Basics of R

What is R?

It is an open-source programming language. Before R, there was a language called S. It was basically designed for the data analyst and for statistical use. It has a limitation that it did not come from traditional programming background. R was created and made public by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of Auckland in 1996. Finally in 2000 R version 1.0.0 was created by the R committee and people associated with S. it was designed as statistical software and data analysis tool.

Many people around the world make contributions in the form of new features, bug fixes or both. These new versions are made available to the public with new releases.

R has many statistical packages. It has maintained S philosophy and also has strong programming for developing new tools.

It is available from the **Comprehensive R Archive Network**, also known as "CRAN".

What is r studio?

It is Integrated Development Environment (IDE) for R. It is a GUI where you can write the code, see the result, see the variables, see the plots etc…
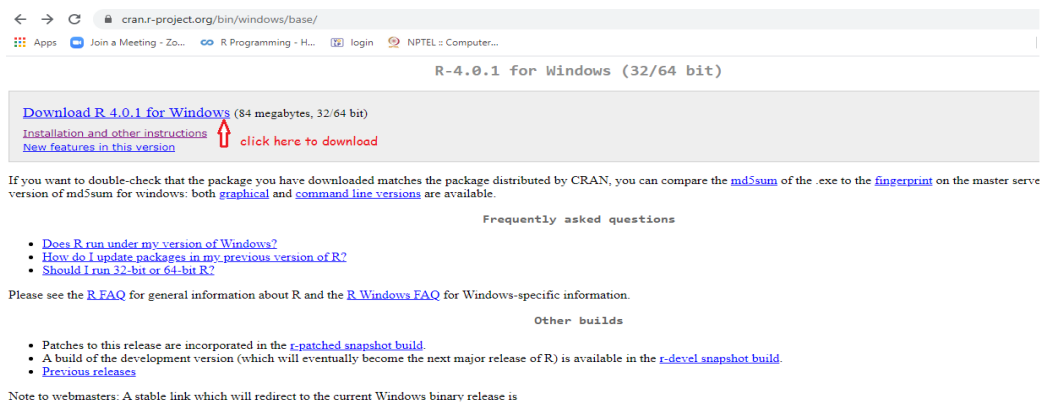
It is available as open source and also as commercial. It has two editions:
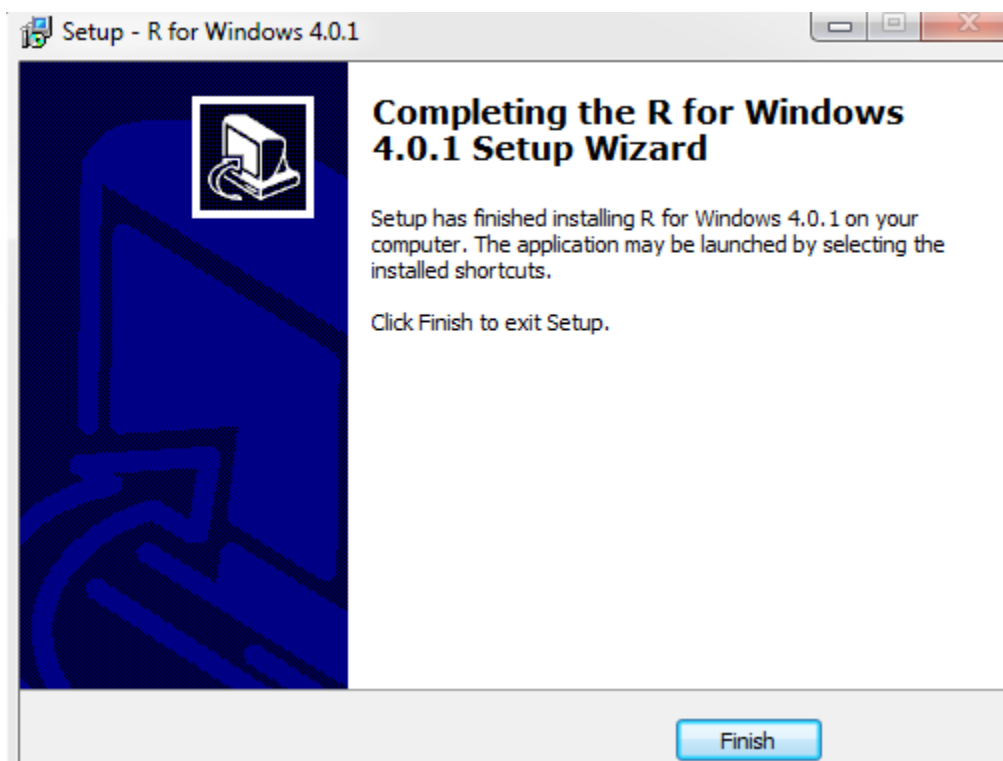
1. Desktop Version
2. Server Version.

How to install R?

Step 1:   Go to link   https://cran.r-project.org/bin/windows/base/

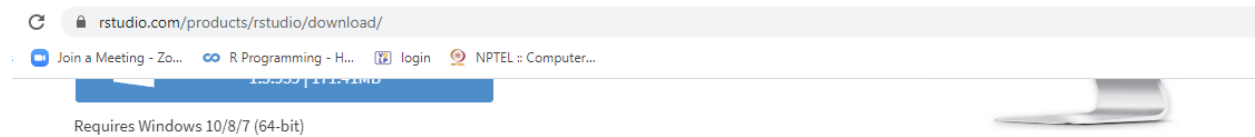**Step 2:** Click on the option   **Download R 4.1.0 for Windows**

Step3: .exe file will get downloaded. Run the file, keep the default selection. Install it.

How to download R studio?

Step1: Go to link   https://rstudio.com/products/rstudio/download/

Step 2: Click on the appropriate option.

Step 3: .exe file will get downloaded. Run the file. Keep the default selection.

The codes will be executed on R studio which has following window panels:

1. Console: You can type the command here and also you can see the printed result.
2. Environment: variables which are generated during the course of programming are stored here.
3. History: All commands which you have used from the beginning can be found here.
4. Files: All files/directories that are available in workspace of R can be seen here.
5. Plots: The plots (graphs/figures) which are generated can be seen here.
6. Packages: It gives the information about the packages installed in R studio. Also you can install new packages as per the requirement.
7. Help: The R document from which you can get help for inbuilt functions.
8. Viewer: To see the local web content generated by R application.
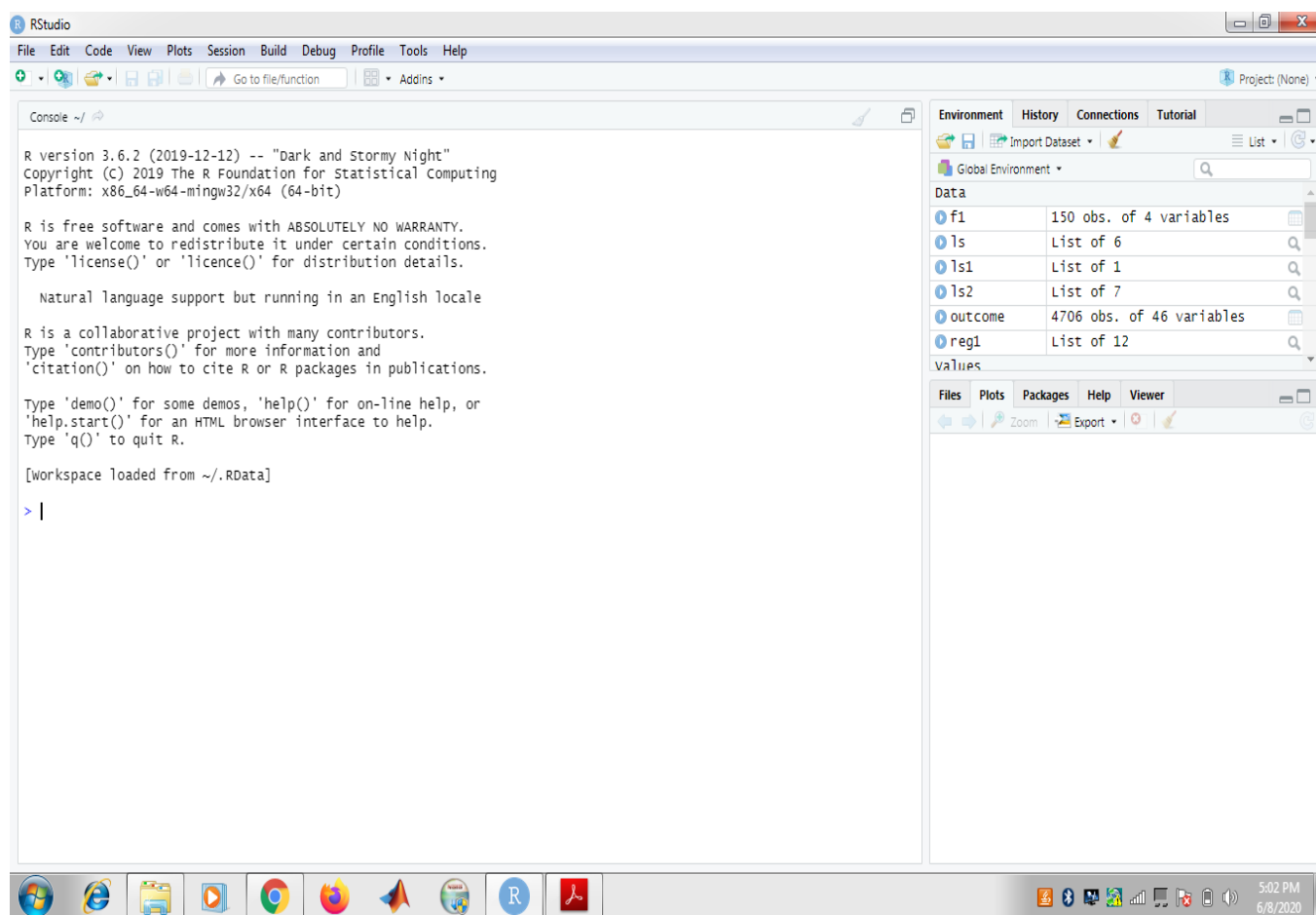
Where to write the code?

