UNIT-1 - R Programming

↳ R studio window has 3 segments:

- Console
- Environment/ History
- Files/ Plots/ Packages/ Help

↳ How to set working directory?

Soln: enter in console: setwd("directory path")

Comments:

↳ for single line comments, insert # at the start of the line.

↳ For multiple lines:

(i) Select multiple lines using the cursor, then press "Ctrl+ shift + C"

(ii) Select multiple lines using cursor, click on "Code" in menu and select "Comment/Uncomment Lines".

clear the console

" control + L "

Clear the environment - rm()

↳ single variable:
Enter in Console / R script : rm(variable)

↳ All variables:
Enter in console / R script : rm(list = ls())

Manual Saving:

- can be permanently saved in a file - save command
- can be reloaded for future sessions - load command

↳ to save a single variable 'a':
save(a, file = "sess1.Rdata")

↳ to save a full workspace with specified file name:
save(list = ls(all.names = TRUE), file = "sess1.Rdata")

ɣshort cut function to save whole workspace :

save.image()

ɣ to load saved workspace :

load(file = "sess1.Rdata")

VARIABLES & DATATYPES IN R:

→ Variable names - Rules :

- allowed characters are alpha numeric, '_' & '.'
- always start with alphabets
- No special characters like !, @, #, $, ...

→ Predefined constants :

① pi

② letters

③ LETTERS

④ ~~moth~~ month.name

⑤ month.abb

⇒ Basic Datatypes:

1. Logical
2. Integer
3. Numeric
4. Complex
5. Character

* typeof(object) → to find the datatype of the object
* as.datatype(object) → coerce or convert data type of object to another.
* is.datatype(object) → verify if object is of certain datatype.

⇒ Basic objects:

1. vector → ordered collected of same datatypes
2. List → ordered collection of objects
3. Dataframe → generic tabular object.

① Vectors

↳ ordered collection of basic datatypes of given length.
↳ all the elements of a vector must be of same data type.

eg:
```
X = c(2.3, 4.5, 6.7, 8.9)
print(X)
```

② Lists:

↳ generic object consisting of an ordered collection of objects
↳ a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a functio, and so on...

eg:
```
ID = c(1, 2, 3, 4)
emp.name = c("Mon", "Rag", "Sha", "Din")
num.emp = 4
emp.list = list(ID, emp.name, num.emp)
print(emp.list)
```

↳ all components of a list can be named
↳ these components can be accessed using the given names.

emp.list = list("Id" = ID, "Names" = emp.name, "Total staff" = num.emp)

print(emp.list$Names)

↓

[1] "Man" "Rag" "Sha" "Din"

↳ to access top level components, use double slicing operator "[[]]" and for lower/inner level components use "[]" along with "[[]]"

print(emp.list[[1]]) or print(emp.list[1])

↳ $Id

[1] 1 2 3 4

print(emp.list[[2]]) or print(emp.list[2])

↳ $Names

[1] "Man" "Rag" "Sha" "Din"

print(emp.list[[1]][1])

↳ [1] 1

print(emp.list[[2]][1])

↳ [1] "Man"

→ List Manipulation.

↳ a list can be modified by accessing components & replacing them.

eg:

```
emp.list['Total Staff'] = 5
emp.list[[2]][5] = "Nir"
emp.list[[1]][5] = 5
```

→ List Concantenation

↳ two lists can be concatenated using the concatenation function `c(list1, list2)`

eg:

```
emp.ages = list("ages" = c(23, 48, 54, 30, 32))
emp.list = c(emp.list, emp.ages)
print(emp.list)
```

↳

```
$Id
[1] 1 2 3 4 5
$Names
[1] "Man" "Rag" "Sha" "Din" "Nir"
$Total staff
[1] 5
```

```
$ages
[1] 23 48 54 30 32.
```

DATAFRAMES

↳ dataframes are generic data objects of R, used to store tabular data.

eg:

```
vec1 = c(1,2,3)
vec2 = c("R", "Scilab", "Java")
vec3 = c("For prototyping", "For prototyping", "For Scaleup")
df = data.frame(vec1, vec2, vec3)
print(df)
```

	vec1	vec2	vec3
1	1	R	For prototyping
2	2	Scilab	For prototyping
3	3	Java	For Scaleup

↳ a dataframe can also be created by reading data from a file using the following command:

newDF = read.table(path = "Path of the file)

↳ in path use '/' instead of '\'

↳ a separator can also be used to distinguish b/w entries. Default separator is space, ' '

newDF = read.table(file="path of file", sep)

→ Accessing Rows and columns:

↳ df[val1, val2] refers to row 'val1', coloumn 'val2'. it can be a number or a string.

