Basics of Calculations

Basics, Built in Functions, Assignments, Variables, Data Types, Functions & Matrices

Kanahaiya Kumar [Data Mining Central]

Basics and R as a Calculator

> Is the prompt sign in R

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

> x <- 10 assigns the value 10 to x

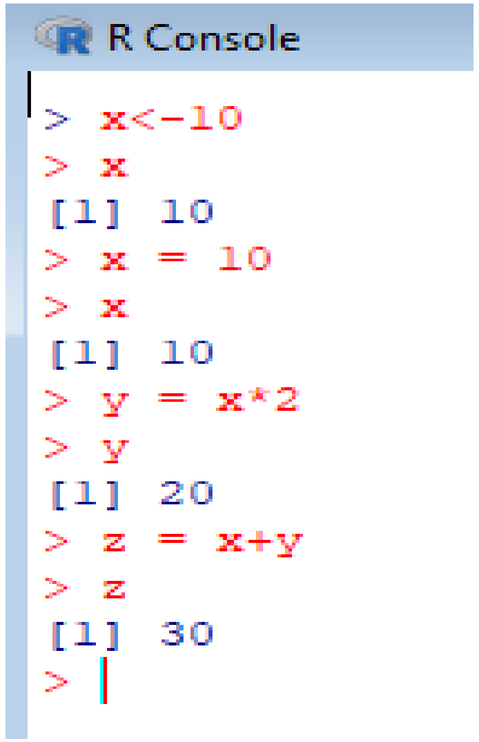
> x = 10 assigns the value 10 to x

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

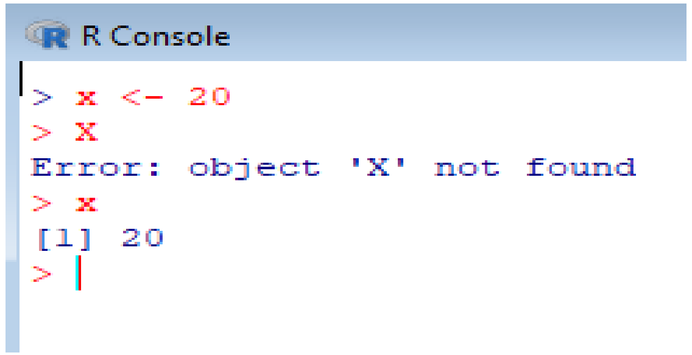


```
> x<-10
> x
[1] 10
> x = 10
> x
[1] 10
> y = x*2
> y
[1] 20
> z = x+y
> z
[1] 30
> 
```

Basics and R as a Calculator

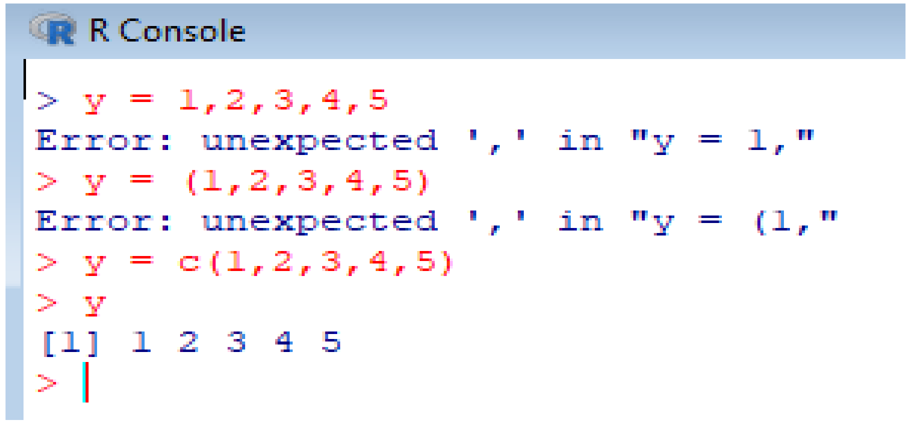
Capital and small letters are different

> X <- 20 and > x <- 20 are different


R Console

```
> x <- 20
> X
Error: object 'X' not found
> x
[1] 20
>
```

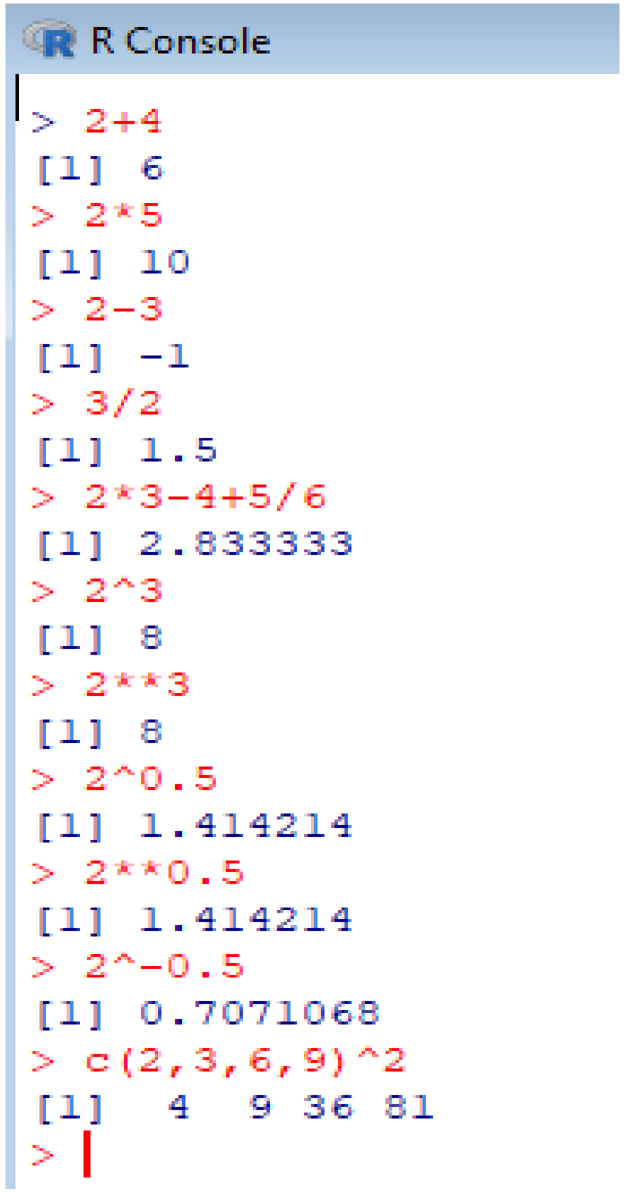
The command c(1,2,3,4,5) combines the numbers 1,2,3,4 and 5 to a vector


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
>
```

Basics and R as a Calculator

```
> 2+4 #command
[1] 6 #output
> 2*5 #command
[1] 10 #output
> 2-3 #command
[1] -1 #output
> 3/2 #command
[1] 1.5 #output
> 2*3-4+5/6 #command
[1] 2.833333 #output
> 2^3 #command
[1] 8 #output
> 2**3 #command
[1] 8 #output
> 2^0.5 #command
[1] 1.414214 #output
> 2**0.5 #command
[1] 1.414214 #output
> 2^-0.5 #command
[1] 0.7071068 #output
> c(2,3,6,9)^2 #command, Vector
[1]  4  9 36 81 #output
```

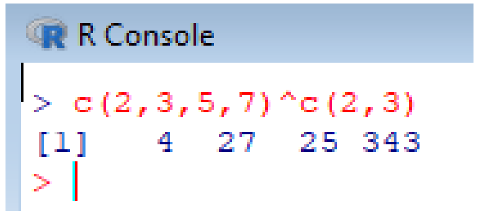