The single line command can be executed on the console window. To write a complete code perform the following steps:

Step 1:

Go to File option→ New file→R script
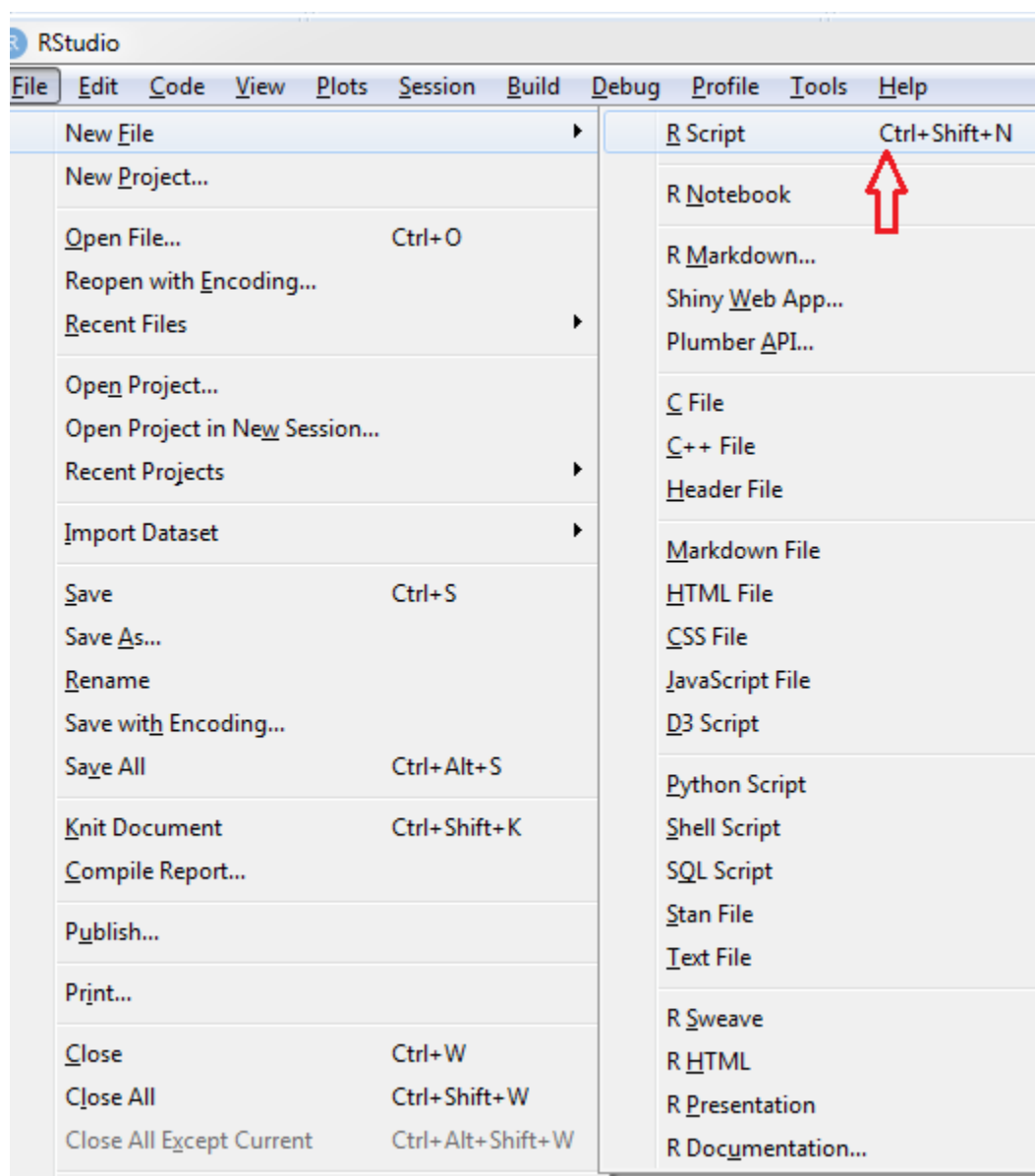
Step 2:

A window panel will get open with "untitled" name. Write the code in the opened panel. Go to File →Save as. Save the code in any of your folder with some relevant name.
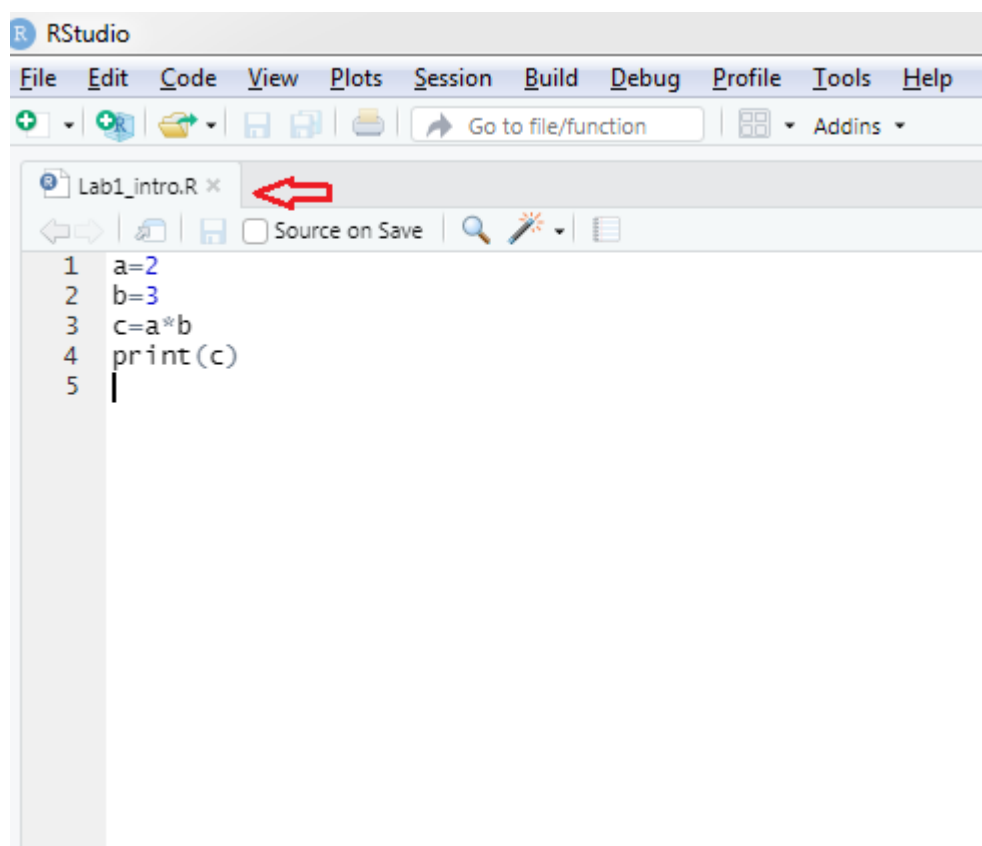
Step 3:

To run the code there are following options:
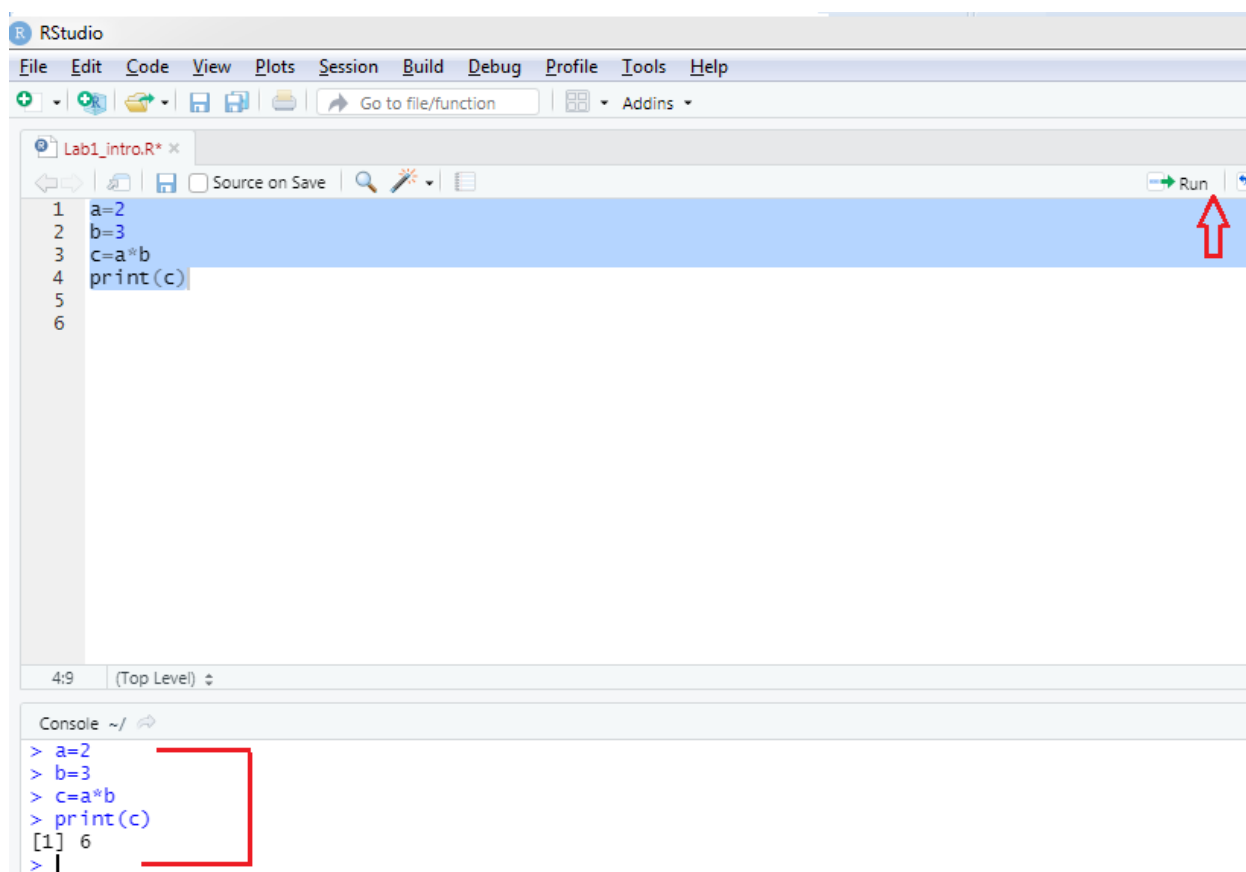
1. **Run (CTRL+Enter):**

   Click Run option present on the top right. This will run only the single statement where you have kept your cursor and the corresponding result can be seen in the console window. For example to run four consecutive statements with run option you have click Run option four times.
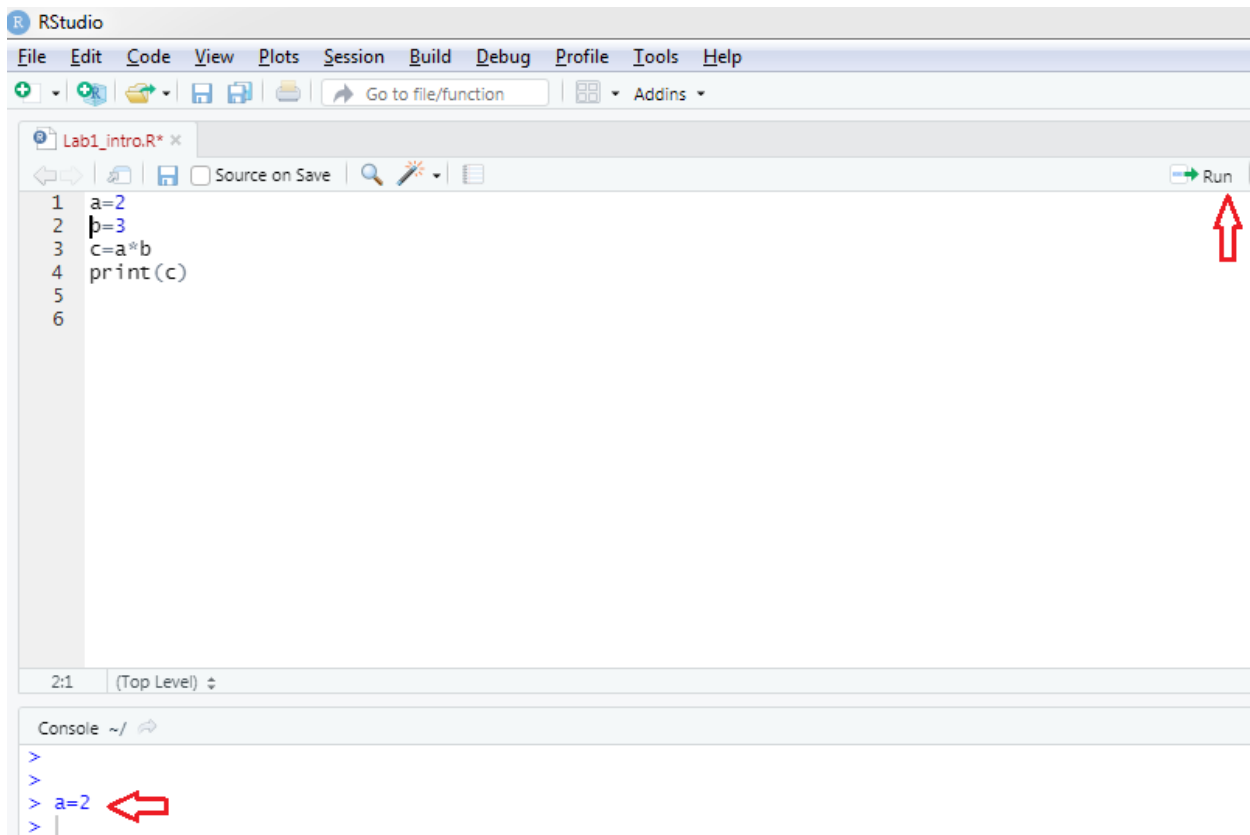
You can also run selected statements together by selecting the required statements and then press Run button.

