R reserved words

Reserved words in R programming are a set of words that has special meaning and cannot be used as identifiers (variable names or function names).

eg: if, for, FALSE, NA_integer_, else, in, NULL, NA_real_, repeat, next, Inf, NA_complex_, while, break, NaN, NA_character_, function, TRUE, NA etc.

help(reserved)
or
?reserved
↓
In console

Variables and constants

Variables are used to store data whose value can be change according to own need. Unique name given to a variable is identifier.

Rules for writing identifiers

*) Identifiers can be a combination of letters, digits, period (.) and underscore.
*) It must be start with a letter or a period.
*) If it start with a period, it cannot be followed by a digit.
*) Reserved words in R cannot be used as identifie

Valid Identifiers

Eg: Total, Sum, fine, numbers, this_is_applicable etc.

Invalid identifiers

TRUE, .3m1, 5num etc.

Constants in R

Constants are entities whose value cannot be altered.

Two types of constants—

Numeric constants & character constants.

) Numeric constants — All numbers fall under this category. They can be a type of integer, double or complex.

It can be checked with type() function.

eg: type(5) gives output 'double'

Numeric constants followed by 'L' regarded as integer. eg: (type(5L) gives int)

and constants followed by 'i' regarded as complex. eg: (type(5i))

Numeric constants preceeded by 0x are interpreted as hexadecimal numbers.

constants — can be represented using either single quotes (' ') or double quotes (" ").

eg: type("5") → gives char.
type("abc") or type('abc')

Built-in constants

> LETTERS
"A", "B", ---, "Z"

> letters
"a", "b", ---, "z"

> pi
3.14

> month.name
"january", "february", --- "December"

> month.abb
"jan", "feb", --- "Dec"

R operators

Operators in R can be mainly classified into the following categories.

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

•) Assignment operators.

R Arithmetic operators

operators to perform arithmetic operation.

operator	Description	x=5 y=2
+	Addition	x+y=7
-	Subtraction	x-y=3
*	Multiplication	x*y=10
/	Division	x/y=2.5
^	Exponential	x^y=25
%%	Modulous	x%%y=1
%/%	Integer devision	x%/%y=2

Relational operators

operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	Equal to
!=	Not equal to

Logical operators

Operators	Description
!	logical NOT
&	Element wise logical AND
&&	logical AND
|	Element wise logical OR
||	logical OR

Assignment operators

Operators	Description
<-, <<-, =	x <- 5 or x <<- 5
->, ->>	

Special values

* NA (not available):
  Used to represent missing values.

eg: > v = c(1,2,3,4)
> v
[1] 1, 2, 3, 4
> length(v) = 6

[1] 1 2 3 4 NA NA

* Inf and -Inf
If a computation result in a no that is too big, R will return Inf for a +ve no and -Inf for -ve no

eg: 2^124

→ [1] 2.126765e+37

* NaN (Not a number):
Computation will produce a result with little sense

eg: >0/0
>NaN

* NULL:
Used as an argument in functions to mean that no value was assigned to the argument.

Vector

It is a basic data structure in R.
It contains elements of same type.

*) Vector created using c().
*) Vectors type can be checked with typeof() function
*) No of elements in vector can be checked with length().

Creating

```
> x <- c(1, 2, 3, 4, 5)
> typeof(x)
[1] double.
```

Creating vector using : operator

```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7
```

Creating vector using seq()

```
>seq(1, 3, by=0.2)
```

How to access vector elements

Elements of vector can be accessed using vector indexing

The vector used for indexing can be logical, integer or character vector.

Using integer vector as index

⇒ Vector index in R start from 1.

⇒ we can use vector of integers as index to access specific elements.

⇒ we can also used -ve integers to return all elements except that those specified.