```
> 2+4
[1] 6
> 2*5
[1] 10
> 2-3
[1] -1
> 3/2
[1] 1.5
> 2*3-4+5/6
[1] 2.833333
> 2^3
[1] 8
> 2**3
[1] 8
> 2^0.5
[1] 1.414214
> 2**0.5
[1] 1.414214
> 2^-0.5
[1] 0.7071068
> c(2,3,6,9)^2
[1]  4  9 36 81
> 
```

$$\boxed{2^2,\ 3^2,\ 5^2,\ 7^2}$$

Multiplication & Division x * y , x/y

> c(2,3,5,7)^c(2,3) # !!ATTENTION! Observe the operation
[1] 4 27 25 343 # output

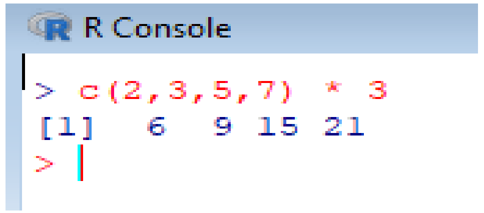


```
> c(2,3,5,7)^c(2,3)
[1]   4  27  25 343
> 
```

$$2^2,\ 3^3,\ 5^2,\ 7^3$$

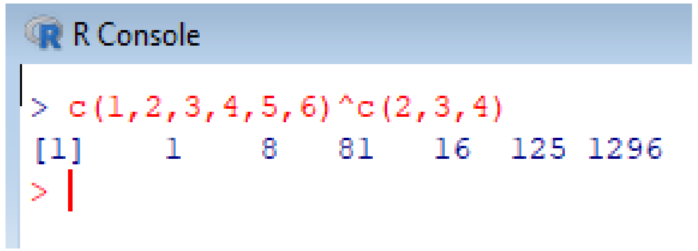
> c(2,3,5,7) * 3
[1] 6 9 15 21

$$2\times3,\ \ 3\times3,\ \ 5\times3,\ \ 7\times3$$



```
> c(2,3,5,7) * 3
[1]  6  9 15 21
> 
```

> 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



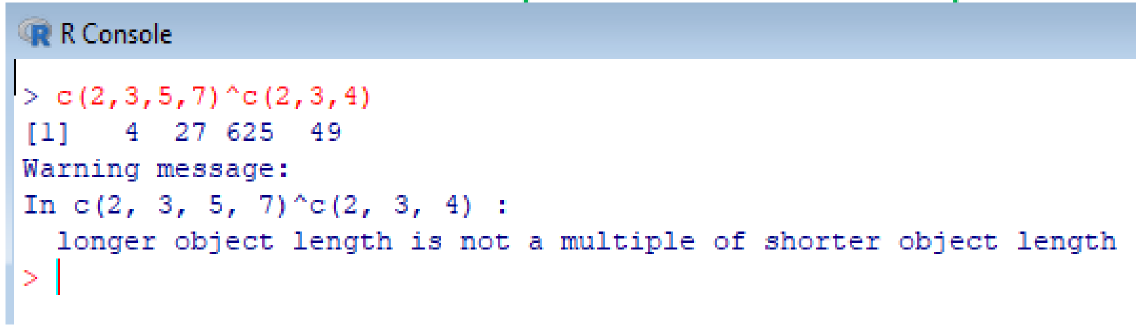
```
> c(1,2,3,4,5,6)^c(2,3,4)
[1]    1    8   81   16  125 1296
> 
```

$$1^2,\ 2^3,\ 3^4,\ 4^2,\ 5^3,\ 6^4$$

Multiplication & Division x * y , x/y

```
> c(2,3,5,7)^c(2,3,4) #error message
[1] 4 27 625 49 # output
```

$$\boxed{2^2,\ 3^3,\ 5^4,\ 7^2}$$



```
> 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
> 
```

```
> c(2,3,5,7) * 3
[1] 6 9 15 21
```

$$\boxed{2\times3,\ 3\times3,\ 5\times3,\ 7\times3}$$

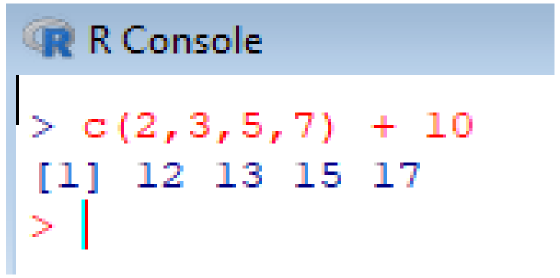
```
> c(2,3,5,7) * c(-2,-3,-5,8)
[1] -4 -9 -25 56
```

$$\boxed{2\times(-2),\ 3\times(-3),\ 5\times(-5),\ 7\times8}$$

Addition & Subtraction x+y, x-y

```
> c(2,3,5,7) + 10
[1] 12 13 15 17
```

$$\boxed{2+10,\ \ 3+10,\ \ 5+10,\ \ 7+10}$$

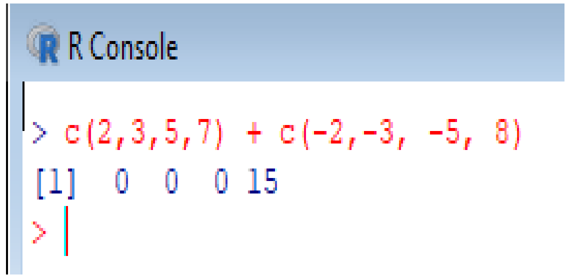


R Console

```
> c(2,3,5,7) + 10
[1] 12 13 15 17
> 
```

```
> c(2,3,5,7) + c(-2,-3, -5, 8)
[1] 0 0 0 15
```

$$\boxed{2+(-2),\ \ 3+(-3),\ \ 5+(-5),\ \ 7+8}$$



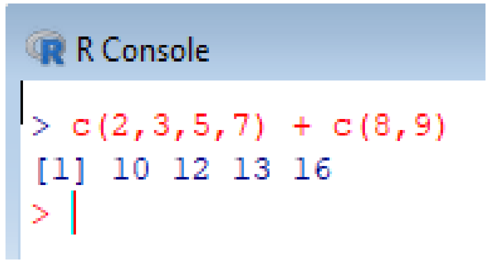
R Console

```
> c(2,3,5,7) + c(-2,-3, -5, 8)
[1]  0  0  0 15
> 
```

Addition & Subtraction x+y, x-y