2. Source(CTRL+SHIFT+S)/Source with Echo(CTRL+SHIFT+ENTER):
Both the options execute the complete file. Only difference is Source displays the output in console window whereas Source Echo displays all the statement along with the output in console.
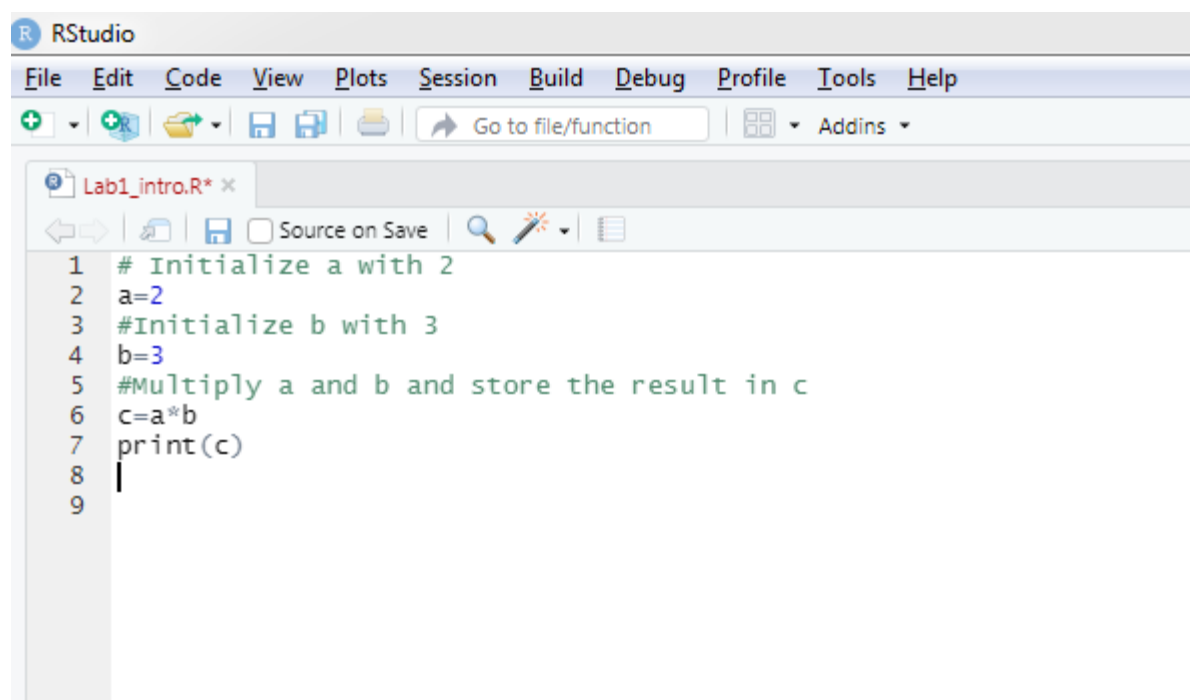
How to make any statement "Comment"?

Comments are used to improve the readability of the code. You can write comment for the statement you have used to explain the purpose of that statement. Indirectly the comments explain about the algorithm. Hence a new user can understand vey easily about why that statement is used.

In R Comments can be given as:

1. **Single line Comment**: Use symbol "#" in front of the statement which you want to use as comment. Here line No.1 and line No 4 are the comments.

```
R RStudio

File  Edit  Code  View  Plots  Session  Build  Debug  Profile  Tools  Help

[icons]  Go to file/function    Addins

Lab1_intro.R* ×

     Source on Save

1   # Initialize a with 2
2   a=2
3   #Initialize b with 3
4   b=3
5   #Multiply a and b and store the result in c
6   c=a*b
7   print(c)
8   |
9
```

2. **Multiple lines**:

Select the multiple lines with cursor and press ctrl+shift+c all selected lines will be converted to comments.

Or

Select multiple lines using cursor, click on the option Code on the left top then select comment/Uncomment lines.
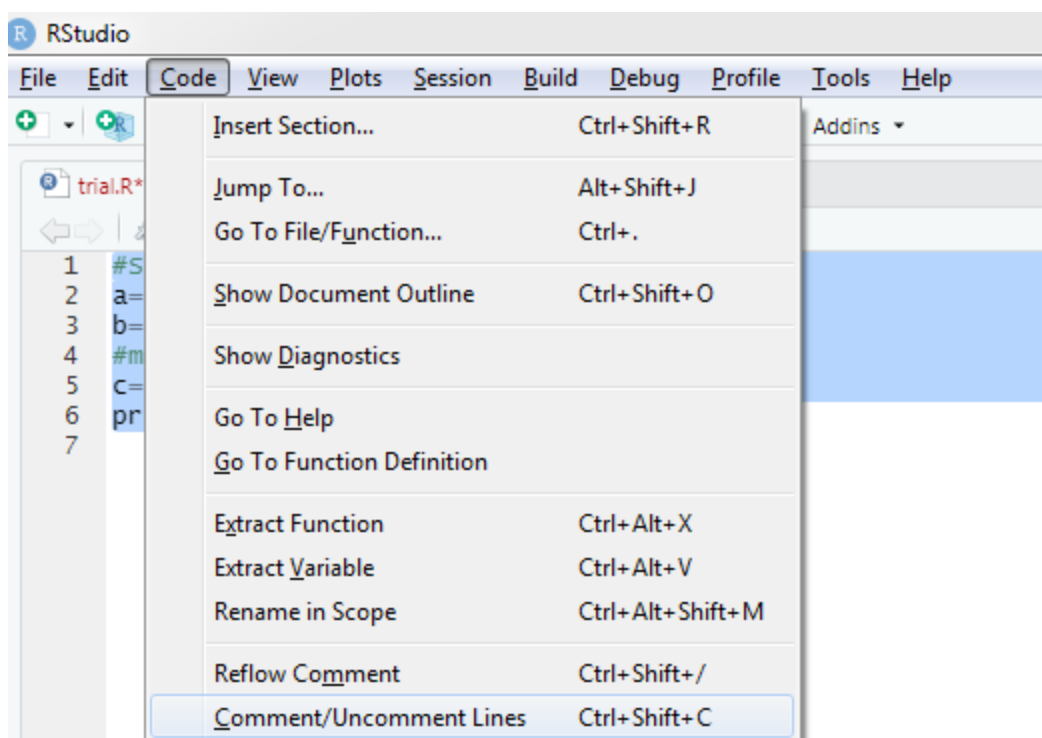
How to clean/clear the console and the variables?

