R

* R is an object oriented scripting language that provides an environment for statistical computing and graphical representation
* It is commonly used to analyze and visualize data

Why use R?

1. It is a great resource for data analysis, data visualization, data science and machine learning.

2. It provides many statistical techniques such as
   * statistical tests
   * regression
   * classification
   * clustering
   * data-reduction

3. It is easy to draw graphs (pie charts, histograms, box plot.....)

4. It provides wide range of packages (libraries of functions)

5. It works of different platforms (Windows, Mac, Linux)

6. It is open source & free. Also has a large community support.

7. Intro to Programming in R

User Input

To get input/inputs from the user use the built-in function readLines. (The data type of the variable = string/character)

Ex: x_input <- readLines(stdin(), n=1)

the above example reads an input from the user. "n" specifies the number of lines to read.

Output statements:

R provides print function to print one variable at a time or display a string on to the console. Print statement does not display multiple items at once.

Ex:

```
x ← 4
print(x)
print("The value of x:")

print("x =", x)
y <- 20
print(x, y)
print(y, x)
```

output:

```
4
The value of x:

x =
4
20
```

Use cat() function to address the shortcoming of print function.

Ex:

```
cat(x, y)
cat(x, y, sep=",")
```

```
4 20
4,20
```

Example

output :

```
cat ("Please enter a value")
value_1 <- readLines(stdin(), n=1)
class(value_1)
```

character

Set working directory

```
setwd( path )
```

Variables: They are data storage containers having a label.

Identifiers: They are used as label/names for variables.

Rules for identifiers:

1. R is case sensitive
   Ex: rate, Rate, RATE are three distinct identifiers

2. An identifier can contain
   * letters or
   * numbers or
   * 2 special characters : dot (.) or underscore (_)

   Ex: first_name, data1, data_1, data.2, length

3. An identifier cannot start with a number or underscore
   Ex: 12_value, _height are not allowed

4. An identifier can start with a dot but cannot be followed by a number
   Ex: .variable_2, .data.2 are allowed
   .12x, .1_value are not allowed

5. Keywords cannot be used as identifiers
   Ex: if, else, while, for are not allowed

* Like do not have to specify the data type of a variable
* It is extracted implicitly based on the type of data stored inside a variable
* R doesnot have a commad for variable declaration
* Simply assigining a value to a variable will be considered as declaration and assignment
  i.e., A variable is created the moment you assign a value to it

Syntax for variable declaration and assignment :

identifier <- value

Ex: age <- 40
name <- "luffy"

Multiple Variables :

* R allows us to assign the same value to multiple variables in one line

Ex: var_1 <- var_2 <- var.10 <- 90

all three variables ; var_1, var_2, var.10 contains the value 90

Note : # symbol is used for commenting (both for single line and multiple lines)

Ex: # This is a comment
Multiple comments
No syntax for multiple comments

Data Types :

* A data type defines the kind of data stored in a variable.
* Variables do not need to be declared with any particular type and can change the type after they have been set.

Basic data types :

1. numeric : these stores numeric values, real or floating points

```
Ex:    435 ,   4.14
       value_1 <- 4.14
       value_2 <- 21
```

2. integers : used for storing integers only.
   letter **L** is used to declare as an integers

```
Ex:    1L ,   1041L
       value_1 <- 1L
```

3. complex : used to store complex numbers and must carry real and imaginary parts

```
Ex:    2+3i ,  -5+6i
       value_1 <- 2+3i
```

4. logical : used to store boolean values

```
Ex:    TRUE , FALSE
       flag <- TRUE
```

Note : In R '0' is considered as FALSE and non-zero numbers including negative numbers are considered as TRUE

5. Characters : used to store strings. Values must be enclosed in a single or double quotes.

Ex: "x" , 'iphone'

```
var_1 <- 'x'
var_2 <- "iphone"
```

* A data type of a variable can be determined using class() function or typeof() function

Syntax: class(variable)
typeof(variable)

Ex:
```
x_value <- 10.5
class(x_value)
```
output : numeric

```
x_name <- "luffy"
typeof(x_name)
```
output : character

Type Conversion: We can convert from one type to another with the following functions

as.numeric(variable)
as.integer(variable)
as.complex(variable)

Ex:
```
var_1 <- 1L
var_2 <- as.numeric(var_1)
class(var_2)
var_3 <- 2
var_4 <- as.integer(var_3)
class(var_4)
```
output : numeric
integer

R operators

* An operator is a symbol or a group of symbols that performs computational tasks on variables and values

R provides the operators in the following groups:

1. Assignment operators
2. Arithmetic operators
3. Comparision (relational) operators
4. Logical operators
5. Miscellaneous operators

R Assignment operators (<-)

They are used to assign values to a variable.