```
> c(2,3,5,7) + c(8,9) # !!! ATTENTION!
[1] 10 12 13 16
```

$$\boxed{2+8,\ \ 3+9,\ \ 5+2,\ \ 7+9}$$

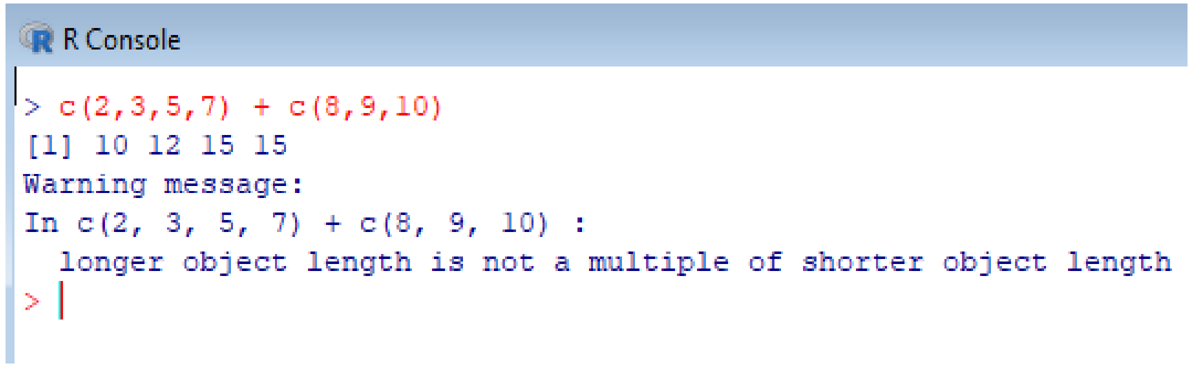


R Console

```
> c(2,3,5,7) + c(8,9)
[1] 10 12 13 16
> 
```

```
> c(2,3,5,7) + c(8,9,10) # error message
[1] 10 12 15 15
```

$$\boxed{2+8,\ \ 3+9,\ \ 5+10,\ \ 7+8}$$



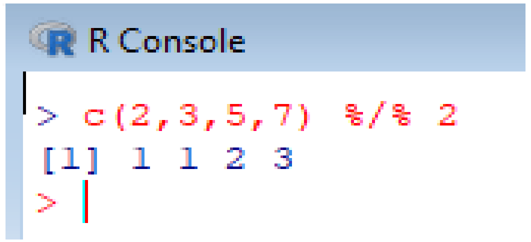
R Console

```
> c(2,3,5,7) + c(8,9,10)
[1] 10 12 15 15
Warning message:
In c(2, 3, 5, 7) + c(8, 9, 10) :
  longer object length is not a multiple of shorter object length
> 
```

Integer Division %/%

Integer Division: Division in which the fractional part (remainder) is discarded

```
> c(2,3,5,7) %/% 2
[1] 1 1 2 3
```

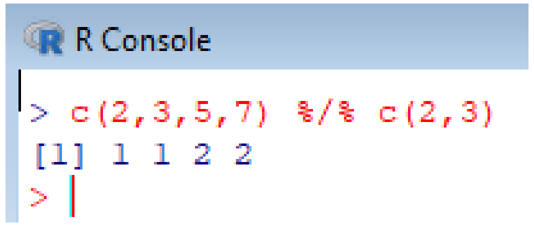


R Console

```
> c(2,3,5,7) %/% 2
[1] 1 1 2 3
>
```

2%/%2, 3%/%2, 5%/%2, 7%/%2

```
> c(2,3,5,7) %/% c(2,3)
[1] 1 1 2 2
```



R Console

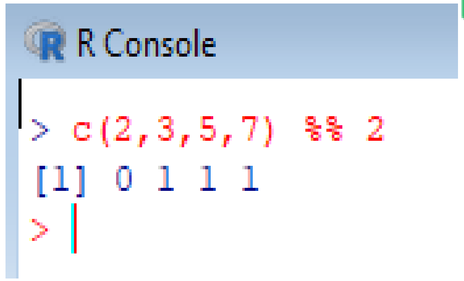
```
> c(2,3,5,7) %/% c(2,3)
[1] 1 1 2 2
>
```

2%/%2, 3%/%3, 5%/%2, 7%/%3

Modulo Division (x mod y) %%

x mod y : modulo operation finds the remainder after division of one number by another

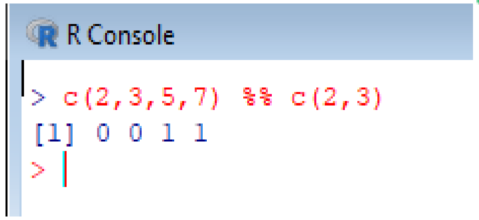
```
> c(2,3,5,7) %% 2
[1] 0 1 1 1
```


R Console

```
> c(2,3,5,7) %% 2
[1] 0 1 1 1
>
```

2%%2, 3%%2, 5%%2, 7%%2

```
> c(2,3,5,7) %% c(2,3)
[1] 0 0 1 1
```

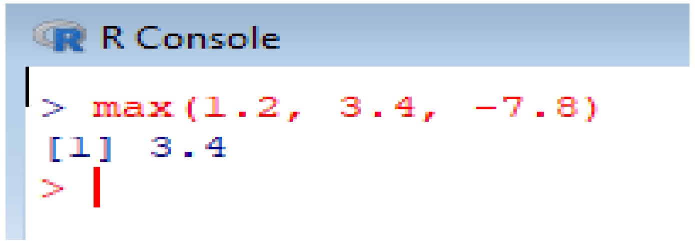

R Console

```
> c(2,3,5,7) %% c(2,3)
[1] 0 0 1 1
>
```

2%%2, 3%%3, 5%%2, 7%%3

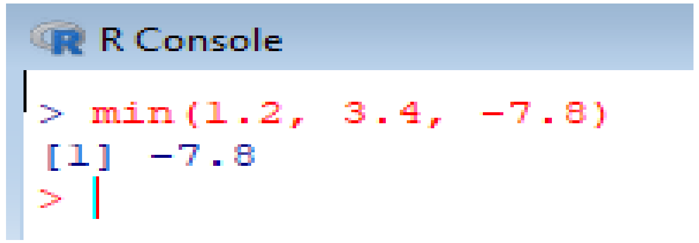
Maximum and Minimum: max and min