To clear the console you can use the command Ctrl+L. To clear the variables you can use the symbol of brush 🖌 present on the top of Global environment on the top right hand side.
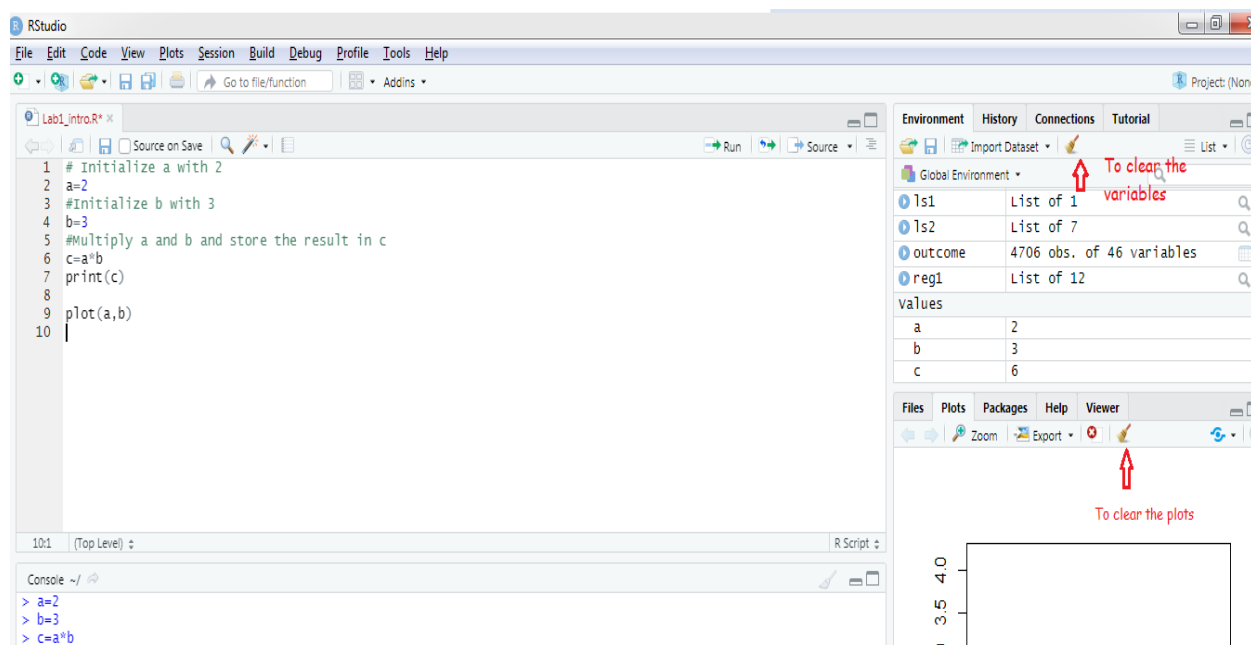
How to install new packages in R?

The functionality which we get after installing R is very basic. The additional functionality can be downloaded as and when required in the form of packages. These packages are created and updated by different developers and can be obtained from different repositories like CRAN,GITHUB etc.

For example, to install the **dslabs**package

>install.packages("dslabs")