↳ "val1" or "val2" can also be array of values like "1:2" or "c(1,3)"

↳ df[val2] (no commas) - just refers to column "val2" only.

eg:

print(df[1:2,]) → first & second row

print(df[,1:2]) → first & second column.

print(df[1:2]) → first & second column.

SUBSET

- subset() which extracts subset of data based on conditions

eg:

```
pd = data.frame("Name" = c("Senthil", "Senthil", "Sam",
    "Sam"), "Month" = c("Jan", "Feb", "Jan", "Feb"),
  "BS" = c(141.2, 139.3, 135.2, 160.1),
  "BP" = c(90, 78, 80, 81))

pd2 = subset(pd, Name == "Senthil" | BS>150)
print("new subset pd2")
print(pd2)
```

(1) "new subset pd2"

	Name	Month	BS	BP
1	Senthil	Jan	141.2	90
2	Senthil	Feb	139.3	78
4	Sam	Feb	160.1	81

→ Editing Dataframes

↳ dataframes can be edited by direct assigmment

eg: df[[2]][2] = "R"

↳ dataframe can also be edited using the edit() command

- create an instance of dataframe and use edit command to open a table editor, changes can be manually made.

eg:

```
myTable = data.frame()
myTable = edit(myTable)
```

→ Adding extra rows and columns:

↳ rbind → for rows

cbind → for columns.

eg:

```
df = rbind(df, data.frame(vec1 = 4, vec2 = "C",
vec3 = "For Scaleup"))
df = cbind(df, vec4 = c(10,20,30,40))
```

→ Deleting rows and columns

eg.
df2 = df[-3, -1] → deleting rows and columns
↳ A -ve (-) sign before value and before ',' for rows & after ',' for colums.

df3 = df[, !names(df) %in% c("vec3")]
↳ '!' means no to these rows/columns which satisfy the condition

df4 = df[!df$vec1==3,]

→ Manipulating rows - the factor issue

New entries
- when character columns are created in a data.frame, they become factors
- Factor variables are those where the character column is split into categories or factor levels.

eg: df[3,1] = 3.1
df[3,3] = "others"
print(df).

Warning message:
↳ invalid factor level.

→ Resolving factor issue:

↳ New entries need to be consistent with factor levels which are fixed when the dataframe is first created.

eg:
vec1 = c(1,2,3)
vec2 = c("R", "Scilab", "Java")
vec3 = c("For prototyping", "For prototyping", "For Scaleup")
df = data.frame(vec1, vec2, vec3, stringAsFactors = F)
df[3,3] = "Others"
print(df)

↳ Now no error.

⇒ Recasting Dataframes:

↳ recasting is the process of manipulating a data frame in terms of its variables.
↳ reshaping the data.

Consider the eg:

```
pd = data.frame("Name" = c("Senthil", "Senthil", "Sam", "Sam",
     "Month" = c("Jan", "Feb", "Jan", "Feb"),
     "BS" = c(141.2, 139.3, 135.2, 160.1),
     "BP" = c(90, 78, 80, 81))
print(pd)
```

identifier variables → Name, Month
measurement variables → BS, BP

	Name	Month	BS	BP
1	Senthil	Jan	141.2	90
2	Senthil	Feb	139.3	78
3	Sam	Jan	135.2	80
4	Sam	Feb	160.1	81

☆ categorical & Date variables can not be measurements.

↳ 2 steps are involved in recasting:
① Melt
② Cast.

① Melt :

↳ call the library 'reshape 2' using the library() command

```
melt(data, id.vars, measure.vars, variable.name = "variable",
     value.name = "value")
```

eg:

```
library(reshape2)
Df = melt(pd, id.vars = c("Name", "Month"),
     measure.vars = c("BS", "BP"))
print(Df)
```

↓

	Name	Month	variable	value
1	Senthil	Jan	BS	141.2
2	Senthil	Feb	BS	139.3
3	Sam	Jan	BS	135.2
4	Sam	Feb	BS	160.1
5	Senthil	Jan	BP	90.0
6	Senthil	Feb	BP	78.0
7	Sam	Jan	BP	80.0
8	Sam	Feb	BP	81.0

② Cast

↳ applying the dcast() function

```
dcast(data, formula, value.var= col.with values)
```

eg:

```
df2= dcast(Df, variable + month ~ Name, value.var= "value")
print (df2)
```

columns "variable" & "month" to remain as is
categories in column "Name" become new variables

Column of df from which the values are to be taken from.