```
> max(1.2, 3.4, -7.8)
[1] 3.4
```



```
> max(1.2, 3.4, -7.8)
[1] 3.4
>
```

```
> min(1.2, 3.4, -7.8)
[1] -7.8
```



```
> min(1.2, 3.4, -7.8)
[1] -7.8
>
```

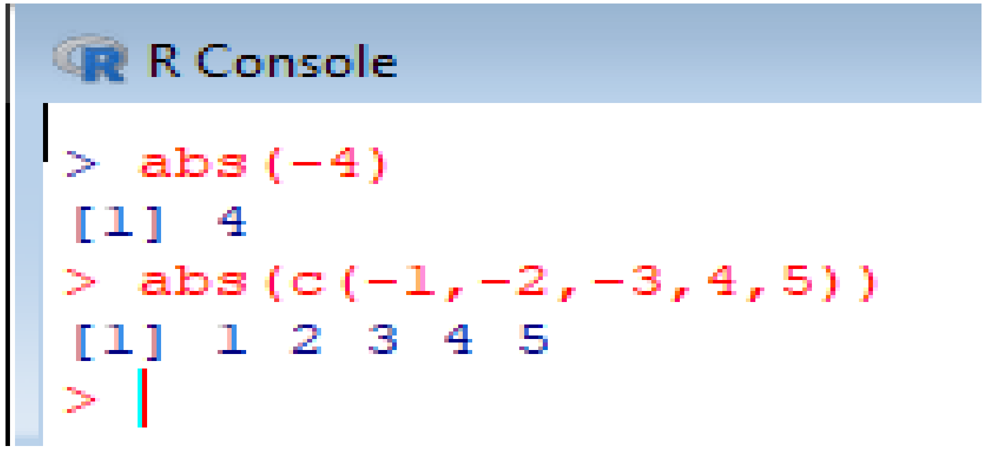
Some useful built in functions

abs()	Absolute value
sqrt()	Square root
round(), floor(), ceiling()	Rounding, up and down
sum(), prod()	Sum and product
log(), log10(), log2()	Logarithms
exp()	Exponential function
sin(), cos(), tan(), asin(), acos(), atan()	Trigonometric functions
sinh(), cosh(), tanh(), asinh(), acosh(), atanh()	Hyperbolic functions

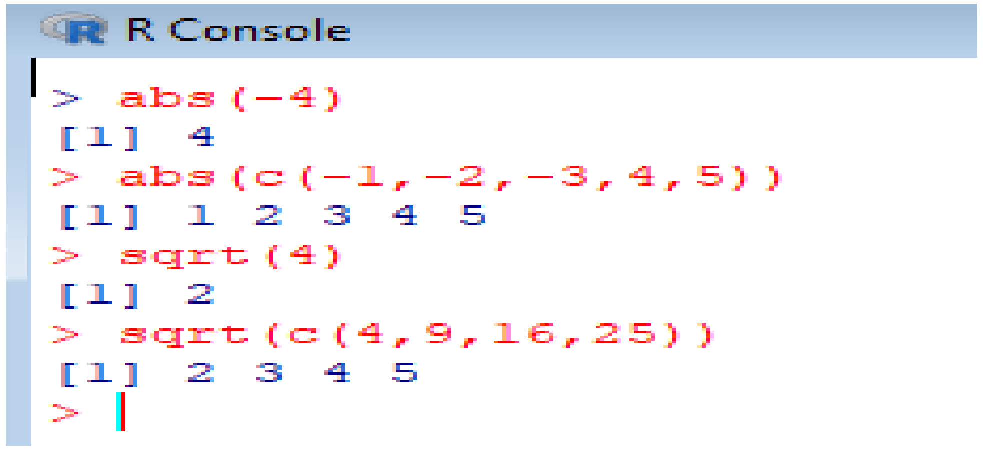
Some useful built in functions

Example:

```
> abs(-4)
[1] 4
> abs(c(-1,-2,-3,4,5))
[1] 1 2 3 4 5
> sqrt(4)
[1] 2
> sqrt(c(4,9,16,25))
[1] 2 3 4 5
```



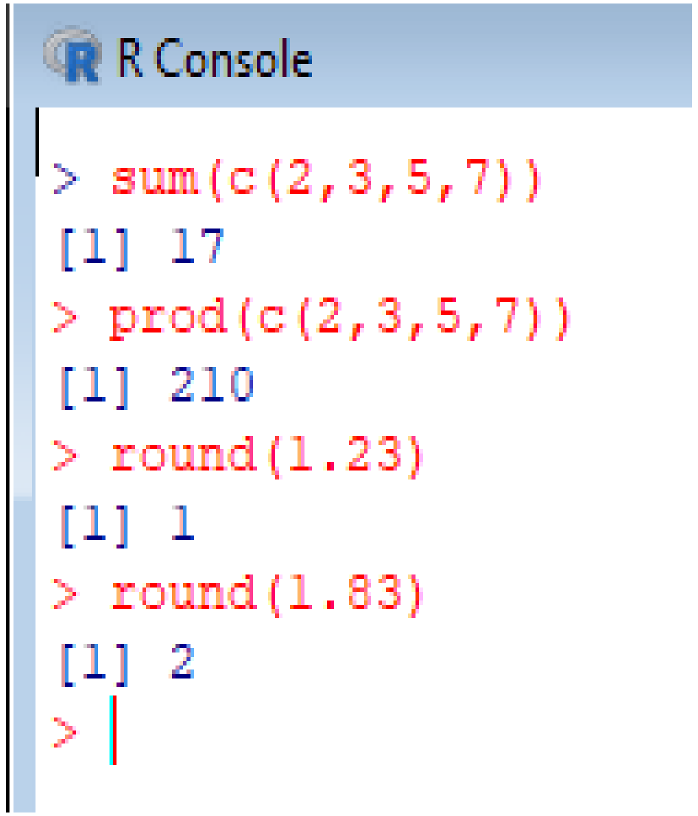
```
> abs(-4)
[1] 4
> abs(c(-1,-2,-3,4,5))
[1] 1 2 3 4 5
> 
```



```
> abs(-4)
[1] 4
> abs(c(-1,-2,-3,4,5))
[1] 1 2 3 4 5
> sqrt(4)
[1] 2
> sqrt(c(4,9,16,25))
[1] 2 3 4 5
> 
```

Some useful in functions

```
> sum(c(2,3,5,7))
[1] 17
> prod(c(2,3,5,7))
[1] 210
> round(1.23)
[1] 1
> round(1.83)
[1] 2
```



```
> sum(c(2,3,5,7))
[1] 17
> prod(c(2,3,5,7))
[1] 210
> round(1.23)
[1] 1
> round(1.83)
[1] 2
> 
```

Assignments

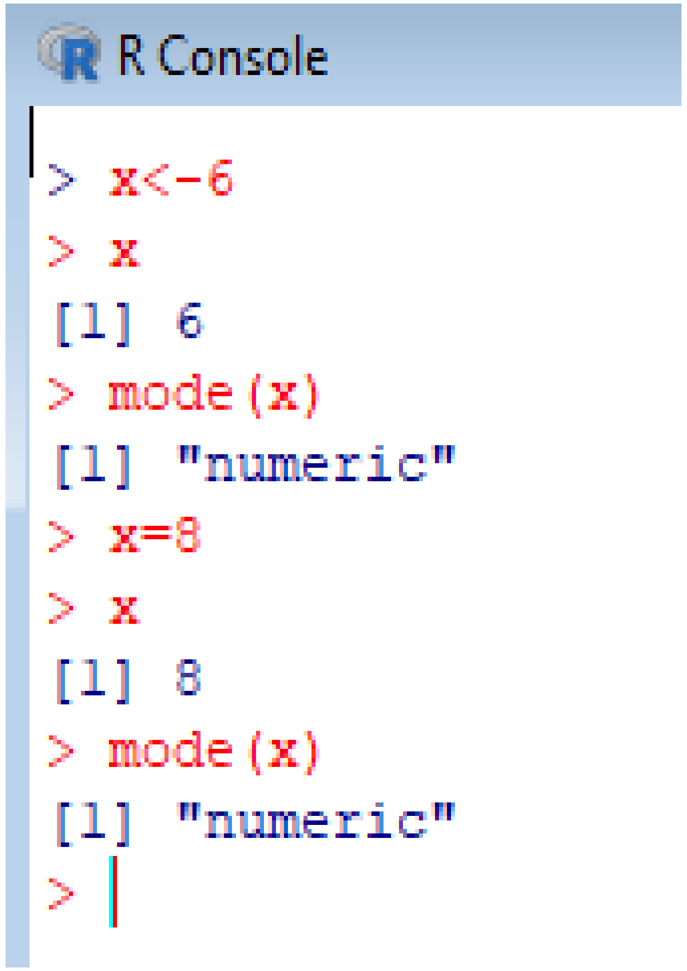
Assignments can be made in two ways:

```
> x<-6
> x
[1] 6

> mode(x)
[1] "numeric"

> x=8
> x
[1] 8

> mode(x)
[1] "numeric"
```



```
> x<-6
> x
[1] 6
> mode(x)
[1] "numeric"
> x=8
> x
[1] 8
> mode(x)
[1] "numeric"
>
```

Variables

Varibale Names – Rules

- Allowed characters are Alphanumeric, ‘_’ and ‘.’
- Always start with alphabets
- No special characters like !,@,#,$,….

Examples:

Correct naming: > b2 =7

> Manoj_GDPL = “Scientist”

> Manoj.GDPL = “Scientist”

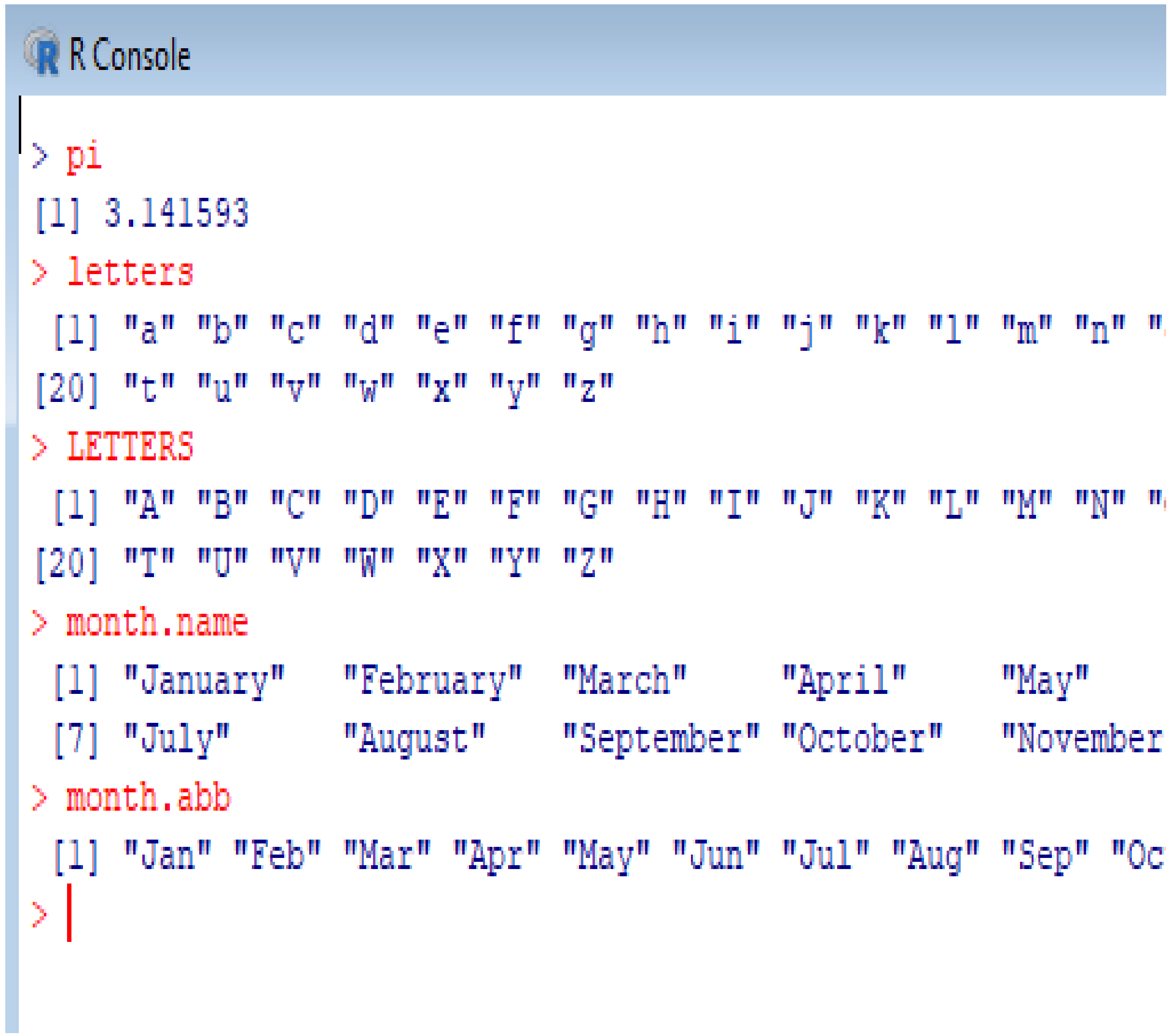
Wrong naming :

> 2b = 7

Error: unexpected input in “2b "

Predefined constants

Constant	Symbol in R
Pi	pi
letters	a,b,c,…..x,y,z
LETTERS	A,B,C,…X,Y,Z
Months in a year	month.name, month.abb



R Console

```
> pi
[1] 3.141593
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "
[20] "t" "u" "v" "w" "x" "y" "z"
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "
[20] "T" "U" "V" "W" "X" "Y" "Z"
> month.name
 [1] "January"   "February"  "March"     "April"     "May"
 [7] "July"      "August"    "September" "October"   "November
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oc
> 
```

Data Types

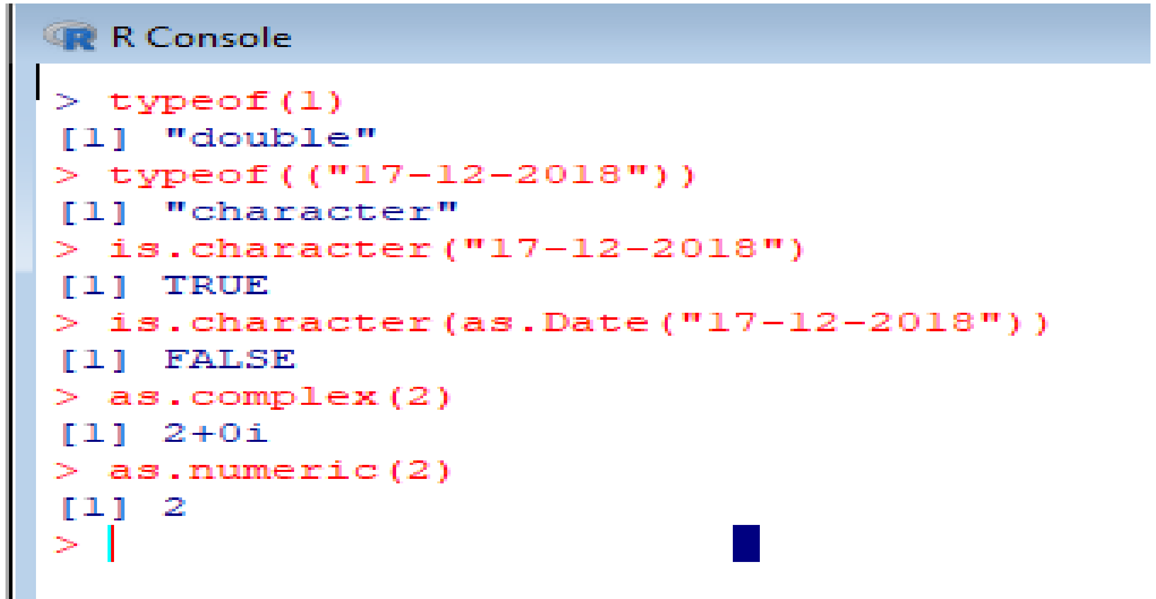
Basic Data Types	Values
Logical	TRUE and FALSE
Integer	Set of all Integers
Numeric	Set of all real numbers
Complex	Set of complex numbers
Character	“a”,”b”,”c”,….,”x”,”y”,”z”,”@”,”#”,”$”, “”,”*”, “1”,”2”,… etc..

Data Types

TASK	ACTION	SYNTAX/EXAMPLE
Find data type of object	use command "typeof()"	Syntax: typeof(object)
Verify if object is of a certain datatype	certain datatype use prefix "is." before datatype as command.	Syntax: is.data_type(object) Example : is.integer()
Coerce or convert data type of object to another	use prefix "as." before datatype as command.	Syntax: as.data_type(object) Example :as.logical()

Note : Not all coercions are possible and if attempted will return "NA" as output

Example



