# Basic of R Programming Language

R is a popular programming language used for statistical computing and graphical presentation. Its most common use is to analyze and visualize data.

## Why Use R?

- It is a great resource for data analysis, data visualization, data science and machine learning
- It provides many statistical techniques (such as statistical tests, classification, clustering and data reduction)
- It is easy to draw graphs in R, like pie charts, histograms, box plot, scatter plot, etc...
- It works on different platforms (Windows, Mac, Linux)
- It is open-source and free
- It has a large community support
- It has many packages (libraries of functions) that can be used to solve different problems

## R & RStudio Setup on Windows

**Watch this Video By Click on it 👇🏻**

## Get Started

```
# Start With "Hello World"
print("Hello World!")
```

```
[1] "Hello World!"
```

```
name <- "Janak Singh"
print(name)
```

```
[1] "Janak Singh"
```

```
add = "KTM"
```

```
cat("Hello Universe!😂")
```

```
Hello Universe!😂
```

```
cat(name)
```

```
Janak Singh
```

```
paste("Hello, Elon Musk!😊")
```

```
'Hello, Elon Musk!😊'
```

```
paste("Hello", "World", sep=",")
```

```
'Hello,World'
```

## User Input

```
my_name <- readline(prompt = "What is Your Name?")
class(my_name)

age <- readline(prompt="What is Your age?")
my_age <- as.integer(age)
class(my_age)
```

```
What is Your Name?Janak SIngh DHami
'character'
What is Your age?20
'integer'
```

```
r <- 5
```

```
name <- readline()
```

```
Ram Shah
```

```
print(name)
```

```
[1] "Ram Shah"
```

```
n <- readline(prompt = "What is your Name?")
paste("Your name is", n)
```

```
What is your Name?Janak Singh dhami
'Your name is Janak Singh dhami'
```

## Comments in R

---

Comments can be used to explain R code, and to make it more readable.

Comments starts with a #. When executing code, R will ignore anything that starts with #.

```
# This is Comment
print("How are you?") #This is also comment 👌🏻
```

```
[1] "How are you?"
```

```
# this is comment
print("Hello Guys")

#print(name) # My name
```

```
[1] "Hello Guys"
```

## Variables in R

---

Variables in *R* are used to store data values. You can assign values to variables using the **<-** operator or the **=** operator.

---

Variable names must start with a letter and can include letters, numbers, and underscores.

**Rules for Variable Naming:**

- Must start with a letter.
- Cannot contain spaces or special characters (except _ and .).

- Case-sensitive (e.g., var and Var are different).

```
# name
# Name
# my_name
# my-name - X
# my addr = "KTM"


# case-sensitive
addr = "Nepal"
Addr = "India"

print(addr)
print(Addr)
```

```
[1] "Nepal"
[1] "India"
```

```
# Assigning values to variables

x <- 10         # Numeric value
y <- 20.5      # Decimal/float value
name <- "Alice" # Character value
isOk <- TRUE    # Logical value
com <- 3i+6   #Complex number

# Using variables
sum <- x + y

message <- paste("Name:", name, "| Sum:", sum, "| isOk:", isOk)

# Printing the result
print(message)
```

```
[1] "Name: Alice | Sum: 30.5 | isOk: TRUE"
```

- The ***paste()*** function concatenates strings.
- The ***print()*** function displays the output.

```
a <- 55L
b <- 55.5
hlo <- "Hii"
is <- TRUE
com <- 3i +7

class(a)
```

```
class(b)
class(hlo)
class(is)
class(com)
```

```
'integer'
'numeric'
'character'
'logical'
'complex'
```

```
# check variable type
class(x)
class(y)
class(com)
```

```
'numeric'
'numeric'
'complex'
```

```
x <- 1L # integer
y <- 2 # numeric

# convert from integer to numeric:
a <- as.numeric(x)

# convert from numeric to integer:
b <- as.integer(y)
```

```
p <- 5
class(p)
```

```
'numeric'
```

```
q = as.integer(p)
class(q)
```

```
'integer'
```

**Multiple Variables**

R allows you to assign the same value to multiple variables in one line:

```
addr1 <- aad2 <- "KTM"
```

```
print(addr1)
print(aad2)
```

```
[1] "KTM"
[1] "KTM"
```

```
# Assign the same value to multiple variables in one line
var1 <- var2 <- var3 <- "Orange"

# Print variable values
var1
var2
var3
```

```
'Orange'
'Orange'
'Orange'
```

## Basic Data Types

---

Basic data types in R can be divided into the following types:

- **numeric** - (10.5, 55, 787)

> A numeric data type is the most common type in R, and contains any number with or without a decimal, like: 10.5, 55, 787

- **integer** - (1L, 55L, 100L, where the letter "L" declares this as an integer)

> Integers are numeric data without decimals. This is used when you are certain that you will never create a variable that should contain decimals. To create an integer variable, you must use the letter L after the integer value

- **complex** - (9 + 3i, where "i" is the imaginary part)

> A complex number is written with an "i" as the imaginary part

- **character** (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- **logical** (a.k.a. boolean) - (TRUE or FALSE)