Ex: var_1 <- 41.2

R Arithmetic operators

They are used with numeric values to perform common mathematical operations.

x <- 7 y <- 3

Operator	Discription	Example	Result
+	Addition	x + y	10
-	Subtraction	x - y	4
*	Multiplication	x * y	21
/	Division	x / y	2.33
^	Exponent	x ^ y	343
%%%	Modulo (remainder)	x %%% y	1
%/%	Integer division	x %/% y	2

R Comparision / Relational operators

They are used compare two values or variables. The output of any relational operator is always a boolean

```
x <- 7        y <- 3
```

Operator	Description	Example	Result
==	Equal to	x == y	FALSE
!=	Not equal to	x != y	TRUE
<	Less than	x < y	FALSE
>	Greater than	x > y	TRUE
<=	Less than or equal to	x <= y	FALSE
>=	Greater than or equal to	x >= y	TRUE

R Logical operators

They are used to perform logical operations. Also used to combine conditional statements. The inputs and output of any logical operators are always a boolean.

```
x <- TRUE        y <- FALSE
```

Operator	Description	Example	Result
||	Logical OR	x || y	TRUE
&&	Logical AND	x && y	FALSE
!	Logical NOT	!x	FALSE

R Miscellaneous operators

:	creates a series of numbers in a sequence	x <- 1:10
%in%	Finds if an element belongs to a vector	x %in% y
%*%	Matrix multiplication	x <- M-1 %*% M-2

Control Structures

To make a decision or to execute a piece of code over and over we use control structures

Simple if statement

Here a specific block of code is executed if a condition is TRUE

Syntax:

```
if (boolean_expression) {
    statements
}
```

* if the boolean_expression results in TRUE then the statements are executed.
* if the boolean_expression results in FALSE then the control flow does not enter the block of code, i.e., the statements are not executed

Ex:

```
if (value_1 > 3){
    cat ("You are in the upper floor.")
}
```

if-else statement

It is used when we have to choose between two available options based on a condition

Syntax :

```
if (boolean_expression){
    statements_yes
} else {
    statements_no
}
```

* if the boolean_expression results in TRUE then statements_yes are executed
* if the boolean_expression results in FALSE then statements_no are executed

Note : The else statement must appear immediately after the end of '}' for the if statement

Example :

```
if (value > 3) {
  cat ("you are in upper floor.")
} else {
  cat ("you are in lower floor.")
}
```

if - else if - else statement

It is used when you have to choose from available set of options. based on different conditions.

Syntax :

```
if ( boolean_expression_1 ) {
        statements_1
} else if ( boolean_expression_2 ) {
        statements_2
} else if ( boolean_expression_3 ) {
        statements_3
} else if
        :
} else {
        statements_last
}
```

* if the boolean_expression_1 results in TRUE then the set of statements_1 are executed
* if the boolean_expression_1 results in FALSE and the boolean_expression_2 results in TRUE then statements_2 executes
* if none of the statements are executed (i.e., if all the boolean_expression results in FALSE) then statements_last executes.

Note: Only one of the statements are executed in the entire if - else if - else statements

Nested if-else statements

We can have any if-else statements within an if-else statement. Also applies for a simple if statements or if-else if-else statements.

Switch function

In R, switch is used to test an expression against a list of elements.

Syntax :

switch (expression , list of elements seperated by a ",")

Ex:

switch (2 , "Python" , "R" , "C++")

Output: "R"

Loops

Loops can execute a block of code over and over until a specific condition is reached.
R has two main loop commands: while loop & for loop

While loop

A set of statements are executed repeatedly as long as the condition is TRUE

Syntax :

```
while (boolean_expression) {
    statements
}
```

Ex3

```
value <- 1
while ( value < 5 ){
  cat (value , "\n")
  value <- value + 1
}
```

output :

```
1
2
3
4
```

for loop

It is used for iterating over a sequence. It executes a block of code for a specific number of iterations

Syntax :

```
for (value in (list/vector) {
        statements
}
```

Example :

```
value <- 1 : 4
for(val in value) {
    cat (value , "\n")
}
```

output :

```
1
2
3
4
```

R Functions

A function is a block of code that only runs when it is called. Use the keyword function() to create a function.

A function can be divided into two parts.

1. Function Definition
2. Function Call

- Function name
- List of parameters
- Function body
- Return value

Syntax:

```
Function_name <- function (<parameters>) {

    Function body
    :

    return (variable*)
}
```

ex: A function to add two numbers and return the sum

```
addTwo <- function (a, b) {
    sum_two = a + b
    return (sum_two)
}
```

Function Call

A function when defined does not execute on its own and sits idle unless called.

Syntax:

```
function name (arguments)
        or
variable <- function name (arguments)
```

Ex: function call for the above function definition

```
num_1 <- 10
num_2 <- 20
num_3 <- addTwo (num_1, num_2)
```