```
> typeof(1)
[1] "double"
> typeof(("17-12-2018"))
[1] "character"
> is.character("17-12-2018")
[1] TRUE
> is.character(as.Date("17-12-2018"))
[1] FALSE
> as.complex(2)
[1] 2+0i
> as.numeric(2)
[1] 2
>
```

Basic Objects

Object	**Values**
Vector	Ordered collection of same data types
List	Ordered collection of objects
Data frame	Generic tabular object

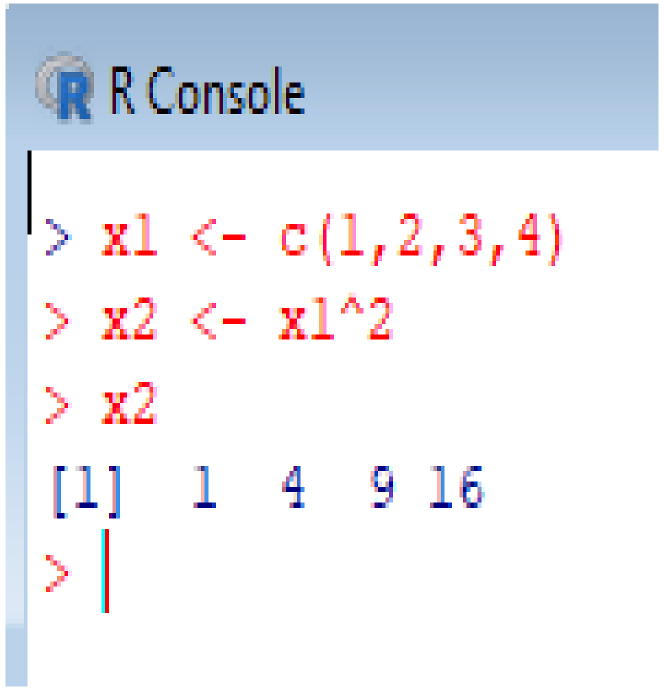
Assignments

An assignment can also be used to save values in variables:

```
> x1 <- c(1,2,3,4)

> x2 <- x1^2

> x2
[1] 1 4 9 16
```



```
> x1 <- c(1,2,3,4)
> x2 <- x1^2
> x2
[1]  1  4  9 16
> 
```

ATTENTION: R is case sensitive (X is not the same as x)

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

Syntax

```
Name <- function(Argument1, Argument2, ...)
{
expression
}
```

where expression is a single command or a group of commands

Functions

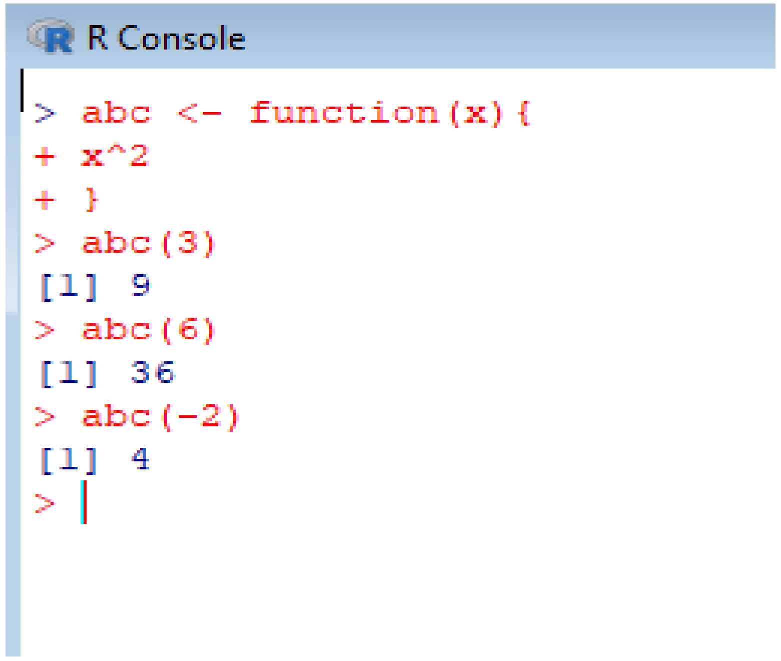
Function arguments with description and default values

- 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 <– is furthermore used for defining functions:

```
> abc <- function(x){
+ ^2
+ }
> abc(3)
[1] 9
> abc(6)
[1] 36
> abc(-2)
[1] 4
```