---

> We can use the class() function to check the data type of a variable.

```
# numeric
x <- 10.5
class(x)

# integer
```

```
x <- 1000L
class(x)

# string/character
name <- "Janak"
class(name)

# complex
x <- 9i + 3
class(x)

# character/string
x <- "R is exciting"
class(x)

# logical/boolean
x <- TRUE
class(x)
```

```
'numeric'
'integer'
'character'
'complex'
'character'
'logical'
```

```
Name <- "Ram"
class(Name)
```

```
'character'
```

```
y <- 5i +6
class(y)
```

```
'complex'
```

## Math in R

---

In this section, we will discuss about how we use math by using arthmatic operators in R.

```
# add
5+6

# sub
55-6

# divide
```

```
58/4

# sq.
2**3

# mod
10%%3
```

```
11
49
14.5
8
1
```

```
3+5
```

```
8
```

```
55-5
```

```
50
```

```
10/5
```

```
2
```

```
2**3
```

```
8
```

```
3*4
```

```
12
```

```
x <- 55
y <- x**2

sum <- x + y

print(sum)
```

```
[1] 3080
```

**Built-in Math Functions**

```
# Max and Min
max(2, 5, 66, 4444)
min(2, 5, 66, 4444)

# Square root
sqrt(25)
```

```
4444
2
5
```

```
marks <- c(25,67, 78, 55)

first <- max(marks)
print(first)
```

```
[1] 78
```

```
fail <- min(marks)
print(fail)
```

```
[1] 25
```

```
a <- 44
sq <- sqrt(a)
print(sq)
```

```
[1] 6.63325
```

> **ceiling() and floor()**
>
> > The **ceiling()** function rounds a number upwards to its nearest integer, and the **floor()** function rounds a number downwards to its nearest integer, and returns the result:

```
ceiling(1.4)

floor(1.4)
```

```
2
1
```

```
# abs()
# The abs() function returns the absolute (positive) value of a number:
abs(-74.7)
```

```
74.7
```

```
t <- -56
```

```
abs(t)
```

```
56
```

## String in R

```
fname <- "Janak" # First name of the user
lname <- "Dhami"  # last name of the user # Corrected the assignment to l
age <- 20 #age of the user
isHealthy = TRUE #health condition of user

msg = paste("My name is", fname, lname, "& i am", age, "Years old.")

print(msg)
```

```
[1] "My name is Janak Dhami & i am 20 Years old."
```

```
first_name <- "Janak"
second_name <- "Dhami"

full_name <- paste("My name is", first_name, second_name)

print(full_name)
```

```
[1] "My name is Janak Dhami"
```

```
nchar(full_name)
```

```
22
```

```
# String Length
nchar(msg)
```

```
43
```

```
grepl("Dhami", full_name)
```

```
TRUE
```

```
# Check a String
# Use the grepl() function to check if a character or a sequence of chara
```

```
grepl("Janak", msg)
```

```
TRUE
```

```
my_name <- "Janak S,ingh D,hami"

# Split the string into individual characters
ind <- strsplit(my_name, ",")
ind[1]
```

```
1. 'Janak S' · 'ingh D' · 'hami'
```

**Escape Characters**

To insert characters that are illegal in a string, you must use an escape character.

---

| Code | Result |
|---|---|
| \\ | Backslash |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |

```
intro <- "Nepal  \"isbeautiful\" country"
cat(intro)
```

```
Nepal  "isbeautiful" country
```

```
my_intro <- "My name is Janak. from DCL."
print(my_intro)
```

```
[1] "My name is Janak. from DCL."
```

```
my_intro <- "My \b name is \t Janak. \n from DCL."
cat(my_intro)
```

```
Myname is 	Janak.
 from DCL.
```

```
# Erro Example
intro <- "Nepal is beautiful country located in "Kathmandu""
```

```
Error in parse(text = input): <text>:2:50: unexpected symbol
1: # Erro Example
2: intro <- "Nepal is beautiful country located in "Kathmandu
                                                    ^
Traceback:
```

```
# Solution
intro <- "Nepal is beautiful country located in \"Kathmandu\". \n Mt. Eve
cat(intro)
```

```
Nepal is beautiful country located in "Kathmandu".
 Mt. Everest is also located in Nepal.
```

## R Booleans / Logical Values

---

When you compare two values, the expression is evaluated and R returns the logical answer

```
10 > 9    # TRUE because 10 is greater than 9
10 == 9   # FALSE because 10 is not equal to 9
10 < 9    # FALSE because 10 is greater than 9
```

```
TRUE
FALSE
FALSE
```

```
10 == 9
```

```
FALSE
```

```
"Janak" == "janak"
```

```
FALSE
```

```
# Example
a <- 10
b <- 9

a > b

# Note: We will use these conditions in if...else.. conditional statement
```

```
TRUE
```

> ***Q&A: What is Statement?***

## Operators in R

---

Operators are used to perform operations on variables and values.

R divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Miscellaneous operators

```
a <- 55
b <- 10
```

**Arithmetic Operators:**

Arithmetic operators are used with numeric values to perform common mathematical operations

```
# 1. addition
add <- a + b

# 2. Subtraction
sub <- a - b

# 3. Division
div <- a / b

# 4. Mode
mod <- b %% a

# 5. Exponent
ex <- 4^2

cat(add, sub, div, mod, ex)
```

```
65 45 5.5 10 16
```

```
a <- 4
b <- 6
```

```
sum <-  a + b
sub <- a - b
div <- a/b
```

**Assignment Operators:**

Assignment operators are used to assign values to variables

```
my_var <- 3

my_var <<- 3 # <<- is a global assigner.

3 -> my_var

3 ->> my_var
y = 55

my_var # print my_var
```

```
3
```

```
a <- 5

55 -> num


print(a)
print(num)
```

```
[1] 5
[1] 55
```

**Comparison Operators:**

Comparison operators are used to compare two values

```
# 1. Equal to
10 == 10

#  2. Not Equal to
5 != 6

# 3. Greater than
6 > 4

# 4. less than
```

```
4 < 56

# 5. Greater than or equal to
#  >=

# 6. Less than or equal to
#  <=
```

```
TRUE
TRUE
TRUE
TRUE
```

```
"Janak" != "janak"
```

```
TRUE
```

```
55>6
```

```
TRUE
```

```
Start coding or generate with AI.
```

**Logical Operators:**

Logical operators are used to combine conditional statements

```
# 1. & and && are the Logical AND Operator
p <- 5
q <- 6

p == 5 & q == 5

# 2. | and || are the locial OR Operator
p == 5 | q == 5

# 3. ! is the logical NOT Operator
!q == 5
!p == 5
```

```
FALSE
TRUE
TRUE
FALSE
```

```
p <- 8
a <- 5
```

```
p > 6 & a < 6
```

```
TRUE
```

```
 a == 5 | p > 10
```

```
TRUE
```

```
!a == 5
```

```
FALSE
```

**Miscellaneous Operators:**

1. Miscellaneous operators are used to manipulate data

```
# : Creates a series of numbers in a sequence   x <- 1:10
num <- 1:5
print(num)

# %in%  Find out if an element belongs to a vector  x %in% y
# wait for next topic

# %*%   Matrix Multiplication   x <- Matrix1 %*% Matrix2
# wait for next topic
```

```
[1] 1 2 3 4 5
```

## Conditional Statements

---

Conditional statements let the program decide what to do based on whether something is true or false. They are used to make decisions and control what happens in your code.

**if..else if ....else... Statement**

**Syntax**

```
if (conditions){
   Positive Statements
   Positive Statements
}else if (conditions){
   Positiv statemests
}else{
```

```
    False Statements
 }
```

```
a <- readline(prompt="Enter any number:")
```

```
Enter any number:67
```

```
if(a == 5){
  print("Ohh, It's 5.")
}
```

```
if(a == 6){
  print("Ohh, It's 6.")
}else{
  print("Ohh, it's not 6.")
}
```

```
[1] "Ohh, it's not 6."
```

```
a <- readline(prompt = "Enter your fav. Number:")
if(a == 5){
  print("Ohh, It's 5.")
}else if( a==6){
  print("It's 6.")
}else if ( a == 0){
  print("It's Zero.")
}else{
  print("Sorry!")
}
```

```
Enter your fav. Number:45745754
[1] "Sorry!"
```

```
height_of_janak <- 5.10
height_of_sita <- 5.2
height_of_ram <- 6
```

```
# if conditions
if (height_of_ram >= height_of_sita){
  cat("Height of Ram is greater than Sita.")
}
```

```
Height of Ram is greater than Sita.
```

```r
# if...else... condition
if (height_of_janak >= height_of_ram){
  cat("Height of Janak is greater than Ram.")
}else{
  cat("Height of Ram is greater than Sita.")
}
```

```
Height of Ram is greater than Sita.
```

```r
# if....else if.....else... condition

if(height_of_janak >= height_of_ram){
  cat("Height of Janak is greater than Ram.")
}else if(height_of_sita >= height_of_ram){
  cat("Height of Sita is Grater than Ram")
}else{
  cat("Height of Ram is Grater than Janak and Sita.")
}
```

```
Height of Ram is Grater than Janak and Sita.
```

```r
# another example
a <- 5
b <- 6

if(a>b){
  print("Ok")
}else if(b>a){
  print("not ok")
} else{
  "ho"
}
```

```
[1] "not ok"
```

```r
age <- readline(prompt="Enter your age:")
age <- as.integer(age)
if(age >= 18){
  print("You are eligible for votting.")
}else if(age < 18){
  print("Sorry! your are not eligible for votting.")
}else{
  print("Sorry!")
}
```

```
Enter your age:-3
[1] "Sorry! your are not eligible for votting."
```

## Switch in R

A switch statement chooses one of several options to run based on the value of an input. It's like a menu where you pick a number or name, and the program performs the corresponding task.

```
switch(expression, option1, option2, ...)
```

```
a <- 3

res <- switch(a,
"Hello",
"Hii",
"Fine",
"Go"
)
res
```

```
'Fine'
```

```
# Example of switch statement
option <- 3

result <- switch(option,
  "Option 1 chosen",   # When option is 1
  "Option 2 chosen",   # When option is 2
  "Option 3 chosen"    # When option is 3
)

print(result)  # Output: "Option 2 chosen"
```

```
[1] "Option 3 chosen"
```

```
choice <- "C"

result <- switch(choice,
  A = "You chose A",
  B = "You chose B",
  C = "You chose C"
)

print(result)  # Output: "You chose B"
```

```
[1] "You chose C"
```

# Loops in R

Loops are used to repeat a block of code multiple times. They are helpful when you need to perform the same operation for multiple values or until a specific condition is met.