⇒ Recasting in a single step:

↳ recast function

```
recast(data, formula, ..., id.var, measure.var)
```

⇒ Add new variable to dataframes based on existing ones:

↳ 'dplyr' → library using library() command.

↳ mutate() command will add extra variable columns based on existing ones.

eg: library(dplyr)

pd2 ← mutate(pd, log_BP = log(BP))

print(pd2)

⇓

	Name	Month	BS	BP	log_BP
1	Senthil	Jan	1412	90	4.499810
2	Senthil	Feb	1393	78	4.356709
3	Sam	Jan	1382	80	4.382027
4	Sam	Feb	1601	81	4.394449

⇒ Joining of 2 Frames

function(df1, df2, by = id.variable)

↳ common to both df

↳ provides identifiers for combining the 2 dataframes

↳ call the library 'dplyr' command using the library() command.

↳ the following commands would be used to combine datasets:

1. left_join()
2. right_join()
3. inner_join()
4. full_join()
5. semi_join()
6. anti_join()

eg:

let first data frame be pd:

	Name	Month	BS	BP
1	Senthil	Jan	141.2	90
2	Senthil	Feb	139.3	78
3	Sam	Jan	135.2	80
4	Sam	Feb	160.1	81

let second dataframe be pd_new

	Name	Department
1	Senthil	PSE
2	Ramesh	Data Analytics
3	Sam	PSE

① left_join

↳ joins matching rows of df1, df2 based on the id.variable

↳ if we consider id.variable = "Name", only "Senthil" & "Sam" are matching rows

↳ the variable ~~from~~ "Department" from "pd_new" would be merged to its 'left' to pd.

```
library(dplyr)
pd_left_join <- left_join(pd, pd_new, by = "Name")
```

⇓

	Name	Month	BS	BP	Department
1	Senthil	Jan	141.2	90	PSE
2	Senthil	Feb	139.3	78	PSE
3	Sam	Jan	135.2	80	PSE
4	Sam	Feb	160.1	81	PSE

② right_join

pd_right_join1 ← right_join(pd, pd_new, by = "Name")

③ inner_join()

↳ merges and retains those rows with IDs present in both dataframes.

pd_inner_join1 ← inner_join(pd_new, pd, by = "Name")

⇓

	Name	Department	Month	BS	BP
1	Senthil	PSE	Jan	141.2	90
2	Senthil	PSE	Feb	139.3	78
3	Sam	PSE	Jan	135.2	80
4	Sam	PSE	Feb	160.1	81

ARITHMETIC OPERATIONS IN R:

1. =, <- ⇒ Assignment
2. + ⇒ Addition
3. - ⇒ Subtraction
4. * ⇒ Multiplication
5. / ⇒ Division.
6. ^, ** ⇒ Exponent
7. %%. ⇒ Remainder
8. %/% ⇒ Integer devision.

→ Hierarchy of operations:

Order of Precedence	Operation
Bracket	( )
Exponent	^, **
Division	/
Multiplication	*
Addition & Subtraction	+, -

Logical Operations in R:

1. < ⇒ less than
2. <= ⇒ less than equal to
3. > ⇒ greater than
4. >= ⇒ greater than equal to
5. == ⇒ exactly equal to
6. != ⇒ Not equal to
7. ! ⇒ not
8. | ⇒ or
9. & ⇒ and
10. isTRUE ⇒ Test if variable is TRUE

Matrix Operations in R:

↳ a matrix is a rectangular arrangement of numbers in rows and columns.
↳ Rows run horizontally and columns run vertically

eg: A = matrix(c(1,2,3,4,5,6,7,8,9), nrows = 3, ncol = 3, byrow = TRUE)

* for a matrix where all rows and columns are filled by a single constant K.

matrix (K, m, n)

* diagnal matrix:

diag (K, m, n)

* identity matrix:
use diag() command with K=1

* dim (A) will return the size of the matrix
* nrow(A) will return the no. of rows
* ncol (A) will return the no. of columns.
* prod(dim (A)) or length (A) will return the number of elements.

- Array/value before ';' for accessing rows
- Array/value ~~before~~ after. ',' for accessing columns.

- use of '-' for removing rows/columns
- strings can be assigned as names of rows and columns using:

colnames(A) ← c("a", "b", "c")
rownames(A) ← c("d", "e", "f")

* to access the last row → A[nrow(A),]

→ Colon operator

Colon operator can be used to create a row matrix

eg: 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

→ Colon operator: sub matrices selection

↳ the colon notation can also be used to pick sub-matrices.

eg: A[1:3, 1:2]

⇒ Matrix Concantenation :

↳ merging of a row or column to a matrix
↳ concantenation of row to a matrix is done using rbind()
↳ concantenation of a column to a matrix is done using cbind()

eg : $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ $B = [10 \quad 11 \quad 12]$

C = rbind(A,B)

	[,1]	[,2]	[,3]
[1,]	1	2	3
[2,]	4	5	6
[3,]	7	8	9
[4,]	10	11	12

⇒ Matrix Algebra :

- Addition / subtraction
- Multiplication
- ~~Matrix Oper~~ Division.

→ Regular matrix multiplication.

`A %*% B`

→ Element wise multimaplication

`A * B`

→ Matrix Division

↳ element wise division.

$A/B = \frac{a_{ij}}{b_{ij}}$

Functions in R :

- a function accepts input arguements and produces output by executing valid R commands present in the function
- function name and file name need not be the same.
- a file can have one or more function definitions.
- functions are created using the command function()

```
f= function (arguments) {
        statements
}
```

- function files have to be loaded before invoking

→ Passing arguements to functions.

- passed in the same order as in function definition
- Names of the arguments can be used to pass their values in any order
- default values are used if some or all arguments are not passed.

⇒ Function with multiple inputs and outputs..

↳ Functions in R take multiple input objects but returns only one object as output.

↳ but a list object can be returned by function

→ Inline functions:

```
func = function(x) x^2+4**x +4
```

Looping over objects:

↳ there are a few looping functions that are pretty useful when working interactively on a command line

① apply: apply a function over the margins of an array or matrix

② lapply: apply a function over a list or a vector

(3) tapply : apply a function over a ragged array

(4) mapply: Multivariate version of lapply.

(5) xxply (plyr package)

(1) apply function:

↳ applies a given function over the margins of a given array.

syntax: apply (array, margins, function, ...)

• Here the margins refer to the dimension of the array along ~~wit~~ which the function need to be applied.

(2) lapply function:

↳ used to apply a function over a list
↳ always returns a list of the same length as the input list

Syntax: lapply (list, function, ...)

⑤ mapply function:

↳ mapply is a multivariate version of lapply.
↳ a function can be applied over several lists simultaneously

↳ Syntax: mapply (fun, list1, list2, ...)

⑥ tapply function:

↳ tapply is used a function over subset of vectors given by a combination of factors.

↳ Syntax: tapply (vector, factors, function, ...)

CONTROL STRUCTURES:

↳ execute certain commands only when certain condition(s) is satisfied (if-then-else)
↳ execute certain commands repeatedly and use a certain logic to stop the iteration (for, while loops)

1 → If else family of constructs:

→ if (condition){
    statements
}

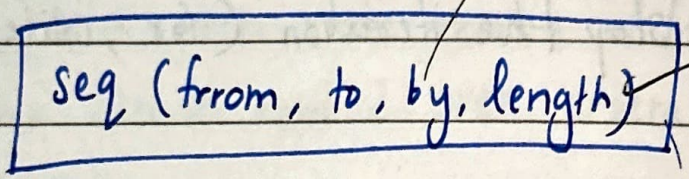
→ if (condition){
    statements
} else {
    alternate statements
}

→ if (condition){
    statements
}elseif(condition){
    alternate statements
}else {alternate statements}

→ Sequence Function :

↳ a sequence is one of the components of a 'for loop'

• Sequence function syntax :



• creates equi-spaced points between 'from' and 'to'

→ for loop, Nested for loops :

↳ the structure of a for loop construct comprises :

- A 'sequence' which could be a vector or a list.
- 'iter' is an element of the sequence
- statements.

↳ Nested-for-loop : one or more for loop constructs are located within another.

→ for (iter in sequence) {
    statements
}

→ for (iter1 in sequence1) {
    for (iter2 in sequence2) {
        statements
    }
}

→ For-loop with if-break:

↳ a 'break' statement once executed, program exits the loop even before iterations are complete.
↳ 'break' command comes out of the innermost loop for nested loops.

→ While loop:

↳ used whenever you want to execute statements until a specific condition is violated.

Data Visualization in R :

① Scatter plot :

~~X:1~~

```
x = 1:10
y = x^2
plot(y)
```

② Line plot :

```
x = 1:10
y = x^2
plot(x, y, type = 'l')
```

③ Bar plot :

```
barplot(H, names.arg, xlab, ylab, main, names.arg, co
```

eg:

```
H ← c(7, 12, 28, 3, 41)
M ← c("Mar", "Apr", "Mar", "Jun", "Jul")
barplot(H, names.arg = M, xlab = "Month", ylab = "Revenue",
col = "blue", main = "Revenue chart", border = "red")
```