```
> abc <- function(x){
+ x^2
+ }
> abc(3)
[1] 9
> abc(6)
[1] 36
> abc(-2)
[1] 4
>
```

Functions

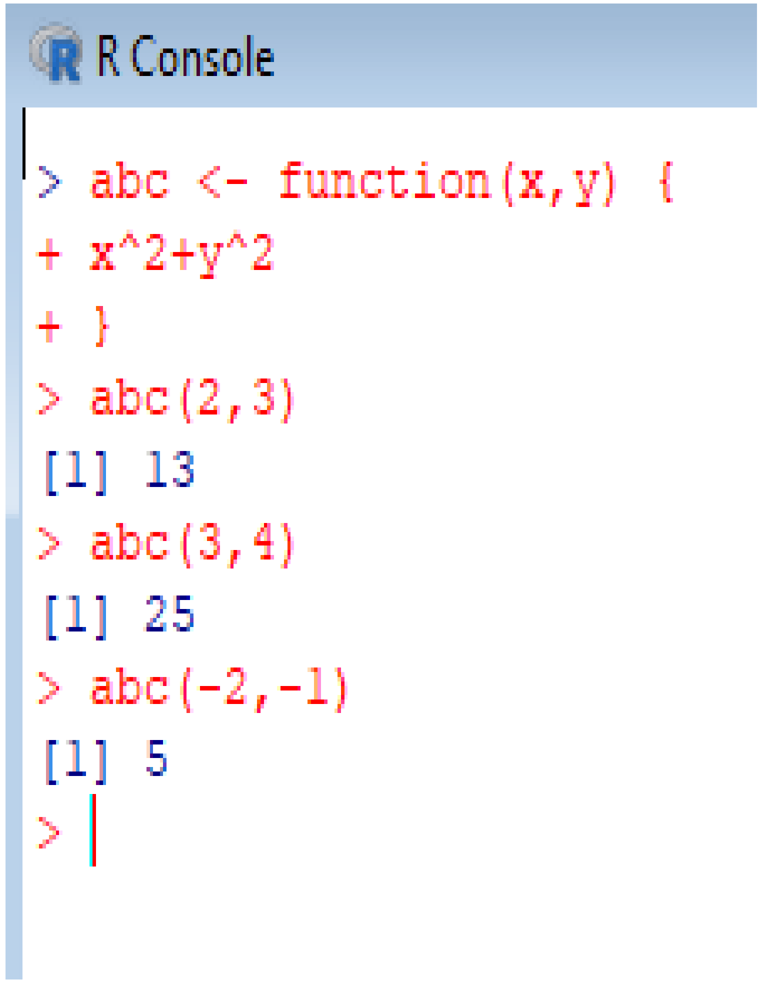
Functions (Two Variables)

```
> abc <- function(x,y){
x^2+y^2
}

> abc(2,3)
[1] 13

> abc(3,4)
[1] 25

> abc(-2,-1)
[1] 5
```



```
> abc <- function(x,y) {
+ x^2+y^2
+ }
> abc(2,3)
[1] 13
> abc(3,4)
[1] 25
> abc(-2,-1)
[1] 5
>
```

Functions

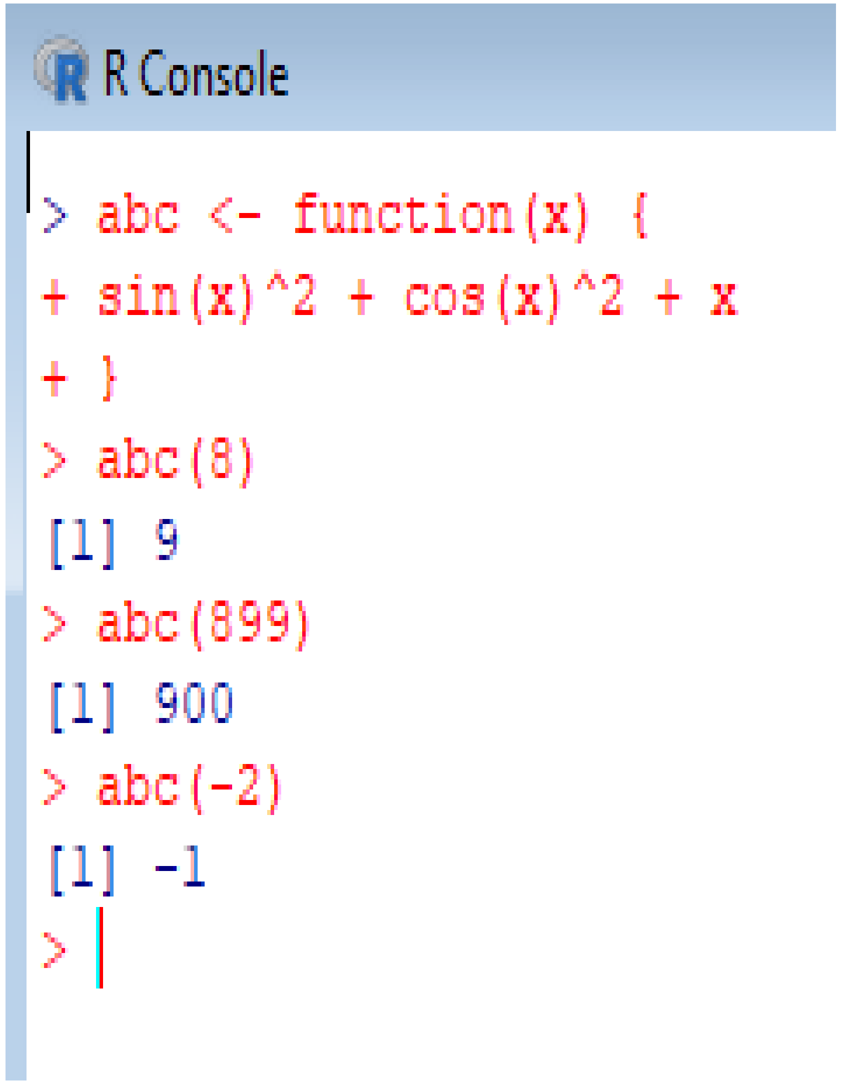
Another example

```
> abc <- function(x){
sin(x)^2+cos(x)^2 + x
}

> abc(8)
[1] 9

> abc(899)
[1] 900

> abc(-2)
[1] -1
```



```
> abc <- function(x) {
+ sin(x)^2 + cos(x)^2 + x
+ }
> abc(8)
[1] 9
> abc(899)
[1] 900
> abc(-2)
[1] -1
>
```

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 Xij (book version) or X[i, j] ("program version"), i = 1,2,...,n, j = 1,2,...,p.

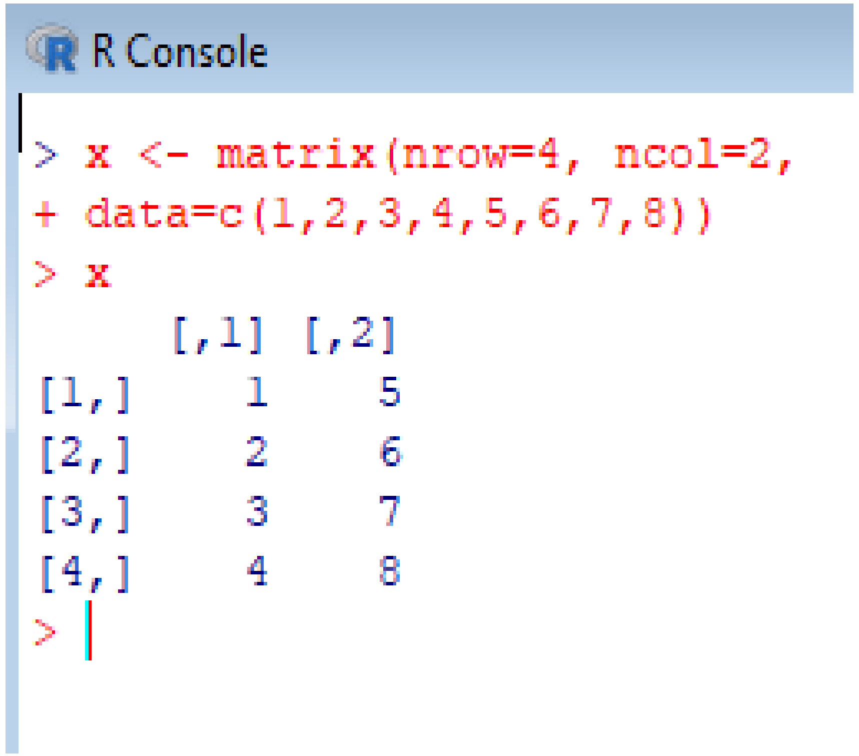
An element of a matrix can also be an object, for example a string.

However, in mathematics, we are mostly interested in numerical matrices, whose elements are generally real numbers

Matrix

In R, a 4 × 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
```



```
> 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
>
```