*R has two loop commands:*

1. **while** loops
2. **for** loops

## While Loop

A while loop keeps running as long as the condition is true.

```
while(conditions){
  statements
}
```

```
a <- 2

while(a < 6){
  print("Hello, How are you?")
  a <- a + 1
}
```

```
[1] "Hello, How are you?"
[1] "Hello, How are you?"
[1] "Hello, How are you?"
[1] "Hello, How are you?"
```

```
# while loop
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

## For Loop

With the for loop we can execute a set of statements, once for each item in a vector, array, list, etc..

```
 for(expressions){
   statements
 }
```

```
x <- 1:5
print(x)
```

```
[1] 1 2 3 4 5
```

```
for(a in x){
  print(a)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```
# for loop
for (i in 1:5){
  print("NO")
  break #Jump/exit from loop
}
```

```
[1] "NO"
```

**nested loop**

Start coding or generate with AI.

```
i <- 0
while (i < 6) {
  i <- i + 1
  if (i == 3) {
    next #skip upcoming expression without terminating the loop
  }
  print(i)
}
```

```
[1] 1
[1] 2
```

**repeat Loop**

A repeat loop runs until you use the break statement to exit it

---

```
repeat{
  statements
  if(exp.){
    break
  }
}
```

```
j <- 1

repeat{
  print("Hii")
  j <- j+1
  if(j > 3){
    break
  }
}
```

```
[1] "Hii"
[1] "Hii"
[1] "Hii"
```

```
# Example: Count to 5 using repeat
x <- 1
repeat {
  print(x)
  x <- x + 1
  if (x > 5) {
    break
  }
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