⇒ But we cannot mix +ve & -ve integers.

```
> x
[1] 0 2 4 6 8 10

> x[3]
[1] 4

> x[c(2,4)]
[1] 2 6

> x[-1]
[1] 2 4 6 8 10        # prints all but 1st element.

> x[c(2.4, 3.54)]
[1] 2 4               # real nos are truncated to integers.
```

Using logical vector as index

When we use logical vector for indexing, the position where logical vector is TRUE is returned.

```
> x[c(TRUE, FALSE, FALSE, TRUE)]
[1] -3  3
```

Using character vector as index

This type of indexing is useful when dealing with named vectors. We can name each elements of a vector.

```
> x <- c("first" = 3, "second" = 0, "third" = 9)
> names(x)
[1] "first"  "second" "third"
> x["second"]
second
     0
```

Modify a vector

we can modify using assignment operator.

```
> x
[1] -3 -2 -1  0  1  2

> x[2] <- 0; x
[1] -3  0 -1  0  1  2

> x[x<0] <- 5; x
[1] 5 0 5 0 1 2

> x <- x[1:4]; x
[1] 5 0 5 0
```

Delete a vector

We can delete a vector by simply assigning NULL to it

```
> x
[1] -3 -2 -1  0  1  2
> x <- NULL
```

List

- List is a data structure having components of mixed data types.
  A vector having all elements of same type is called atomic vector but a vector having elements of different type is called list.
- List created using list() function.
- Structure can be examined with str() function.
- x <- list("a"=2.5, "b"=TRUE, "c"=1:3)

  a, b, & c are tags which make it easier to reference the components of list.
- Tags are optional.
  Same list without tag

  x <- list(2.5, TRUE, 1:3)

```
> x = list(1.5, "abc", 1:3)
> x
[[1]]
[1] 1.5

[[2]]
[1] "abc"
```

(double

[[3]]

[1] 1 2 3

```
> x = list("a" = 1.5, "b" = "abc", "c" = 1:3)
> x
$a
[1] 1.5
$b
[1] "abc"
$c
[1] 1 2 3

> typeOf(x)
[1] "list"
> str(x)
List of 3
 $ a: num 1.5
 $ b: chr "abc"
 $ c: int [1:3] 1 2 3
```

Conso

Access components of a list

List can be accessed in similar fashion to vectors. Integer, logical or character vectors can be used for indexing.

Accessing index using integer vector

" logical "

Refer video. ⟸ Character vector

etc

Data frames

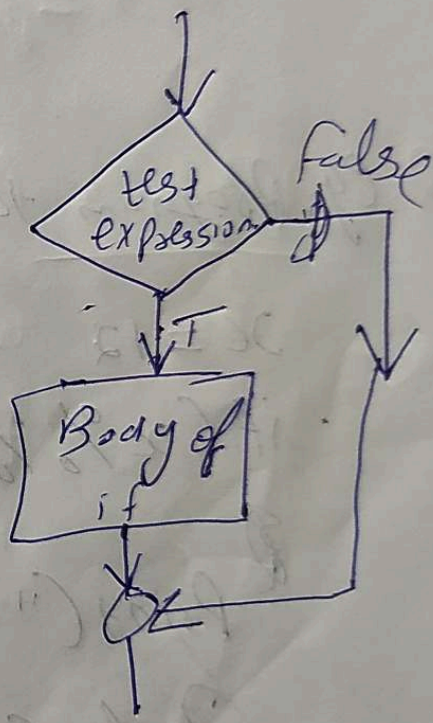
Single square bracket '[ ]' — Sub list

Double " [[ ]] — Contents :- (#)