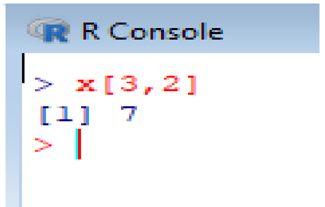
Matrix

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 values from the parameters are written column-wise in 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
```

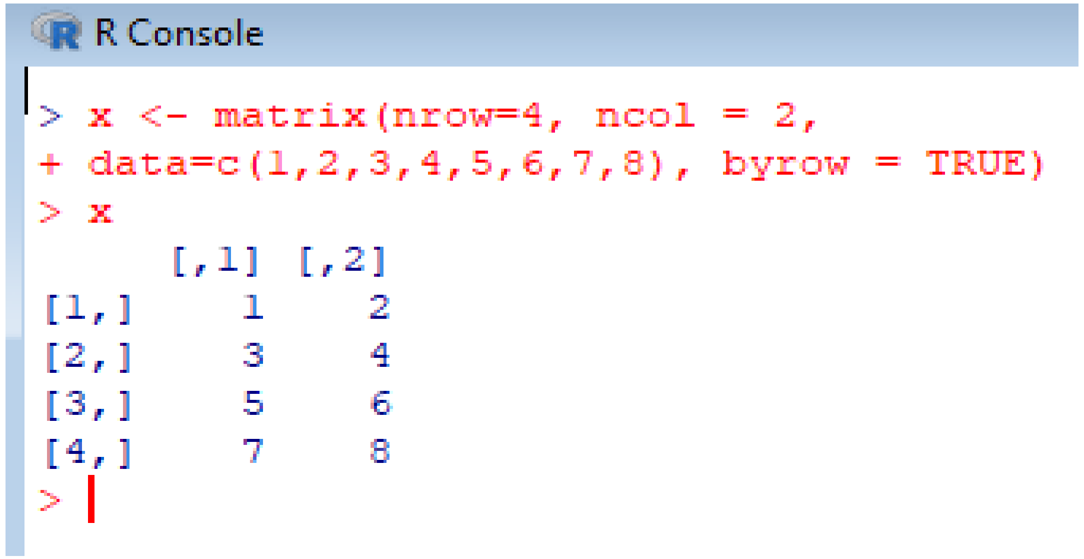


```
> x[3,2]
[1] 7
>
```

Matrix

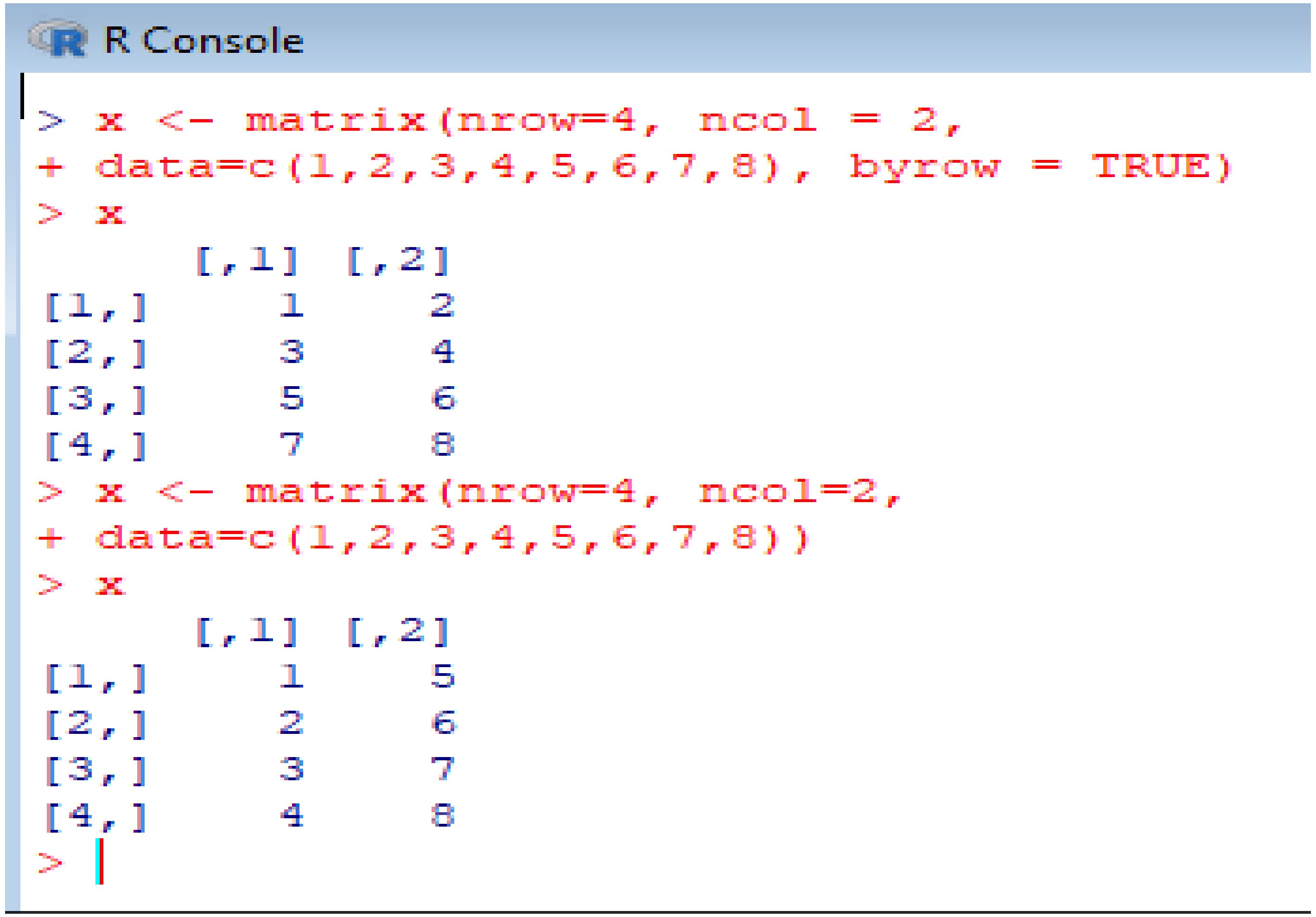
In case, the data has to be entered row wise, then a 4 × 2-matrix X can be created with

```
> x <- matrix( nrow=4, ncol=2,
data=c(1,2,3,4,5,6,7,8), byrow = TRUE)
> x
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```


R Console

```
> x <- matrix(nrow=4, ncol = 2,
+ data=c(1,2,3,4,5,6,7,8), byrow = TRUE)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> 
```

Matrix



```
> x <- matrix(nrow=4, ncol = 2,
+ data=c(1,2,3,4,5,6,7,8), byrow = TRUE)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
> 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
> 
```