## Functions

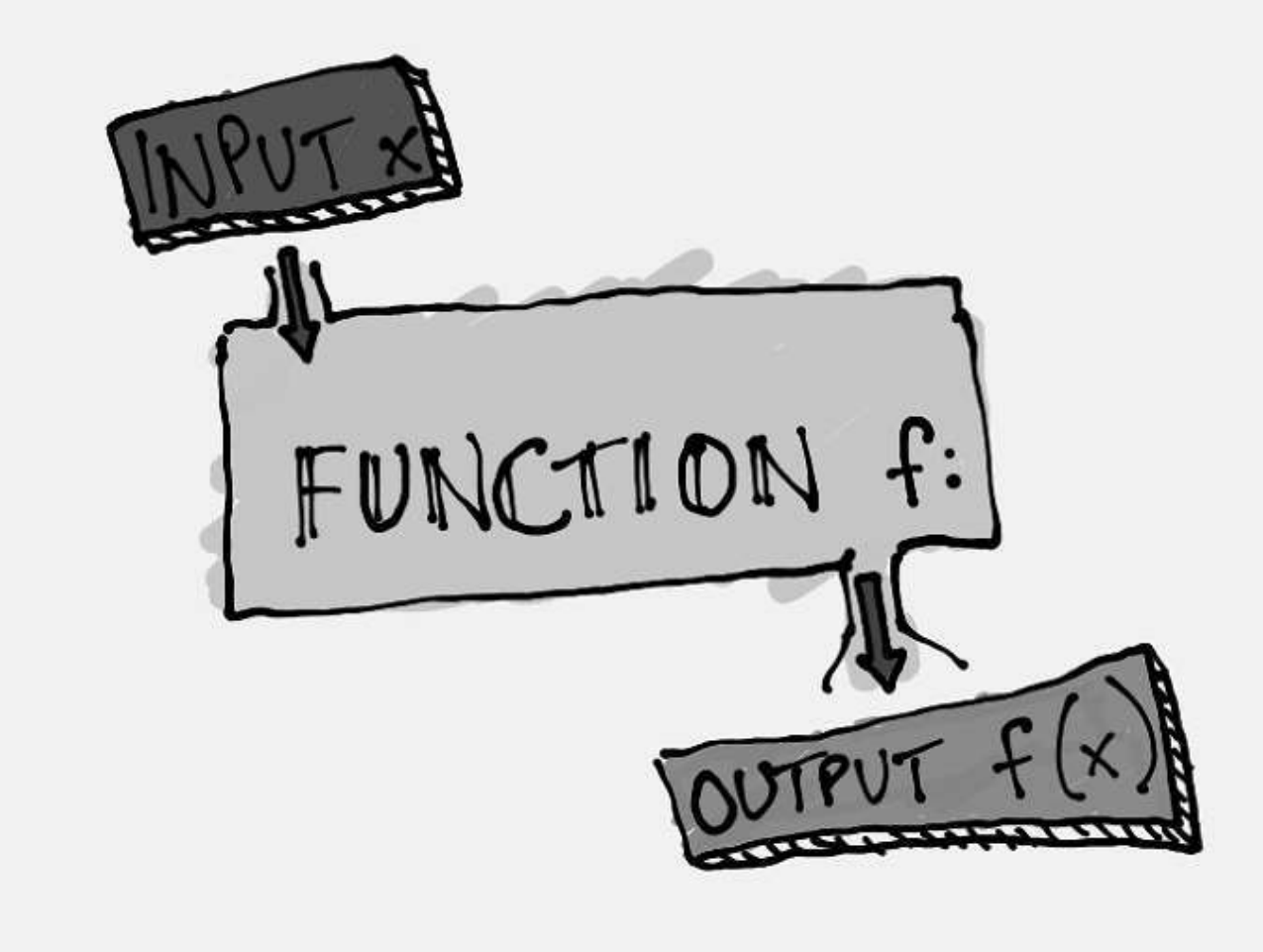
---

1. A function is a block of code which only runs when it is called.

2. You can pass data, known as parameters, into a function.

3. A function can return data as a result.

---




**Creating a Function**

Syntax:

```
# function creating
function_name <- function(para1, para2,...) {
  Statements
  return()
}

# Function calling
function_name(para1, para2,...)
```

```
add <- function(a, b){
  sum <- a + b
  sub <- a - b
  print(sum)
```

```
  print(sub)
}


add(7, 9)
```

```
[1] 16
[1] -2
```

```
# functions
my_fun <- function(x,y){
  print(x)
  print(y)
}

#Calling
my_fun(4, 6)
```

```
[1] 4
[1] 6
```

```
# function with return value
# create function with name add
add <- function(a,b){
  sum <- a + b
  # sum # This line will return the value of 'sum'
  return(sum) #This is also correct
}

# calling function

result <- add(44, 88)

cat("The sum of a and b is:", result)
```

```
The sum of a and b is: 132
```

**Note:** *In R, the last expression evaluated within a function is automatically returned.*

## Data Structures

---

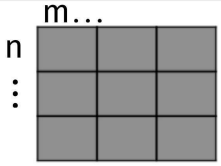
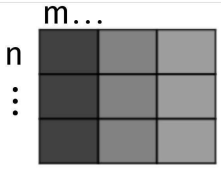
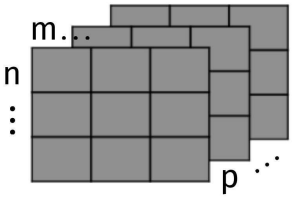
R provides several built-in data structures to store and manipulate data. Each data structure serves a specific purpose depending on the type and arrangement of the data.

---

1. Vector

- A one-dimensional array that holds elements of the same type (e.g., numeric, character, logical).
- Created using the c() function.

2. Matrix
    - A two-dimensional array with rows and columns. All elements must be of the same type.
    - Created using the matrix() function.
3. List
    - A collection of elements that can be of different types (e.g., numeric, character, logical).
    - Created using the list() function.
4. Data Frame
    - A table-like structure where each column can have different types of data.
    - Ideal for handling datasets.
5. Factors
    - Used for categorical data and can store unique levels of a variable.
    - Created using the factor() function.
6. Array
    - A multi-dimensional array that can hold elements of the same type.
    - Created using the array() function.

| | Dimensions | Mode (data "type") | Example |
|---|---|---|---|
| **Vector** |   | Identical | `c(10,0.2,34,48,53)` |
| **Matrix** |   | Identical | `matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3)` |
| **Data frame** |   | Can be different | `data.frame(x = 1:3, y = 5:7)` |
| **Array** |   | Identical | `array(data = 1:3, dim = c(2,4,2))` |
| **List** | Vector<br>Matrix<br>Data frame<br>Array | Can be different | `list(x = cars[,1], y = cars[,2])` |

## 1. Vector

---

A vector is simply a list of items that are of the same type.

To combine the list of items to a vector, use the c() function and separate the items by a comma.

**Purpose:** Used for storing homogeneous data.

```
students <- c("Janak","Ram", "Manish")
marks <- c(56, 78, 68.89)
class(students)
print(marks)
```

```
'character'
[1] 56.00 78.00 68.89
```

```
students[2]
marks[3]
```

```
'Ram'
68.89
```

**Access Vectors**

You can access the vector items by referring to its index number inside brackets []. The first item has index 1, the second item has index 2, and so on.

```
# creating vectors
names <- c("ZemB","Janak", "Ram", "Sita")
numbers<- c(1, 4, 6,2, 4, 6, 8, 9)
conditon <- c(TRUE, FALSE)

# access the element
names[4]
```

```
'Sita'
```

```
vectors <- c("janak", 5, TRUE)
vectors
```

```
'janak' · '5' · 'TRUE'
```

```
# multiple access
names[c(1, 4)]
```

```
'ZemB' · 'Sita'
```

```
names[c(4)]
```

```
'Sita'
```

```
# Access all items except for the first item
numbers[c(-1)]
```

```
4 · 6 · 2 · 4 · 6 · 8 · 9
```

```
length(students)
```

```
3
```

```
# measure length of vector
length(names)
```

```
4
```

```
# sort the vector ascending
sort(names)
```

'Janak' · 'Ram' · 'Sita' · 'ZemB'

```
# Vector with numerical values in a sequence
num <- 1.2:5
num
```

1.2 · 2.2 · 3.2 · 4.2

```
sr <- seq(from=1, to=100, by = 25)
sr
```

1 · 26 · 51 · 76

```
# To make bigger or smaller steps in a sequence, use the seq() function:
series <- seq(from = 550, to = 100, by = -20)
series
```

550 · 530 · 510 · 490 · 470 · 450 · 430 · 410 · 390 · 370 · 350 · 330 · 310 · 290 · 270 · 250 · 230 · 210 · 190 · 170 · 150 · 130 · 110

**Change individual item from vector**

```
student_names <- c("Janak", "Ramesh", "Ram", "Hari")
student_names
```

'Janak' · 'Ramesh' · 'Ram' · 'Hari'

```
student_names[1] <- "Manish"
student_names
```

'Manish' · 'Ramesh' · 'KP' · 'Hari'

```
student_names[2] <- "Krishna"
student_names
```

'Manish' · 'Krishna' · 'KP' · 'Hari'

**Repeat Vectors**

To repeat vectors, use the rep() function:

```
msg <- c("Hii!", "Hello!")
```

```
final <- rep(msg, each=2)
```

```
final
```

```
'Hii!' · 'Hii!' · 'Hello!' · 'Hello!'
```

```
test <- rep(c("Get Out", "Janak"), each=2)
cat(test)
```

```
Get Out Get Out Janak Janak
```

```
hare <- "हरे"
ram <- "राम"
krishna <- "कृष्ण"
```

```
mantra_form <- rep(c(paste(hare, ram), ram, hare, paste(hare, krishna), k
mantra_form
```

```
'हरे राम' · 'हरे राम' · 'राम' · 'राम' · 'हरे' · 'हरे' · 'हरे कृष्ण' · 'हरे कृष्ण' · 'कृष्ण' · 'कृष्ण' · 'हरे' · 'हरे'
```

```
# Repeat each value:
mantra <- rep(c("हरे राम", "राम ", "हरे", "हरे कृष्ण", "कृष्ण", "हरे"), each=2)
mantra
```

```
'हरे राम' · 'हरे राम' · 'राम ' · 'राम ' · 'हरे' · 'हरे' · 'हरे कृष्ण' · 'हरे कृष्ण' · 'कृष्ण' · 'कृष्ण' · 'हरे' · 'हरे'
```

```
# Repeat the sequence of the vector:
jap = rep(mantra, times=108)
# jap
```

```
# Repeat each value independently:
repeat_indepent <- rep(c(1,2,3), times = c(5,2,1))
repeat_indepent
```

```
1 · 1 · 1 · 1 · 1 · 2 · 2 · 3
```

## 2. Lists

---

A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable.

- To create a list, use the list() function:
- **Purpose:** Used for storing heterogeneous data.
- **Access:** Accessed using [[ ]] or $ for named elements.

```
# create lists
countries <- list("Nepal", "Bharat", "Bhutan", "Bangaladesh", "Shrilanka"
```

```
countries[1]
```

1. 'Nepal'

```
location <- list(country="Nepal", district="Kathmandu")
location$district
```

'Kathmandu'

```
student <- list(name="Janak Singh Dhami", year="2", sub=c("Math", "Chemis
student$sub[3]
```

'Physics'

```
about <- paste("My Name is", student$name, "and i\'m from", student$locat
cat(about)
```

```
My Name is Janak Singh Dhami and i'm from Kathmandu
```

```
# access list item by position
student[[1]]
```

'Janak Singh Dhami'

**Change Item Value**

```
# i want change my name
student$name <- "Janak Dhami"

student$name
```

'Janak Dhami'

```
countries[6] <- "Maldivash"

countries
```

1. 'Nepal'
2. 'Bharat'
3. 'Bhutan'
4. 'Bangaladesh'
5. 'Shrilanka'
6. 'Maldivash'

**List Length**

```
length(countries)
```

6

**Check if Item Exists**

```
"Nepal" %in% countries
```

TRUE

**Add List Items**

```
# i think, i should add pakistan in SARC countries
countries <- append(countries, "Pakistan")
countries
```

1. 'Nepal'
2. 'Bharat'
3. 'Bhutan'
4. 'Bangaladesh'
5. 'Shrilanka'
6. 'Maldivash'
7. 'Pakistan'

To add an item to the right of a specified index, add "after=index number" in the append() function:

```
countries <- append(countries, "Afganistan", after=6)
countries
```

1. 'Nepal'
2. 'Bharat'
3. 'Bhutan'
4. 'Bangaladesh'
5. 'Shrilanka'
6. 'Maldivash'
7. 'Afganistan'
8. 'Pakistan'

```
# By mistake
countries <- append(countries, "China")
```

```
countries
```

1. 'Nepal'
2. 'Bharat'
3. 'Bhutan'
4. 'Bangaladesh'
5. 'Shrilanka'
6. 'Maldivash'
7. 'Afganistan'
8. 'Pakistan'
9. 'China'

**Remove List Items**

```
# 🧐Humm, China is not SARC country....I need to remove this one from our countries <- countries[-9] # 9 is index number of China
countries
```

1. 'Nepal'
2. 'Bharat'
3. 'Bhutan'
4. 'Bangaladesh'
5. 'Shrilanka'
6. 'Maldivash'
7. 'Afganistan'
8. 'Pakistan'

**Range of Indexes**

You can specify a range of indexes by specifying where to start and where to end the range, by using the : operator

```
# i think i need to create another hindu countries list
hindu_countries <- (countries)[1:2]

hindu_countries
```

1. 'Nepal'
2. 'Bharat'

**Join Two Lists**

There are several ways to join, or concatenate, two or more lists in R.

The most common way is to use the c() function, which combines two elements together:

```
# 🧐Humm, i should also create muslim countries list..... Okay👌 I CAN
first_list <- (countries)[4:4]
second_list <- (countries)[6:8]

muslim_countries <- c(first_list, second_list)
muslim_countries
```

1. 'Bangaladesh'
2. 'Maldivash'
3. 'Afganistan'
4. 'Pakistan'

**Access List through For Loop**

```
for (i in muslim_countries){
  print(i)
}
```

```
[1] "Bangaladesh"
[1] "Maldivash"
[1] "Afganistan"
[1] "Pakistan"
```

## 3. Matrices

---

A matrix is a two dimensional data set with columns and rows.

A column is a vertical representation of data, while a row is a horizontal representation of data.

A matrix can be created with the matrix() function. Specify the nrow and ncol parameters to get the amount of rows and columns

```
my_matrix <- matrix(c(2, 5, 6, 7, 8, 6), nrow=3, ncol=2)

print(my_matrix)
```

```
     [,1] [,2]
[1,]    2    7
[2,]    5    8
[3,]    6    6
```

```
first <- matrix(c('A', 'B', 'C', 'D'), nrow = 2, ncol = 2)
print(first)
```

```
     [,1] [,2]
[1,] "A"  "C"
[2,] "B"  "D"
```

**Access Matrix Item**

You can access the items by using [ ] brackets. The first number "1" in the bracket specifies the row-position, while the second number "2" specifies the column-position

```
my_matrix[1, 2]
```

```
7
```

*The whole row can be accessed if you specify a comma after the number in the bracket*

```
my_matrix[1,]
```

```
2 7
```

*The whole column can be accessed if you specify a comma before the number in the bracket*

```
my_matrix[,2]
```

```
7 8 6
```

**Access More Than One Row**

```
country <- matrix(c("Nepal", "India", "China", "Pakistan", "USA", "UK"),

print(country)
```

```
     [,1]    [,2]
[1,] "Nepal" "Pakistan"
[2,] "India" "USA"
[3,] "China" "UK"
```

```
# c(1,2) is a vector that tells R to select the 1st and 2nd rows.
# , after c(1,2) means all columns.
country[c(1, 2),]
```

A matrix: 2 × 2 of type chr

| | |
|---|---|
| Nepal | Pakistan |
| India | USA |

**Access More Than One Column**

More than one column can be accessed if you use the c() function:

```
country[,c(1,2)]
# , before c(1,2) means all rows.
```

A matrix: 3 × 2 of type chr

| | |
|---|---|
| Nepal | Pakistan |
| India | USA |
| China | UK |

### Add Rows and Columns

Use the `cbind()` function to add additional columns in a Matrix.

*Note: The cells in the new column must be of the same length as the existing matrix.*

```
print(country)
```

```
     [,1]    [,2]
[1,] "Nepal" "Pakistan"
[2,] "India" "USA"
[3,] "China" "UK"
```

```
new_country <- cbind(country, c("Japan", "UAE", "Korea"))
print(new_country)
```

```
     [,1]    [,2]       [,3]
[1,] "Nepal" "Pakistan" "Japan"
[2,] "India" "USA"      "UAE"
[3,] "China" "UK"       "Korea"
```

Use the rbind() function to add additional rows in a Matrix

Note: The cells in the new row must be of the same length as the existing matrix.

```
new_new_country <- rbind(new_country, c("Russia", "Afganistan", "Banglade
```

```
print(new_new_country)
```

```
     [,1]     [,2]         [,3]
[1,] "Nepal"  "Pakistan"   "Japan"
[2,] "India"  "USA"        "UAE"
[3,] "China"  "UK"         "Korea"
[4,] "Russia" "Afganistan" "Bangladesh"
```

### Remove Rows and Columns

Use the c() function to remove rows and columns in a Matrix

```
#Remove the 3rd row and the 3rd column

pre <- new_new_country[-c(3), -c(3)]
print(pre)
```

```
     [,1]     [,2]
[1,] "Nepal"  "Pakistan"
[2,] "India"  "USA"
[3,] "Russia" "Afganistan"
```

### Check if an Item Exists

To find out if a specified item is present in a matrix, use the `%in%` operator.

```
"Nepal" %in% pre
```

```
TRUE
```