OR

In RStudio, you can navigate to the *Tools* tab and select install packages.
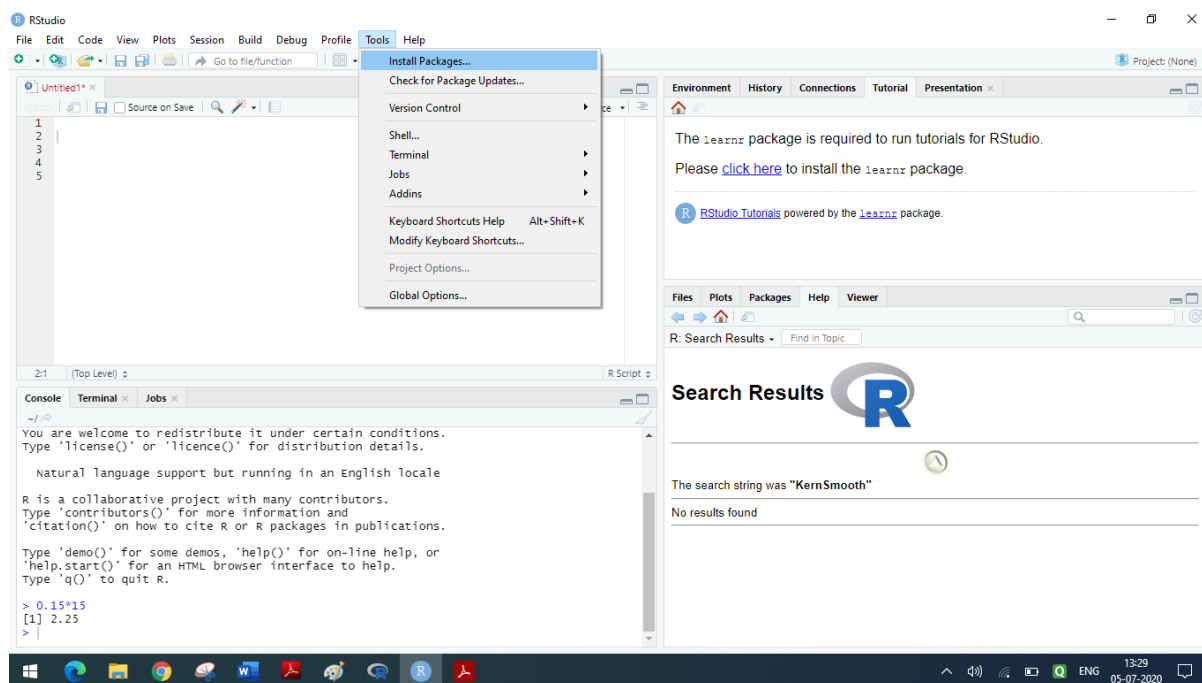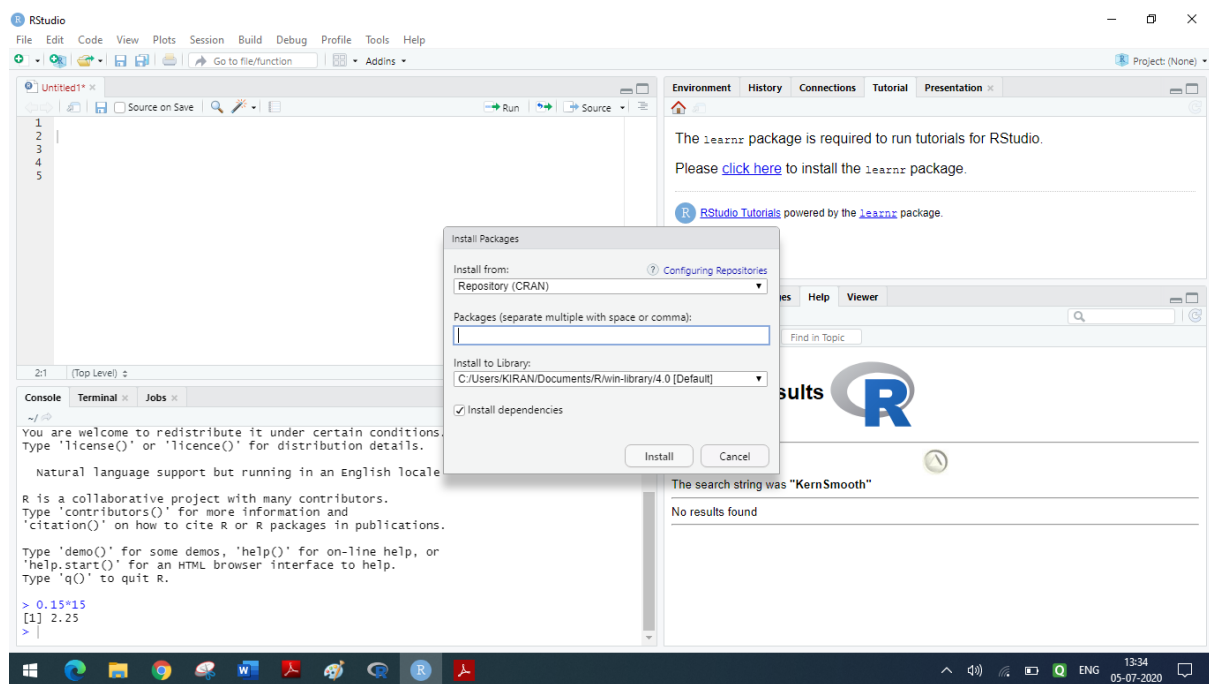
Select all the default settings while installing the package.
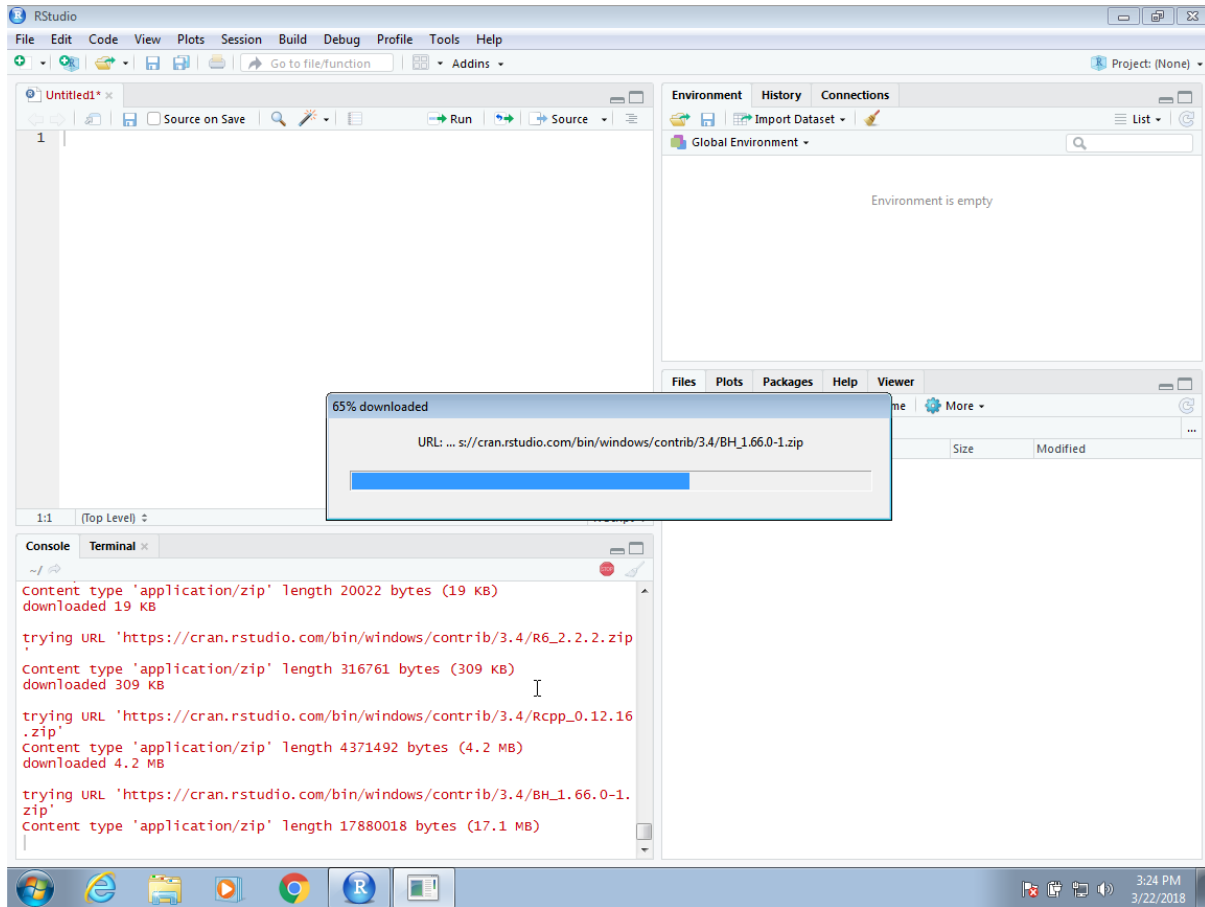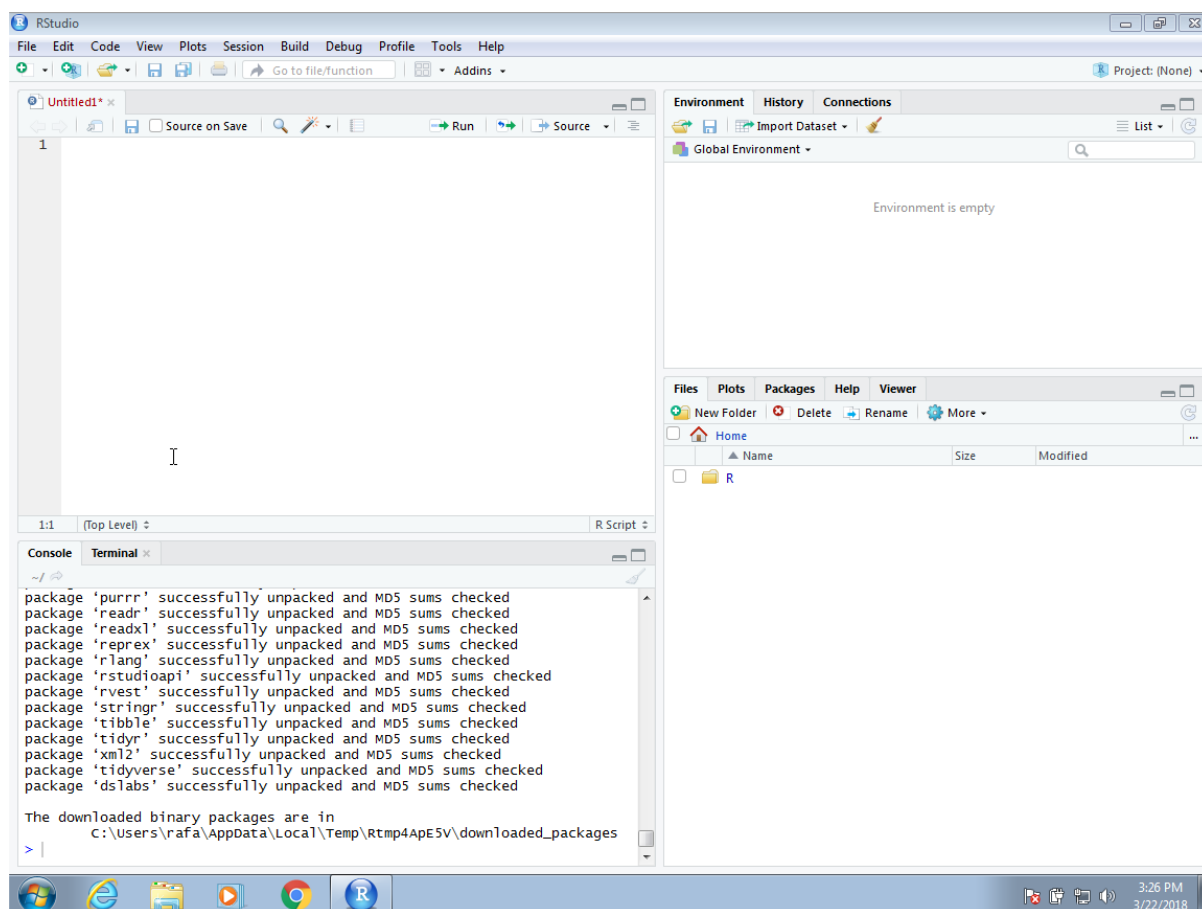
Once the package is installed it can be used in R session by command

library(*package_name*) ie.   **library**(dslabs)

The package remains installed and only needs to be loaded in R session by library command. Once loaded remains till you quit R session. If you try to load a package and get an error, it probably means you need to install it first.

We can install more than one package at once by feeding a character vector to this function:

install.packages(c("tidyverse", "dslabs"))

Note: To save all the objects/variables you can use

>save.image("project.RData) ## any general name but should have extension .RData. When you want to retrieve it back you can select open file option present in File tab on left top above the script pane.

How to get help in R?

Help Command- R contains many inbuilt function. You can get help about these functions by typing

>help (*function_name*)

for example:

>help (sort)

OR

> ?sort

It gives the information about the function (here about the function sort).

You can also get the whole document by typing

>help.start ()

R as a calculator:

- ❖ You can store the values in the objects and can read the values as well. For example let us store 1 in object named "a" and 5 in object named "b"

```
# store 1 in a
>a=1
# You can also use assignment operator  as "=" or "<-"  i.e a<-1
#Initialize b with 3
>b<-3
#Multiply a and b and store the result in c
>c<-a*b

# R will not print anything by its own. If you want to the values stored in a and b either you
 have to type a and b in console
 >a  ## displays value of a
 >b  ## displays value of b
       OR
 you can use print command.

 print(a)
```

- ❖ You can also store more than one value in variable a by concate command. This is called as vector.
  ```
  >a<- c(1,3,4)
  >a
  [1] 1 3 4
  ```
- ❖ You can store data in different variable and then concate then

```
a<-c(1,2,3)
b<-c(4,5,6)
c<-c(a,b)
>c
[1] 1
```

❖ You can use ":" colon operator to store more than one value which is discussed in detail late on.

```
>b<-100:130
[1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
[20] 119 120 121 122 123 124 125 126 127 128 129 130
```

You'll notice that a [1] appears next to your result. R is just letting you know that this line begins with the first value in your result. Some commands return more than one value, and their results may fill up multiple lines. For example, the command 100:130returns 31 values; it creates a sequence of integers from 100 to 130.Notice that new bracketed[ ] numbers appear at the start of the second of output. These numbers just mean that the second line begins with the 20th value in the result.

The colon operator (+) returns every integer between two integers. It is an easy way to create a sequence of numbers. If you type an incomplete command and press Enter, R will display a + prompt, which means it is waiting for you to type the rest of your command. Either finish the command or hit Escape to start over:

For Example:
```
>5 -  ## You have pressed enter after '-' sign. R is waiting for next command.
+
+ 1
[1] 4
```

Exercises:

1. Install R and R studio on your Laptop.
2. Create one folder of your name on your D drive. All the lab assignments will be saved in this folder.
3. Create a R script and write a code to initialize the variables l,m and n with your birth date,birth month and birth year. Then concate these three variables and save the result in DOB variable. Print result DOB. (Hint: To concate the three variables you can use the command -- c(l,m,n).) Save this code with file name as your **roll no for example if your roll number is 30 then save the file as "rn0_30".**
4. Practice the different options as commenting the statements, changing the values of variables, use of run, source, source Echo.

1.   Variables

Variables are used to store some information. Variables are used to store some values which you will require for your program. Creating variables means reserving some memory to store the values. A variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. You can change a variable's data type of the same variable again and again when using it in a program.

Rules for the names of variables:

1.   Allowed characters are alphanumeric characters, '_' , '&' , '.'
2.   It can start with alphabets. It can also start with '.' But then should not be followed by number.
3.   No special character like ! , @ ,#, $ etc are allowed.
For example:

Valid Name	Invalid Name
B2=5	2b=5
.adb=4	.3adb=3
Abc.xyz=4	as@g=1
ABC.XYZ=10	d_!=2

R contains some predefined constants like
> pi
[1] 3.141593
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

2. Data Types:

Following are the data types in R:
1.   Logical
2.   Integer
3.   Numeric
4.   Complex
5.   Character
Each of the above data types has some set of values assigned to it. The class (type) can be checked as follows:
a.   class(*variable_name*)
It gives you one of the above mentioned type.

OR

b.   is.*data_type(variable_name)*
It gives you either true or false depending on the result.

You can also convert the type of the data as follows:
        as.*data_type(variable_name)*

1.   **Logical Type**: It has two values " TRUE" and "FALSE".
For example:
```
> w=5 > 2
> class(w)
[1] "logical"
> w
[1] TRUE
```
Here the value of w is TRUE and its class is logical.

2.   **Numeric Type:** These are double precision real numbers. The numbers which you enter are by default saved as numeric data type. This means that even if you see a number like "1" or "2" in R, which you might think of as integers, they are likely represented behind the scenes as numeric objects (so something like "1.00" or "2.00"). If you explicitly want an integer, you need to specify the L suffix. So entering 1 in R gives you a numeric object; entering 1L explicitly gives you an integer object.

```
>is.numeric(2)
 [1] TRUE
```

Forexample, the operation 2/3 performs real division, which results in 0.666667
```
>x <- 2/3
> print(class(x))
[1] "numeric"
```
It is the default computational data type.

3. Integer Type: 2 is *not* an integer in R. It is a numeric type which is the larger type that contains all floating-point numbers as well as integers. To get an integer you have to make the value explicitly an integer, and you can do that using the function as.integer or writing L after the literal.

```
>is.integer(2)
## [1] FALSE
>is.integer(2L)
## [1] TRUE
>x <- as.integer(2)
>is.integer(x)
## [1] TRUE
```

>class(x)
[1] "integer"

4   Complex Type:

You can use complex numbers as well in R. You can write the complex number as 2+5i. If you want to convert the complex number to numeric or integer then the imaginary part is discarded in   coercion.

e.g:

>d=2+4i
> class(d)
[1] "complex"
> as.integer(d)
[1] 2
Warning message:
imaginary parts discarded in coercion
> as.character(d)
[1] "2+4i"   ## converted to character.

5. **Character Type:** In other languages like "C" you have considered character type for single character and for multi-character you have used String. But in R they both are treated as character type only.

e.g
> d="121"
> f=is.character(d)
[1] TRUE
> as.integer(d)
[1] 121
> as.integer(c)
[1] NA
Warning message:
NAs introduced by coercion
Note: All conversions/coercion from one data type to another is not possible.

The two characters if concate by concate command gives the following output.
>x<-"joe"
> y<-"ram"
> c(x,y)
[1] "joe" "ram"
> paste(x,y)
[1] "joe ram"

Exercise:

1. Assume s=2 and p=2L. Find the class of s and p . Then convert value of s to integer and save it in q. Check the class of q now.
2. If you have taken b=4/3 what will be the output when you run following commands:
   i. as.integer(b)
   ii. class(b)-
   iii. as.numeric(b)
   iv. Use is.integer and is.numeric command for b and state what is the output of both the commands.---

3. Assign 1 to variable x and 2 to variable y. Then define z=x>y. Print the value of z. What is the class of z?
4. Store the string "My SGPA "in variable x, "for last semester is" in variable y and 9.12 in variable z. Now, Print the statement "My SGPA for last semester is 9.12".

Data Objects:
Following are the data objects in R:
1. Vectors
2. Lists
3. Matrices
4. Arrays
5. Factors
6. Data Frames.

1. Vectors:
Vectors are the most basic data object in R. You can create vectors as single element or multi element vector. All the elements of a vector are of same type.

Creating vectors:

When you store more than one value i.e its length becomes greater than 1 it becomes multiple element vector. You can use following methods to store more than one value in a vector:
1. **Using colon operator**:
e.g:
>4:12 ## by default increments by 1.
[1] 4 5 6 7 8 9 10 11 12

>2.2:8.2
[1] 2.2 3.2 4.2 5.2 6.2 7.2 8.2
>2.2:9.1

2.   Using sequence operator:
e.g:
>w=seq(2,20,4)
> w
[1] 2 6 10 14 18

3. **Using Concatenation command**:

```
>x=c(1.1,2.3,4.2)
> x
[1]  1.1 2.3  4.2
> length(x)
[1]  3

>p=c(1,2,"d",4,6,"g")
>p
[1] "1" "2" "d" "4" "6" "g"
>class(p)
>"character
```

Numbers are considered as character if any one element is character.

4. **Using rep command:**

>t=rep(2,5)
[1] 2 2 2 2 2
>f=rep(t,2)
[1] 2 2 2 2 2 2 2 2 2 2

Accessing vector elements using position:
Vector index in R starts from 1.The vector elements can be accessed by their position as follows:
For example:
❖ **With position**
>p=c("jan","feb","mar","apr","may","june","july","aug","sept","oct","nov","dec")
> print(p)
 [1] "jan" "feb" "mar" "apr" "may" "june" "july" "aug" "sept" "oct" "nov" "dec"
> a<-p[1]
Square bracket is used to access the element.
> a

```
[1] "jan"
> c<-p[c(1,2)]
> c
[1] "jan" "feb"


>x=c("mon"=10,"tue"=20,"wed"=30)
> x
mon tue wed
 10  20  30
> b=names(x)
> b
[1] "mon" "tue" "wed"
> k=x["wed"]
> k
wed
 30
```

❖ **With Negative index**
Giving a negative value in the index drops that element from result

```
>d<-p[c(-1,-5,-10)]
> d
[1] "feb" "mar" "apr" "june" "july" "aug"  "sept" "nov"  "dec"
## Here 1st,5th and 10th element is discarded.
```

❖ **With logical operators**

The vector element can be accesses with logical index.
```
>p[c(TRUE,FALSE,TRUE)]  ## Not mentioned positions are considered by default as true and
```
are displayed.
```
[1] "jan" "mar" "apr" "june" "july" "sept" "oct"  "dec"


> q<-c(1:12)
> t<-q[c(5:9)]
>t
[1] 5 6 7 8 9
> t[t>6]
[1] 7 8 9
## The values for which the conditions are true are returned.
```

Modifying a Vector
You can modify a vector using assignment operator.
For example: let vector x contains -3 to 4 values

```
> x
[1] -3 -2 -1  0  1  2  3  4
> x[4]<-100
> x
[1]  -3  -2  -1 100   1   2   3   4
```
The element present at 4ᵗʰ position is modified to 100.
```
> x[x<0]<-0
> x
 [1]  0 0 0 100 1 2 3 4
```
Elements are made to zero with logical condition. All elements less than zero were made equal to 0.

Manipulating the vectors:

Finding the length of vector:
```
> q
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> length(q)
[1] 12
```

Sorting the Vector
By default the sorting is carried out in ascending order. To sort in descending order:

```
>q<-1:12
[1] 1 2 3 4 5 6 7 8 9 10 11 12
>g=sort(q, decreasing=TRUE)
[1] 12 11 10 9 8 7 6 5 4 3 2 1
```

Finding the Maximum minimum and mean value of a vector
```
>s< - -13:19
>max(s)
[1] 19
>min(s)
[1] -13
>mean(s)
[1] 3
```

Performing operations on vector
```
>v1<-c(2,3,6,7,9,10)
> v2<-c(1,4,8,2,3,11)
```

> v1+v2 ## Addition
[1] 3 7 14 9 12 21
> v1-v2 ## Subtraction
[1] 1 -1 -2 5 6 -1
> v1*v2 ## Multiplication
[1]   2 12 48 14 27 110
> v1/v2 ## Division
[1] 2.0000000 0.7500000 0.7500000 3.5000000 3.0000000 0.9090909
> v1%%v2 ## Gives the remainder
[1] 0 3 6 1 0 10

Evaluating powers of vector elements:

>v1
[1] 1 2 3 4 5 6
> sqrt(v1) ## square root
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
> v1^2   ## square of the element
[1] 1 4 9 16 25 36

Exercises:

1. Create a vector of numbers 1, 4, 7, 10, 13… 37 by using seq() function.
2. Create a vector of numbers 5 repeated 10 times by using rep() function.
3. Suppose that x and y are two vector containing elements 1, 5, 2 and 3, 7, 9 respectively
    i.   Augment x by adding y to the left
    ii. Augment y by adding elements 4, 3, 2 at end
    iii. Print the maximum value of Y and minimum value of x
4. Create a vector x with elements 1, 5, 2, 3, 7, 6, 8 and create vectors y, z, w from x using
  $y = x^2$, $z = 1/x$, $w = \log 10x$
5. Suppose age is a vector containing ages of 10 persons as 22, 27, 31, 41, 30, 25, 19, 20, 23, 35
    i.   Access age of fourth person

    ii.   Create a vector age30 with age of person > 30

    iii. Access age of last 3 person

    iv.   Find number of elements in vector age

    v.   Access ages of persons except 5th and 7th

    vi.   Create a vector age2 with age of persons between 20 and 25

LISTS: Unlike vectors, lists can have elements of different types.

For example

Here the variable t is a list which contains two different vectors, one vector having value 1 2 3 4 and other 5 6 7 8. Note that it has not returned the value like concatenation command gives for vectors i.e 1 2 3 4 5 6 7 8. Thus there are two elements in this list which are two vectors.

```
>t=list(1:4,5:8)
> t
[[1]]
[1] 1 2 3 4

[[2]]
[1] 5 6 7 8
```

Following list f consists of three vectors of types character, logical and numeric data type.

Ex1:

```
> v1=c("mon","tue","wed")
> v2=c(T,F)
> v3=56:80
> f=list(v1,v2,v3)
> f
[[1]]
[1] "mon" "tue" "wed"

[[2]]
[1]  TRUE FALSE

[[3]]
 [1] 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

>class(f)
[1] "lists"
```

Ex2:

```
> s<-list(1:3,T,F)
> s
[[1]]
[1] 1 2 3

[[2]]
[1] TRUE

[[3]]
[1] FALSE
```

```
> length(s)
[1] 3

> m<-list(f,s)
> m

[[1]] ## indicates first element of list m
[[1]][[1]]  ## indicates first element of first list ( f[1] – mon,tue,wed )
[1] "mon" "tue" "wed"

[[1]][[2]] ## indicates second element of first list
[1]  TRUE FALSE

[[1]][[3]]
 [1] 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


[[2]] ## second element of list m
[[2]][[1]] ## first element of second list
[1] 1 2 3

[[2]][[2]] ## second element of second list
[1] TRUE

[[2]][[3]]
[1] FALSE


> length(m)  ## list m contains two elements i.e f and s
[1] 2
> length(s)
[1] 3        ## list s contains three elements
> length(f)
[1] 3        ## list f contains three elements
```

Accessing the list:

List can be indexed with the help of [[]] operator. [[]] is member access.

For example:

```
>s
[[1]]
[1] 1 2 3
[[2]]
[1] TRUE
[[3]]
[1] FALSE

> s[1] ## accessing the first element of list s  [] list slicing.
[[1]]
[1] 1 2 3

> s[[1]][[1]]   ## accessing the first element of first element of list s
```

[1] 1
> s[[1]][[3]] ## accessing the third element of first element of list s.

[1] 3

Naming the elements of list:
 The elements of list can be named and then those can be accessed with their names.
For example:
>ls1<-list(a=1:5,b=c("red","blue"),d=c(T,F,T))
> ls1
$a
[1] 1 2 3 4 5
$b
[1] "red" "blue"
$d
[1] TRUE FALSE TRUE

To get a particular column
>ls1$a ## accessing the complete first element of the list
[1] 1 2 3 4 5
>j<-ls1$a[4] ## accessing the 4th element of first element of list
> j
[1] 4

Modifying the list:
    The list can be modified without reloading the elements. You can modify a particular element without disturbing the other elements.
   For example:
   Let us say we want to modify the 4th element of $a to 6 i.e 4 to be replaced by 6.
  >ls1$a[4]=6
>ls1$a
[1] 1 2 3 6 5

$b
[1] "red" "blue"

$d
[1] TRUE FALSE TRUE

Exercise:

1. Create a list named ls. The list ls contains following 6 vectors with their values as follows :
Rollno- 1:4
First name- Ravi,Om,Ajay,Shiv
Last name-Dev,Gandhi,Pande,Rao
Subject-AE,DS,ML,OS
Marks-35,40,38,02
Result –P,P,P,F.
Task To be performed:
1.   Print the list ls.
2.   Print all the independent element of list ls.
3.   Find the class of element of the list.
4.   What will be the output of the following commands:
What will be the output for
a)   Print(ls[[2]][1])
b)   print(ls[[4]][4])
c)   print(ls[5])
5.   It was found that the marks of Ajay were entered wrongly. The correct marks should be 45. Replace the marks of Ajay without disturbing the other elements or without loading the complete list again.
6.   Change the subject name of OS to OE. Modify the required element only.
7.   It was decided that the data base should also have native place information. Ravi stays at Pune, Ajay at Mumbai, Shiv at Nashik and Om at Nagpur. Please add this information also in the list.
8.   Add one more student information names Julie Gommes. The marks obtained by Julie in subject DS is 30 and the result is P. Her native place is Hyderabad.

Homework for Practice:
1. Create the list x which contain numeric vector n(2,3,5), a character vector s("aa","bb","cc","dd") , and logical vector b(T,F,T,F), and numeric value 3. Perform the following task:
 1.From the list print s vector values by list slicing.
2.From the list find s vector values, and numeric value 3, with the help of   index vector by list slicing
3. From the list find s vector values by member access
4. Modify s vector value replacing "aa" by "tt" and again find s