Conditional statement in R

1. if statements

```
if (test expression) {
    Statement
}
```

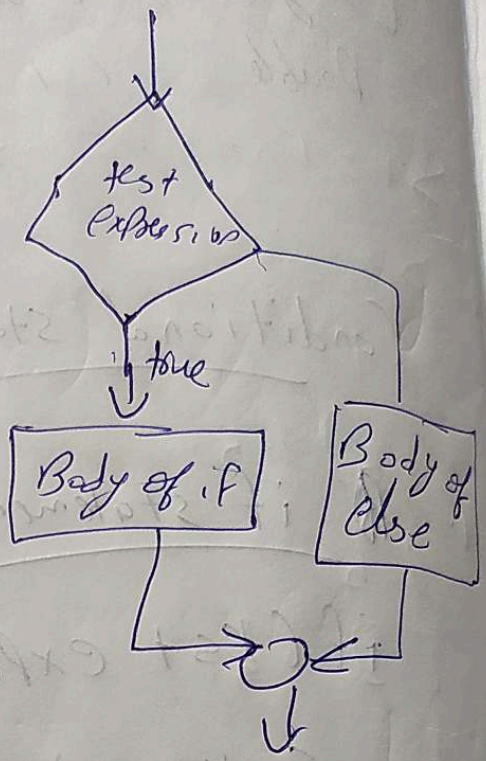


eg:

```
x<-10
if(x%%2==0){
Print("even")
```

2. if else statement

```
if (test expression) {
  statement 1
}else{
  statement 2
}
```



Eg: Program to check whether odd or even.

```
x = 12
if (x % 2 == 0) {
  Print("even")
}else{
  Print("odd")
}
```

3. nested if or else if ladder

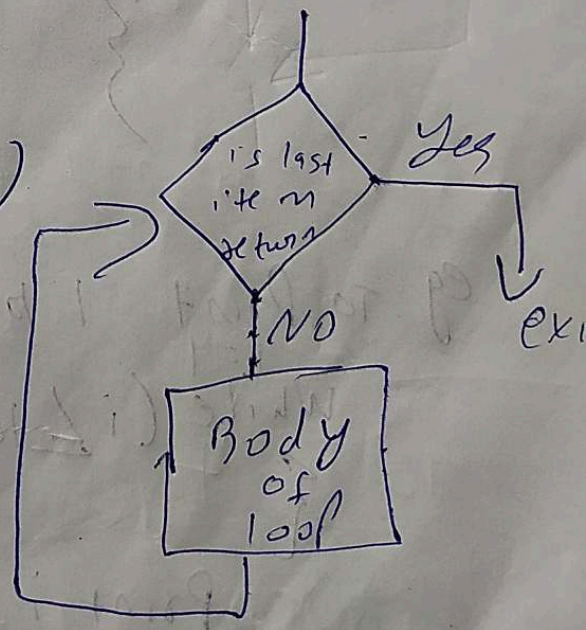
```
if(test expression1){
    Statement 1
}else if(test expression 2){
    Statement 2
} else if(test expression 3){
    Statement 3
} else{
    Statement 4
}
```

Loop

1. for loop

Syntax: for (Val in sequence)

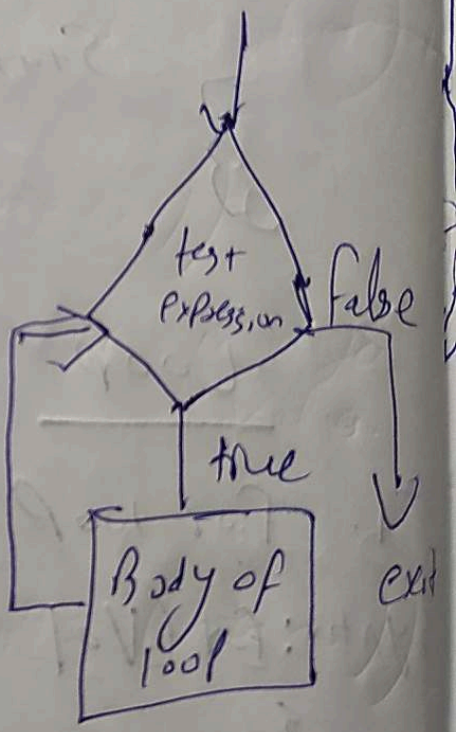
```
{
    Statement
}
```



```
x=c(1,5,3,9,8)
count=0
for(i in x)
{
    if(i%%2==0)
      count++
}
Print(count)
```

Q. While loop

Syntax:
```
while(test-exp)
{
  statement
}
```



eg: To Print 1 to 10

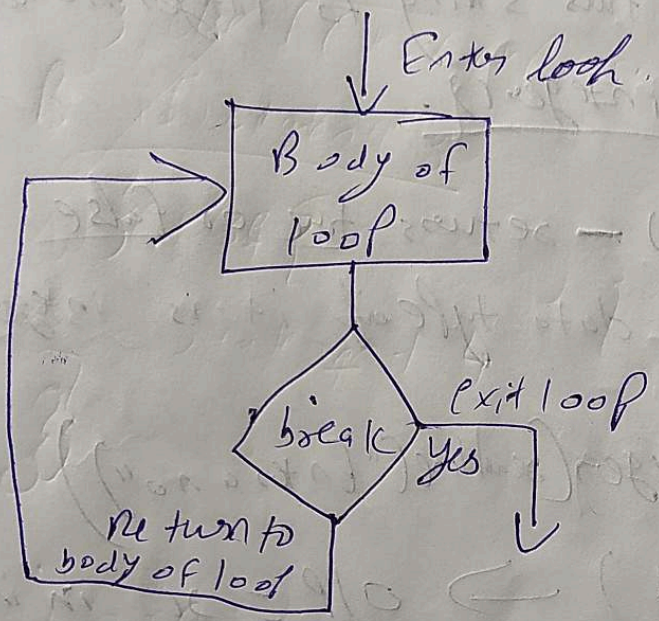
```
i=1
while(i<=10)
{
  Print i
  i=i+1
}
```

3. Repeat loop

Syntax: repeat {

Statements

}



Iterate over a block of code multiple number of times. There is no conditional check.

Put condition explicitely inside the body of the loop, also use break statement inside the loop.

eg: Print 1st 10 numbers using repeat loop

```
i=1
repeat {
    Print(i)
    if(i==10) i=i+1
    if(i==11)
        break
}
```

o/p

1 2 3 4 5
6 7 8 9 10

How to read from users

readline().

Scan()

readline() — all values are read as string.

To convert this string to integer number, we use as.integer().

is.integer() — returns true or false ie to check the data type if it is integer or not.

eg: x = as.integer(readline("Enter a no"))

print(x) → O/P = x in integer values.

or

x = as.integer(readline(prompt = "Enter no"))

Print(x).

To Print we use "Paste" argument.

Print(Paste = "my name is", x)

Q) Check whether Prime or not

```
num = as.integer(readline(prompt="Enter a no:"))
count=0
for(i in 1:num)
{
  if(num%%i==0)
    count=count+1
}
if(count==2)
  Print(Paste(num," is a Prime")) else
  Print(Paste(num," is not a Prime"))
```

To find factorial

```
num = as.integer(readline(Prompt="Enter a no:"))
fact=1
for(i in 1:num)
{
  fact=fact*i
}
Print(Paste("Factorial of",num,"is",fact))
```

5

Or

```
num = as.integer(readline(prompt="Enter a no:"))
if(num<0){
  print(paste("Not possible"))
}else if(num=0){
  fact=1
  Print(Paste("factorial of",num,"is",fact)
}else{
  fact=1
  for(i in 1:num)
  {
    fact=fact*i
  }
  Print(Paste("factorial of",num,"is",
              fact))
```

Scan() —

Can continuously read only integers.

```
x = scan()
Print(x)
```

To read other than int we have to use 'what'

eg: x = scan(what = double())
scan(what = "") // String
scan(what = character())

Module 4

R Functions

~~function name~~ It is a block of code only runs when it is called. It passes data known as parameters or arguements and it can return data or object.

Function definition

R function is created using 'function' keyword

Syntax: functionname <- function(arg1, arg2, arg3)
{
body of function
return(object)
}

Different Parts of a function

1. Function name – This is the actual name of the function.
2. Arguements – It is a place holder ~~called~~ when a function is invoked Pass a value to the arguement.
   Arguements are optional which means a function may contain no arguements.
3. Function body – The function body contains a collection of statements that defines what the function does.
4. Return value – ~~Means~~ is the last expression in the function body to be evaluated.

Example 1 – single i/p & single o/p

```
Fun ← function (x){
    z = x + 2
    return (z)
}

Fun (6)
```

o/p

8

Example 2 – Two i/p & 1 o/p

```
fun <- function(x,y){
    z=x+y
    return(z)
}

fun(10,20)
```

o/p

30

Example 3 – Two o/p

```
fun <- function(x,y){
    z1=x+y
    z2=x-y
    return(c(z1,z2))
}

fun(3,2)
```

o/p

5, 1

Q) Create a function to Print the squares of a number in sequence.

Prog

```
for(i in 6)
{
    for(i in 1:6)
    fun <- function(){
    for(i in 1:6)
        2=i^2
        Print(2)
}
fun()
```

Q) Create a function to find the sum, difference & product of two numbers.

Ans

```
fun <- function(x,y){
    s = x+y
    d = x-y
} mul = x*y
```

```
val 1 = as.integer(readline(prompt="Enter 1st no"))
val 2 = as.integer(readline(prompt="Enter 2nd no"))
fun(val 1, val 2)
```

R. Packages

is a related set of f()s, help files & data files that have been bundled together.

Packages in R are similar to classes in Java

~~It has~~

The Package has to be installed in local library.

To get the list of packages loaded by default, you can use the getoption command.

eg: getoption("defaultPackages")

R-string

Any value written within a pair of single quote or double quotes in R is treated as string.

Functions in string ⇒ paste(), format(), nchar() toupper(), tolower etc.

```
a = c('a', 'b', 'c')
b = c('d', 'e', 'f')
> print(paste(a, b))
[1] "a d" "b e" "c f"
> print(paste(a, b, sep = "x"))
[1] "a xd" "b xe" "c xf"
> print(paste(a, b, sep = "x", collapse = "@"))
[1] "a xd@b xe@c xf"
```

```
a = c(a, b, c, d)
b = c(e, f, g)
> paste(a, b)
```

⇒ "ae" "bf" "cg" "de"

format() function:

~~Format() is used to~~

Numbers & strings can be formatted to a specific style using format() function.

```
format(x, digits, nsmall, scientific, widths,
justify = c("left", "right", "centre", "none"))
```

x = vector i/p

digits - total no of digits displayed.

nsmall - Minimum no of digits to the right of decimal point.

Width

justify

eg:

```
> result <- format(23.1234, digits=4)
> print(result)
[1] "23.12"
```

```
> result <- format(23.47, nsmall=5)
> print(result)
[1] "23.47000"
```

```
> result <- format(6)
> print(result)
[1] "6"
```

Numbers are padded with blank in beginning for width

```
> result <- format(13.7, width=6)
> print(result)
[1] "  13.7"
```

```
> result <- format("Hello", justify = "c")
  > print(result)
[1] "  Hello  "
```

---

nchar()

Counts no of characters including spaces in a string.

Syntax

```
nchar(x)
```

eg:

```
> result <- nchar("Hello world")
> result
[1] 11
```

---

toupper() - change to upper case

Syntax: toupper(x)

tolower() - change to lower

Syntax: tolower()

eg: >result <- toupper("arun")
>print(result)
[1] "ARUN"

Substring()

To extract parts of a string.

Syntax

substring(x, first, last)

x = vector i/p

first - 1st position to extract

last - position of last position to extract.

eg: >result <- substring("Extract", 5, 7)
>print(result)
[1] "act"

If

String handling fn's

Paste()
format()
toupper()
tolower()
substring()
nchar()

Module 5

Graphs and Charts

Barplot

Created using barplot().

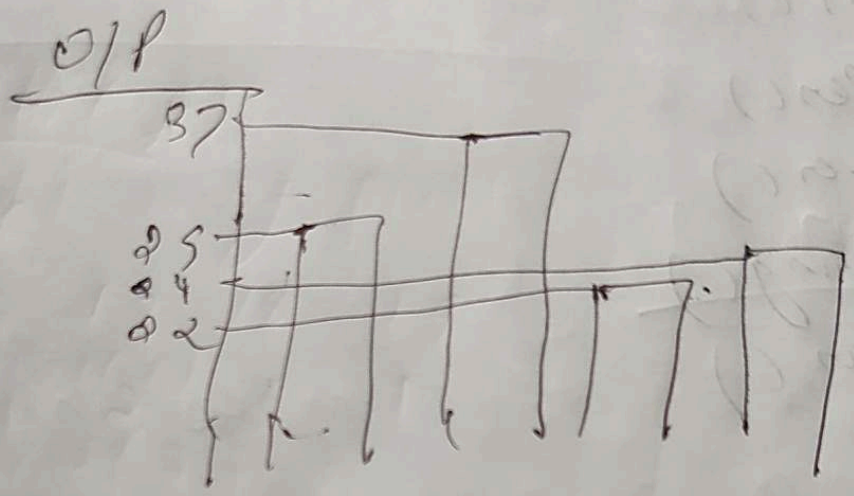
Inputs ⇒ Vector or matrix.

If we supply vector, the plot will have bars with their heights equal to elements in the vector

eg: tmp = c(25, 37, 22, 24)

barplot (tmp)

O/P



tmp = c(25, 37, 22, 24)

barplot(tmp)

main = "Maximum Temperatures"

xlab = ~~"Degree Celsius"~~, "Day"

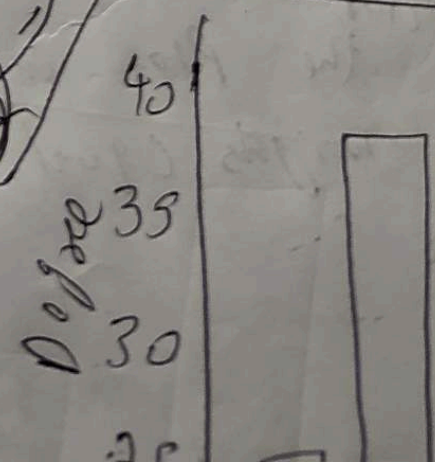
ylab = ~~"Day"~~ "Degree"

names.arg = c("sun", "mon", "Tue", "wed")

col = "darkred"

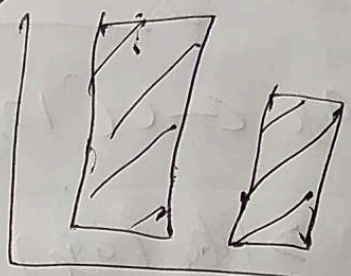
horiz = ~~TRUE~~

O/P ⇒



To shade we use density

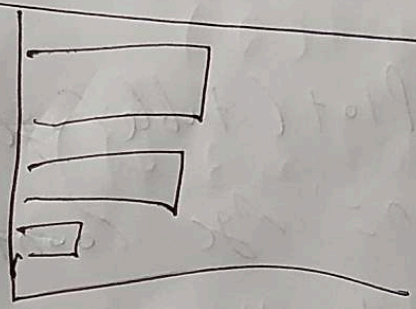
eg: density = 10 O/P ⇒



border = "red" ⇒ Changes border to red colour.

horiz = TRUE

⇒



col = "green"

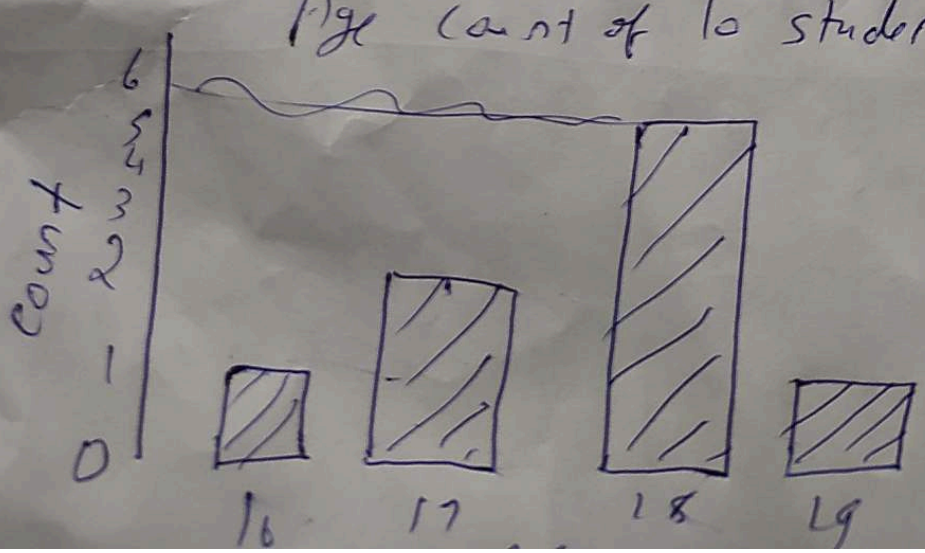
⇒ Changes colour to green

X lab = X axis name

Y lab = Y axis name

Plotting categorical data

```
age <- c(17, 18, 18, 17, 18, 19, 18, 16, 18, 18)
>table(age)
age
16 17 18 19
 1  2  6  1
barplot(table(age),
 main="Age count of 10 students",
 xlab="Age",
 ylab="count",
 border="red",
 col="blue",
 density=10
 )
```



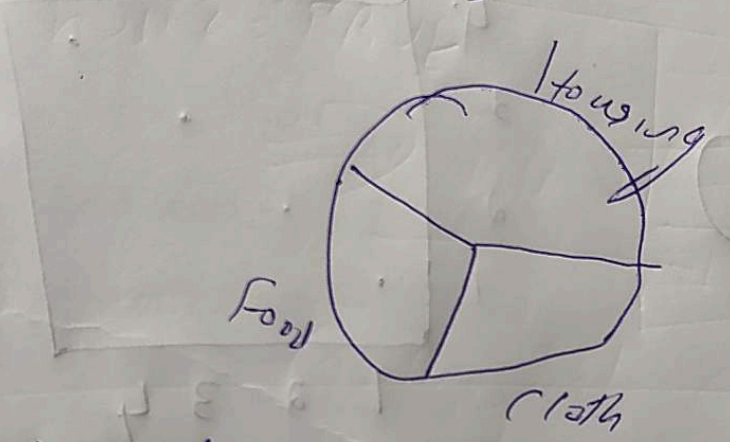
Pie chart

We can create Pie chart using Pie()

> expenditure

Housing	Food	clothing
600	300	150

Pie(expenditure)



labels – give name to slices

main – to add title

col – to define colors

border – to color borders.

Pie(expenditure,

labels = as.character(expenditure),

main = "monthly Expenditure",

col = c("red", "orange", "yellow",

border = "brown",

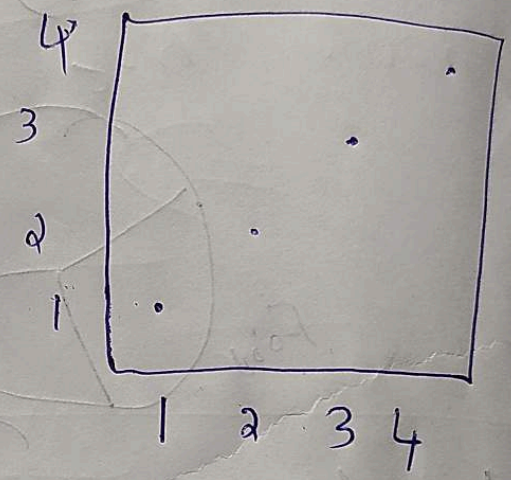
B Histogram

Created using hist()

Scatter Plot

Created using plot()

eg: plot(1)
plot(2)
plot(1,2,3,4))



x = 1:10
y = 21:30
plot(x,y);
plot(x,y)
plot(x,y, main="scatterplot",
xlab="x values", ylab="y values")

Box Plot

Using boxplot()

```
boxplot(airquality$Ozone,
main="mean ozone in Parts per billion
    at Roosevelt island",
xlab="Parts Per Billion",
ylab="Ozone",
col="orange",
borders="brown",
horizontal=TRUE,
notch=TRUE
)
```