### Number of Rows and Columns

Use the `dim()` function to find the number of rows and columns in a Matrix.

```
dim(pre)
```

```
3 · 2
```

### Matrix Length

Use the `length()` function to find the dimension of a Matrix.

```
length(pre)
```

```
6
```

### Loop Through a Matrix

You can loop through a Matrix using a `for` loop. The loop will start at the first row, moving right

```
for (rows in 1:nrow(pre)) {
  for (columns in 1:ncol(pre)) {
    print(pre[rows, columns])
  }
}
```

```
[1] "Nepal"
[1] "Pakistan"
[1] "India"
```

```
[1] "USA"
[1] "Russia"
[1] "Afganistan"
```

**Combine two Matrices**

Again, you can use the `rbind()` or `cbind()` function to combine two or more matrices together

```
# Combine matrices
Matrix1 <- matrix(c("apple", "banana", "cherry", "grape"), nrow = 2, ncol
Matrix2 <- matrix(c("orange", "mango", "pineapple", "watermelon"), nrow =

print(Matrix1)


print(Matrix2)

# Adding it as a rows
Matrix_Combined <- rbind(Matrix1, Matrix2)
print(Matrix_Combined)

# Adding it as a columns
Matrix_Combined <- cbind(Matrix1, Matrix2)
print(Matrix_Combined)
```

```
     [,1]     [,2]
[1,] "apple"  "cherry"
[2,] "banana" "grape"
     [,1]     [,2]
[1,] "orange" "pineapple"
[2,] "mango"  "watermelon"
     [,1]     [,2]
[1,] "apple"  "cherry"
[2,] "banana" "grape"
[3,] "orange" "pineapple"
[4,] "mango"  "watermelon"
     [,1]     [,2]     [,3]     [,4]
[1,] "apple"  "cherry" "orange" "pineapple"
[2,] "banana" "grape"  "mango"  "watermelon"
```

## 3. Arrays

---

Compared to matrices, arrays can have more than two dimensions.

We can use the array() function to create an array, and the dim parameter to specify the dimensions

```
thisarray <- c(1:12)
multiarray <- array(thisarray, dim = c(2, 2, 2))

print(multiarray)
```

```
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

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

```
print(multiarray[2,2,2])
```

```
[1] 8
```

You can also access the whole row or column from a matrix in an array, by using the c() function

```
# access all items from the 1st row  c(row number),, matrix
print(multiarray[c(1),, 2])
```

```
[1] 5 7
```

```
# Access all the items from the first column from matrix one | c(column),
print(multiarray[, c(2), 1])
```

```
[1] 3 4
```

**Check if an Item Exists**

```
2 %in% multiarray
```

```
TRUE
```

**Amount of Rows and Columns**

```
dim(multiarray)
```

```
2·2·2
```

```
length(multiarray)
```

```
8
```

```
for(x in multiarray){
  print(x)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
```

## 4. Data Frame

- Data Frames are data displayed in a format as a table.
- Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.
- Use the `data.frame()` function to create a data frame:

```
# create dataframe
df <- data.frame(
  Names = c("janak", "Ram", "Sita", "Shiv"),
  Addr = c("dcl", "india", "mithila", "kailash"),
  isFine = c(TRUE, TRUE, FALSE, FALSE),
  Marks = c(67, 55, 89, 98)
)
print(df)
```

```
  Names    Addr isFine Marks
1 janak     dcl   TRUE    67
2   Ram   india   TRUE    55
3  Sita mithila  FALSE    89
4  Shiv kailash  FALSE    98
```

**Summarize the Data**

```
summary(df)
```

```
   Names               Addr              isFine            Marks      
 Length:4           Length:4           Mode :logical   Min.   :55.00  
 Class :character   Class :character   FALSE:2         1st Qu.:64.00  
 Mode  :character   Mode  :character   TRUE :2         Median :78.00  
                                                       Mean   :77.25  
                                                       3rd Qu.:91.25  
                                                       Max.   :98.00  
```

```
# demo data
car_info <- data.frame(mtcars)
```

```
summary(car_info)
```

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

### Access Items

We can use single brackets [ ], double brackets [[ ]] or $ to access columns from a data frame.

```
df[1]
```

A data.frame: 4 × 1

| Names |
|---|
| <chr> |
| janak |
| Ram |
| Sita |
| Shiv |

```
df[["Marks"]]
```

67 · 55 · 89 · 98

```
df[["Names"]][3]
```

'Sita'

```
df$Addr
```

'dcl' · 'india' · 'mithila' · 'kailash'

```
# access specific cols (2nd and 3rd)
nndf <- df[2:3]
nndf
```

A data.frame: 4 × 2

| Addr | isFine |
|---|---|
| <chr> | <lgl> |
| dcl | TRUE |
| india | TRUE |
| mithila | FALSE |
| kailash | FALSE |

### Add Rows

Use the `rbind()` function to add new rows in a Data Frame.

```
df
```

A data.frame: 4 × 4

| Names | Addr | isFine | Marks |
|---|---|---|---|
| <chr> | <chr> | <lgl> | <dbl> |
| janak | dcl | TRUE | 67 |
| Ram | india | TRUE | 55 |
| