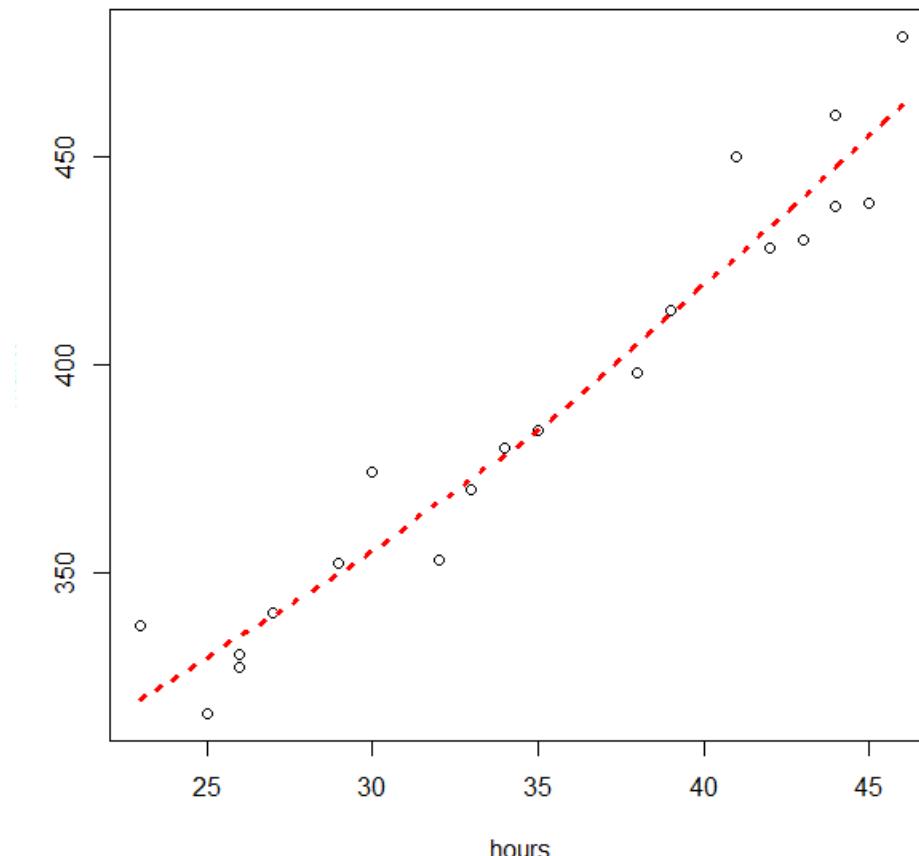
Example: `scatter.smooth(hours, marks)`



# Scatter plots with smooth curve:

Example: Add more options

```
scatter.smooth(hours, marks, lparms = list(col =  
"red", lwd = 3, lty = 3))
```



# Three dimensional scatter plot:

```
scatterplot3d(x,y,z)
```

Plots a three dimensional (3D) point cloud of the data in **x**, **y** and **z**

Need a package **scatterplot3d**

```
install.packages("scatterplot3d")
```

```
library(scatterplot3d)
```

# Three dimensional scatter plot:

## Example

The data on height (in cms.), weight (in kg.) and age (in years) of 5 persons are recorded as follows. We would like to create a 3 dimensional plot for this data.

Person No.	Height (Cms.)	Weight (Kg.)	Age (Years)
1	100	30	10
2	125	35	15
3	145	50	20
4	160	65	30
5	170	70	35

# Three dimensional scatter plot:

**scatterplot3d()** Plots a three dimensional (3D) point cloud

```
install.packages("scatterplot3d")
```

```
library(scatterplot3d)
```

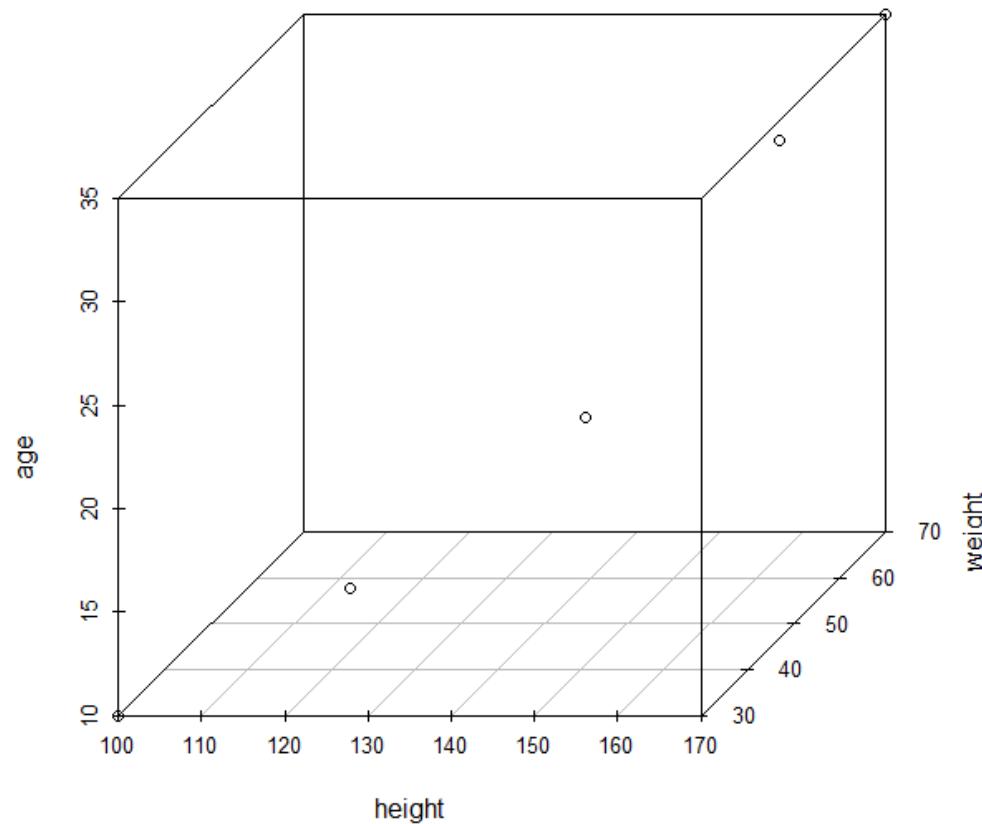
```
height = c(100, 125, 145, 160, 170)
```

```
weight = c(30, 35, 50, 65, 70)
```

```
age = c(10, 15, 20, 30, 35)
```

# Three dimensional scatter plot:

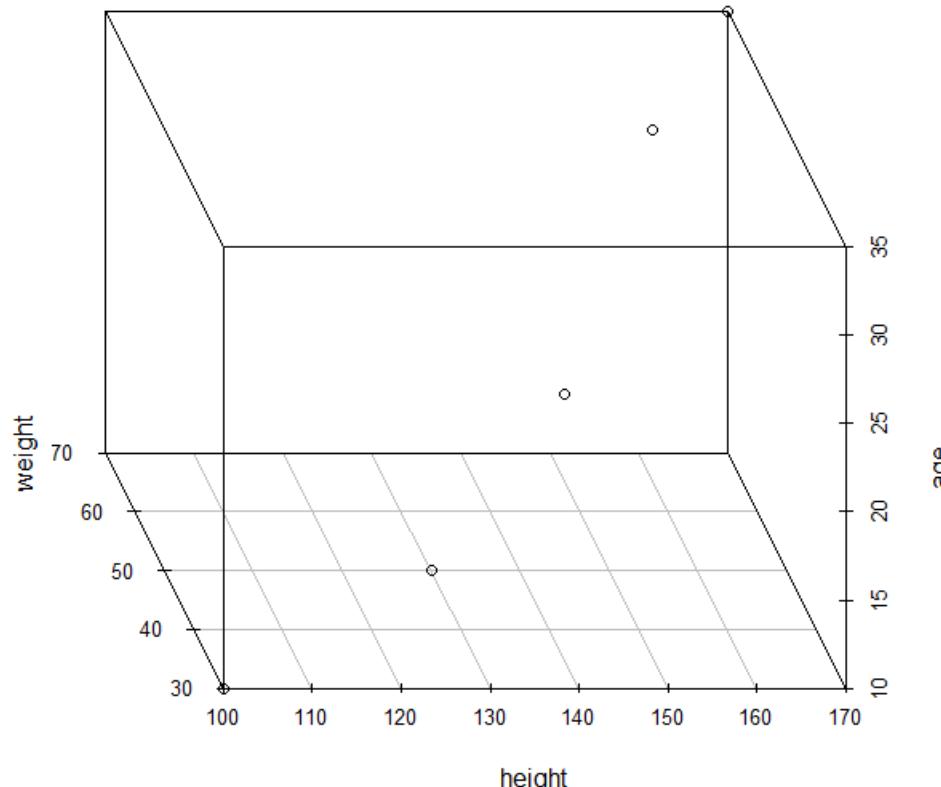
```
scatterplot3d(height, weight, age)
```



# Three dimensional scatter plot:

Direction of the figure can be changed.

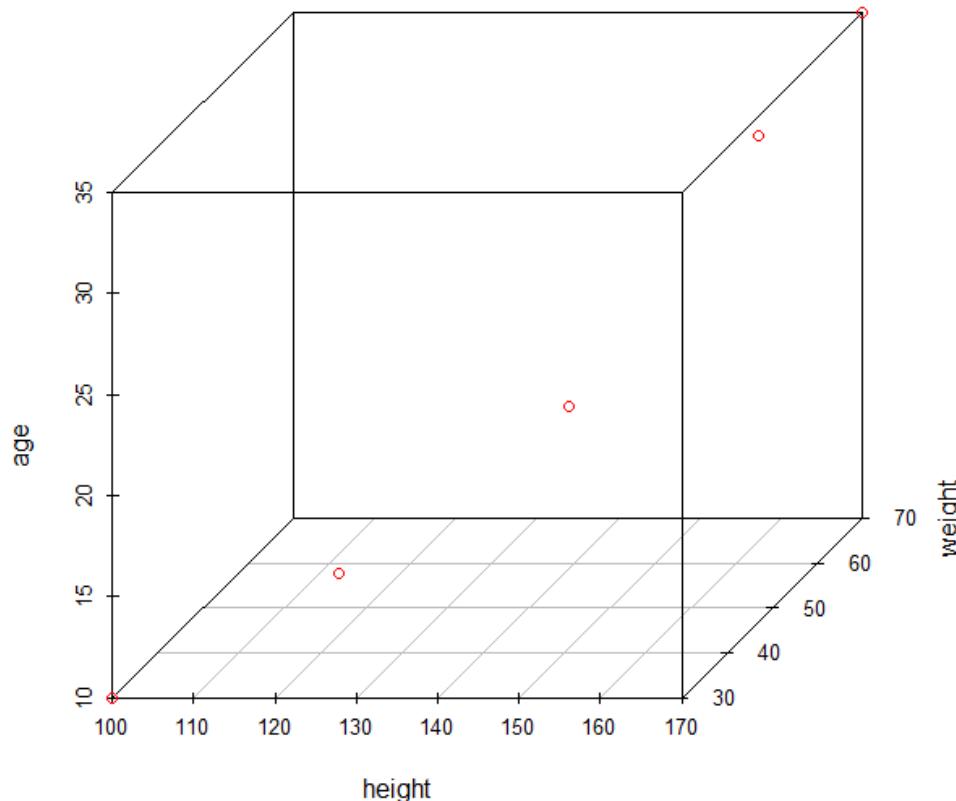
```
scatterplot3d(height, weight, age, angle = 120)
```



# Three dimensional scatter plot:

Colours of points can be changed.

```
scatterplot3d(height, weight, age, color="red")
```



## More functions:

- `contour()` for contour lines
- `dotchart()` for dot charts (replacement for bar charts)
- `image()` pictures with colors as third dimension
- `mosaicplot()` mosaic plot for (multidimensional) diagrams of categorical variables (contingency tables)
- `persp()` perspective surfaces over the x–y plane

# More functions:

## Example of perspective plot

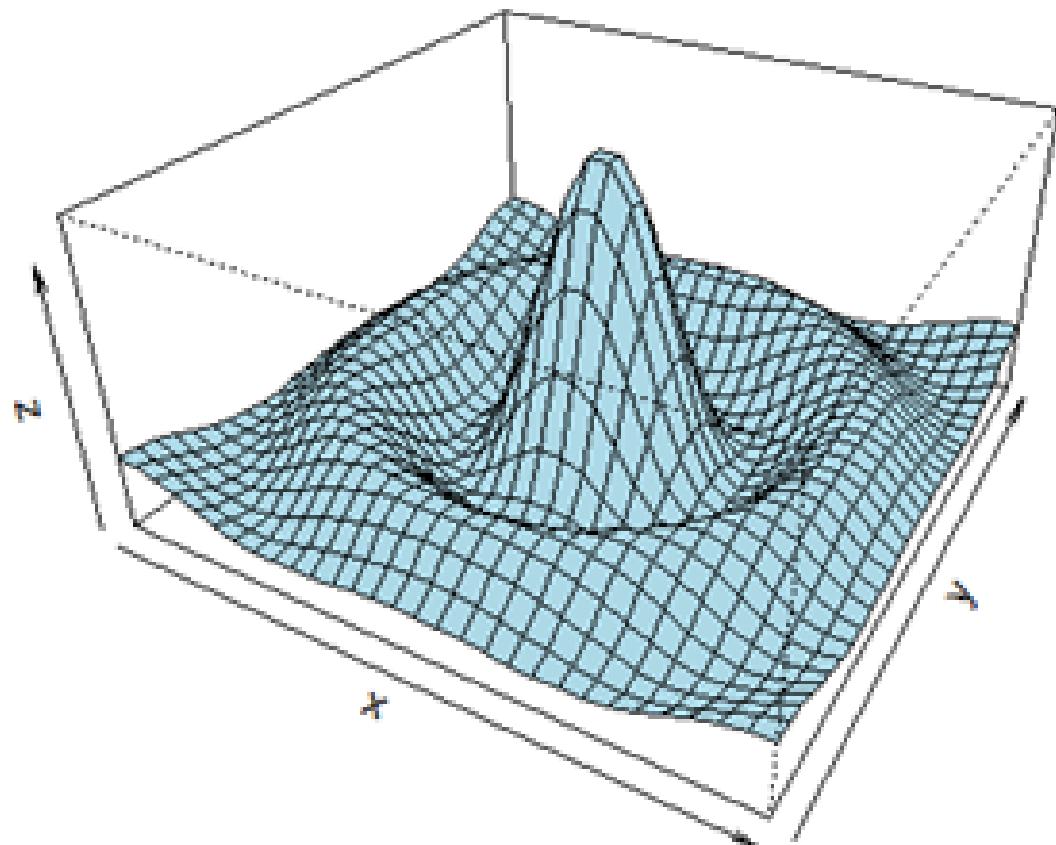
**persp( )** perspective surfaces over the x–y plane

```
x = seq(-10, 10, length= 30)  
  
y = x  
  
f =function(x,y){r = sqrt(x^2+y^2);10*sin(r)/r}  
  
z = outer(x, y, f)  
  
z[is.na(z)] = 1  
  
op = par(bg = "white")
```

# More functions:

## Example of perspective plot

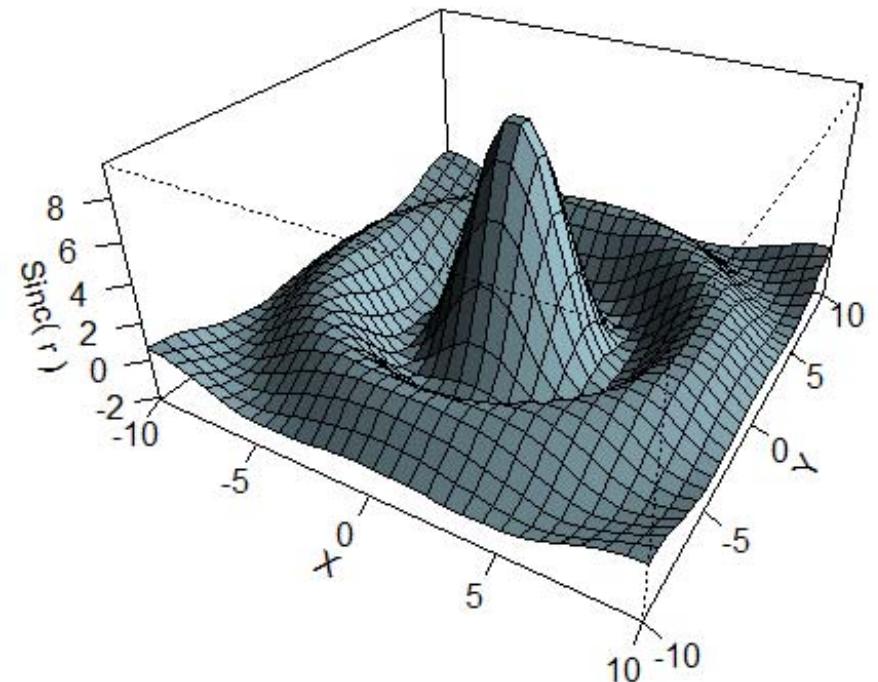
```
persp(x,y,z, theta=30, phi=30, expand=0.5, col="lightblue")
```



## More functions:

### Example of perspective plot

```
persp(x, y, z, theta = 30, phi = 30, expand =  
0.5, col = "lightblue", ltheta = 120, shade =  
0.75, ticktype = "detailed", xlab = "x", ylab =  
"y", zlab = "Sinc( r )")
```



# **Foundations of R Software**

## **Lecture 53**

### **Some Examples of R Programming**

**Shalabh**

**Department of Mathematics and Statistics  
Indian Institute of Technology Kanpur**

# **Steps to write a programme**

- A programme is a set of instructions or commands which are written in a sequence of operations i.e., what comes first and what comes after that.
  
- The objective of a programme is to obtain a defined outcome based on input variables.
  
- The computer is instructed to perform the defined task.

# **Steps to write a programme**

- Computer is an obedient worker but it has its own language.
- We do not understand computer's language and computer does not understand our language.
- The software help us and works like an interpreter between us and computer.

# **Steps to write a programme**

- We say something in software's language and software informs it to computer.
- Computer does the task and informs back to software.
- The software translates it to our language and informs us.

# Steps to write a programme

- Programme in R is written as a function using **function**.
- Write down the objective, i.e., what we want to obtain as an outcome.
- Translate it in the language of R.
- Identify the input and output variables.
- Identify the nature of input and output variables, i.e., numeric, string, factor, matrix etc.

# Steps to write a programme

- Input and output variables can be single variable, vector, matrix or even a function itself.
- The input variables are the component of **function** which are reported in the argument of **function( )**
- The output of a **function** can also be input to another **function**.
- The output of an outcome can be formatted as per the need and requirement.

# Steps to write a programme

Tips:

- ❖ Loops usually slower the speed of programmes, so better is to use vectors and matrices.
- ❖ Use **#** symbol to write comment to understand the syntax.
- ❖ Use the variable names which are easy to understand.
- ❖ Don't forget to initialize the variables.

## Example 1

Suppose we want to compute

$$\frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n y_i^2} \quad \text{and} \quad \sum_{i=1}^n \left( \frac{x_i}{y_i} \right)^2$$

Data  $x_1, x_2, \dots, x_n$   $y_1, y_2, \dots, y_n$

**x, y:** Two data vectors

# Example 1

**Input variables :** `x, y, n` (if `x` and `y` have different number of observations, choose different numbers, say `n1` and `n2`)

**Output variables:** `g, h,`    
$$g = \frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n y_i^2} \quad \text{and} \quad h = \sum_{i=1}^n \left( \frac{x_i}{y_i} \right)^2$$

We need summation, so use `sum` function or alternatively compute it through vectors.

## Example 1

```
# Remove all data
rm(list = ls())

# Define input data vectors, for example
x = c(10,20,30)
y = c(1,2,3)

+++++START OF FUNCTION+++++
example1 = function(x,y)

# Start of function body
{
# First give all other input variables

# Computation of number of observations
n = length(x)
```

# Example 1

CONTD...

```
#Initialize the values to store squared values
x1 = 0
y1 = 0
z1 = 0

#Start of loop
for (i in 1:n)
{
# Define x1, y1 and z1 to store their squares
x1[i] = x[i]^2
y1[i] = y[i]^2
z1[i] = (x[i]/y[i])^2
#End of loop
}
```

CONTD...

# Example 1

CONTD...

```
# Obtain the sum of squared quantities
sum_square_x = sum(x1)
sum_square_y = sum(y1)
sum_square_z = sum(z1)

# Computation of g and h
g = sum_square_x/sum_square_y
h = sum_square_z

# Format the output
cat("The value of g and h are", g, "and", h,
"\n", )
}

+++++END OF FUNCTION+++++
```

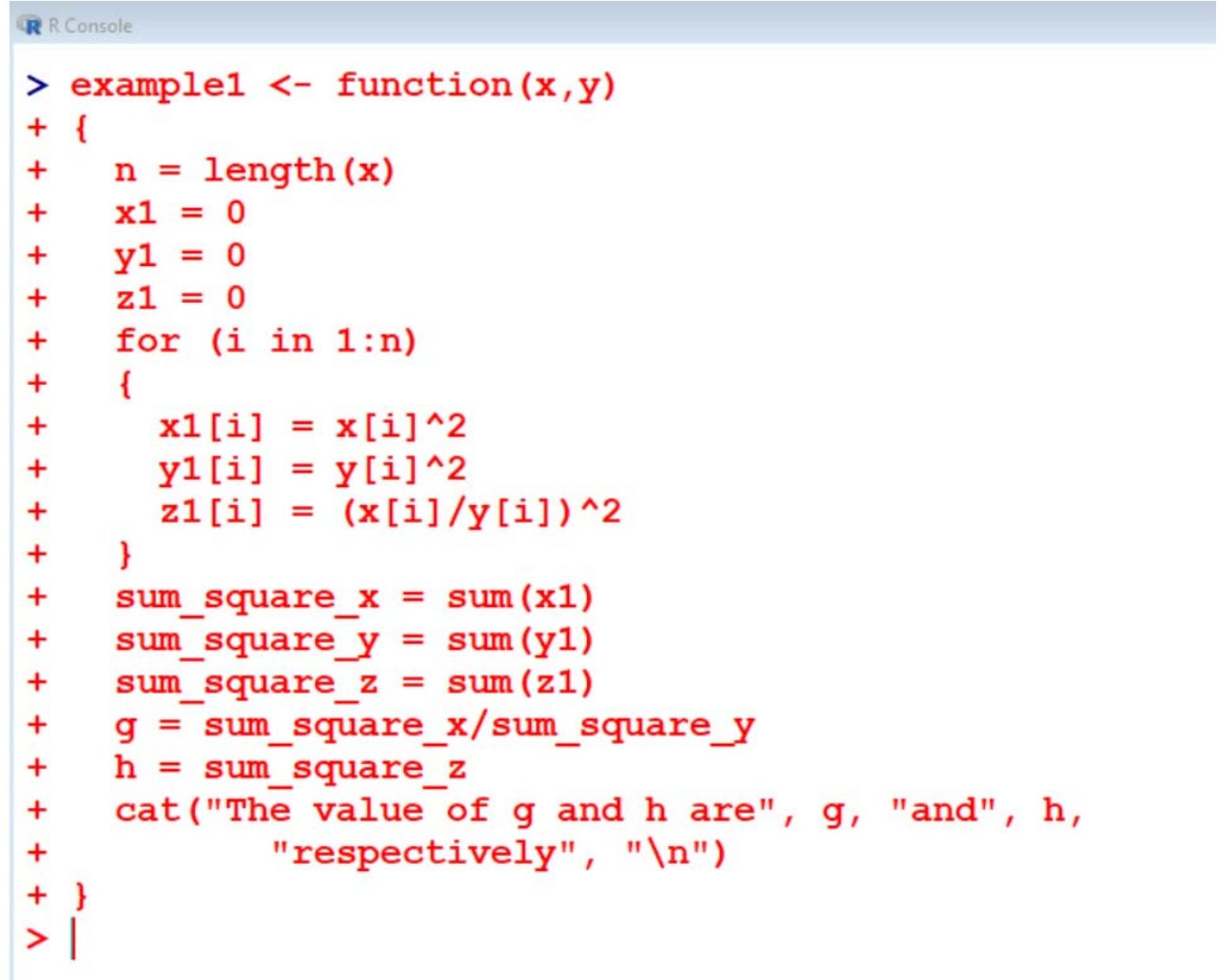
## Example 1: At a glance

```
example1 <- function(x,y)
{
  n = length(x)
  x1 = 0
  y1 = 0
  z1 = 0
  for (i in 1:n)
  {
    x1[i] = x[i]^2
    y1[i] = y[i]^2
    z1[i] = (x[i]/y[i])^2
  }
  sum_square_x = sum(x1)
  sum_square_y = sum(y1)
  sum_square_z = sum(z1)
  g = sum_square_x/sum_square_y
  h = sum_square_z
  cat("The value of g and h are", g, "and", h,
      "respectively", "\n")
}
```

# Example 1

```
R Console
> example1
function(x,y)
{
  n = length(x)
  x1 = 0
  y1 = 0
  z1 = 0
  for (i in 1:n)
  {
    x1[i] = x[i]^2
    y1[i] = y[i]^2
    z1[i] = (x[i]/y[i])^2
  }
  sum_square_x = sum(x1)
  sum_square_y = sum(y1)
  sum_square_z = sum(z1)
  g = sum_square_x/sum_square_y
  h = sum_square_z
  cat("The value of g and h are", g, "and", h,
      "respectively", "\n")
}
> |
```

# Example 1



The screenshot shows the R Console interface with a blue header bar containing the R logo and the text "R Console". Below the header, there is a scrollable text area displaying R code. The code defines a function named "example1" that performs several operations on vectors "x" and "y". It calculates the length of "x", initializes "x1", "y1", and "z1" to 0, and then iterates through each element of "x" to calculate the square of each element and store it in "x1", "y1", and "z1" respectively. It also calculates the sum of squares of "x", "y", and "z", and then divides the sum of squares of "x" by the sum of squares of "y" to get "g". Finally, it prints a message to the console stating the values of "g" and "h". The code ends with a closing brace for the function definition.

```
> example1 <- function(x,y)
+ {
+   n = length(x)
+   x1 = 0
+   y1 = 0
+   z1 = 0
+   for (i in 1:n)
+   {
+     x1[i] = x[i]^2
+     y1[i] = y[i]^2
+     z1[i] = (x[i]/y[i])^2
+   }
+   sum_square_x = sum(x1)
+   sum_square_y = sum(y1)
+   sum_square_z = sum(z1)
+   g = sum_square_x/sum_square_y
+   h = sum_square_z
+   cat("The value of g and h are", g, "and", h,
+       "respectively", "\n")
+ }
```

## Example 1

```
> x=c(10,20,30)  
> y=c(1,2,3)  
> example1(x,y)
```

The value of g and h are 100 and 300 respectively

```
> x=c(67,87,26,85,6,45)  
> y=c(54,64,22,94,20,88)  
> example1(x,y)
```

The value of g and h are 0.8996568 and 5.953203  
respectively

Just by changing the values of **x** and **y**, one can get required different outcomes.

# Example 1

```
R Console
> x=c(10,20,30)
> y=c(1,2,3)
> example1(x,y)
The value of g and h are 100 and 300 respectively
>
> x=c(67,87,26,85,6,45)
> y=c(54,64,22,94,20,88)
> example1(x,y)
The value of g and h are 0.8996568 and 5.953203 respectively
```

# Example 1 (Alternative approach)

Input variables : **x, y, n**

Output variables: **g, h**,  $g = \frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n y_i^2}$  and  $h = \sum_{i=1}^n \left( \frac{x_i}{y_i} \right)^2$

**g = sum(x^2)/sum(y^2)**

**h = sum(x/y)^2**

## Example 2

Suppose we want to compute

$$f(x, y) = \frac{\left(\frac{x + \ln y}{y}\right)^2}{5 + \left(\frac{x + \ln y}{y}\right)^3} \left[ \exp\left(\frac{x + \ln y}{y}\right) \right]^{\frac{2}{3}}$$

This can be written as

$$f(x, y) = \frac{(g(x, y))^2}{5 + (g(x, y))^3} \left[ \exp(g(x, y)) \right]^{\frac{2}{3}}$$

where  $g(x, y) = \frac{x + \ln y}{y}$

## Example 2

Input variables :  $x, y$

Output variables: :  $f$

We break this function in two components –

- Compute  $g(x, y)$  as a function and then
- compute  $f(x, y)$  by calling  $g(x, y)$ .

## Example 2

```
# Remove all data  
rm(list = ls())  
  
# Define input data vectors  
x  
y
```

CONTD...

## Example 2

CONTD...

```
# define g(x,y)
g = function(x,y)
# Start of function
{
  (x+log(y))/y
# End of function
}
```

$$g(x, y) = \frac{x + \ln y}{y}$$

+++++

```
# define f(x,y)
f = function(x,y)
{
  (((g(x,y))^2)/(5+(g(x,y))^3))*(exp(g(x,y)))^(2/3)
}
```

$$f(x, y) = \frac{(g(x, y))^2}{5 + (g(x, y))^3} \left[ \exp(g(x, y)) \right]^{\frac{2}{3}}$$

## Example 2: At a glance

```
# define g(x,y)
```

```
g = function(x,y)
{
  (x+log(y))/y
}
```

```
+++++
```

```
# define f(x,y)
```

```
f = function(x,y)
{
  (((g(x,y))^2)/(5+(g(x,y))^3))*(exp(g(x,y)))^(2/3)
}
# g(x,y) must have been defined earlier.
```

## Example 2

```
R Console

> # define g(x,y)
> g = function(x,y)
+ # Start of function
+ {
+   (x+log(y))/y
+ # End of function
+ }
>
> # define f(x,y)
> f = function(x,y)
+ {
+   (((g(x,y))^2) / (5+(g(x,y))^3)) * (exp(g(x,y)))^(2/3)
+ }
> |
```

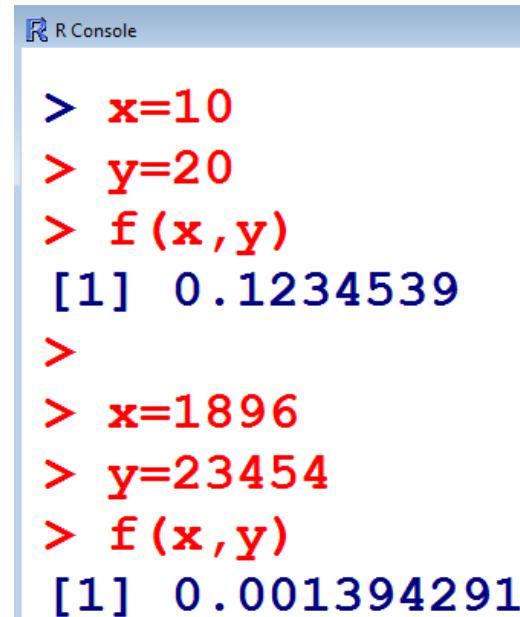
## Example 2

```
R Console

> g
function(x,y)
# Start of function
{
  (x+log(y))/y
# End of function
}
> f
function(x,y)
{
  (((g(x,y))^2) / (5+(g(x,y))^3)) * (exp(g(x,y)))^(2/3)
}
```

## Example 2

```
> x=10  
> y=20  
> f(x,y)  
[1] 0.1234539  
  
> x=1896  
> y=23454  
> f(x,y)  
[1] 0.001394291
```



The image shows a screenshot of an R console window titled "R Console". It contains two separate blocks of R code and their corresponding outputs. The first block starts with "x=10" and "y=20", followed by a call to "f(x,y)" which returns "[1] 0.1234539". The second block starts with "x=1896" and "y=23454", followed by a call to "f(x,y)" which returns "[1] 0.001394291". The text in the console is color-coded: blue for the prompt (">"), red for variable names ("x", "y", "f"), and black for the output values.

```
R Console  
> x=10  
> y=20  
> f(x,y)  
[1] 0.1234539  
>  
> x=1896  
> y=23454  
> f(x,y)  
[1] 0.001394291
```

There is no need to calculate the value of  $g(x,y)$ .

Just by changing the values of  $x$  and  $y$ , one can get different required outcomes.

## Example 3

Suppose we want to compute

$$f(x) = \begin{cases} \exp\left(\frac{x + \ln(1 + x^3)}{x^2}\right) & \text{if } x > 0 \\ 10 & \text{if } x = 0 \\ \frac{2 + x^3}{x} & \text{if } x < 0 \end{cases}$$

and plot with line over a values of x as a sequence starting from -1 to 5 and increasing it by 0.2.

## Example 3

Input variable : **x**

Output variable: **f**

```
# Remove all data  
rm(list = ls())
```

```
# Define input data  
x
```

CONTD...

## Example 3

CONTD...

```
f = function(x)
{
  if(x>0) {exp((x+log(1+x^3))/x^2)}
  else if(x==0) {10}
  else {(2+x^3)/x}
}
```

$$f(x) = \begin{cases} \exp\left(\frac{x + \ln(1 + x^3)}{x^2}\right) & \text{if } x > 0 \\ 10 & \text{if } x = 0 \\ \frac{2 + x^3}{x} & \text{if } x < 0 \end{cases}$$

CONTD...

## Example 3

CONTD...

```
h = function()
# Start of function
{
# Generation of data on x
x = seq(-1,5,by=0.2)
# Initialization of y to store values of f(x)
y = 0
```

CONTD...

## Example 3

CONTD...

```
# Generation of f(x) values corresponding to x
for(i in 1:length(x))
{
  y[i] = f(x[i])
}
# length(x) and length(y) must be same to plot
# y = f(x) with respect to x
plot(x, y, type = "l")
}
```

## Example 3: At a glance

```
f = function(x)
{
  if(x>0) {exp((x+log(1+x^3))/x^2)}
  else if(x==0) {10}
  else {(2+x^3)/x}
}

h = function()
{
x = seq(-1,5,by=0.2)
y = 0

for(i in 1:length(x))
{
  y[i] = f(x[i])
}
plot(x,y,type = "l")
}
```

## Example 3

```
R Console

> f = function(x)
+ {
+   if(x>0) {exp((x+log(1+x^3))/x^2)}
+   else if(x==0) {10}
+   else {(2+x^3)/x}
+ }
>
> h = function()
+ {
+   x = seq(-1,5,by=0.2)
+   y = 0
+   for(i in 1:length(x))
+   {
+     y[i] = f(x[i])
+   }
+   plot(x,y,type = "l")
+ }
> |
```

## Example 3

```
R Console

> f
function(x)
{
  if(x>0) {exp((x+log(1+x^3))/x^2)}
  else if(x==0) {10}
  else {(2+x^3)/x}
}
> h
function()
{
  x = seq(-1,5,by=0.2)
  y = 0
  for(i in 1:length(x))
  {
    y[i] = f(x[i])
  }
  plot(x,y,type = "l")
}
> |
```

## Example 3

```
> f(123)
[1] 1.009126

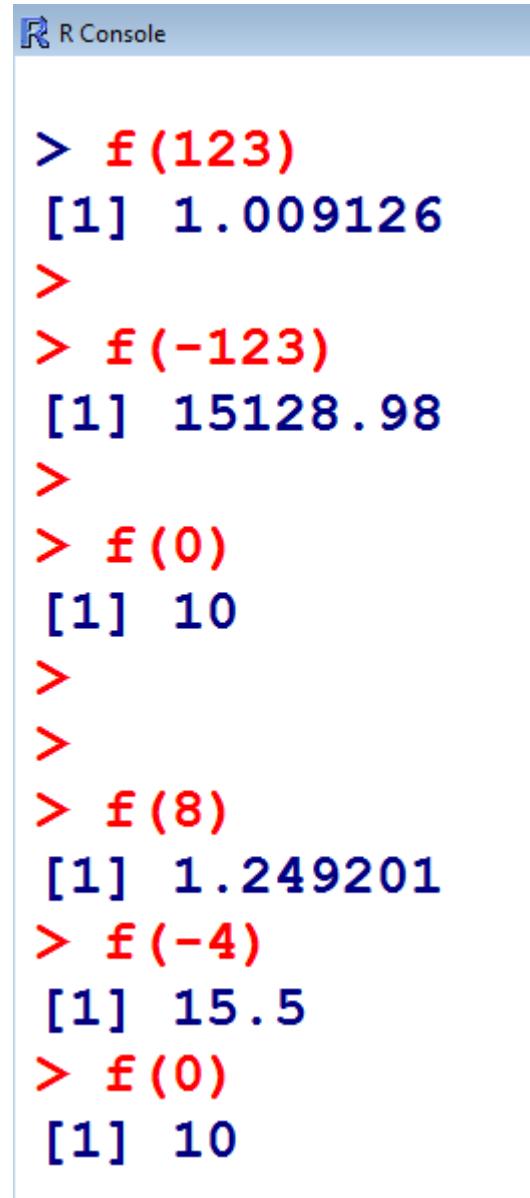
> f(-123)
[1] 15128.98

> f(0)
[1] 10

> f(8)
[1] 1.249201

> f(-4)
[1] 15.5

> f(0)
[1] 10
```



```
R Console

> f(123)
[1] 1.009126
>
> f(-123)
[1] 15128.98
>
> f(0)
[1] 10
>
>
> f(8)
[1] 1.249201
> f(-4)
[1] 15.5
> f(0)
[1] 10
```

## Example 3

> h( )

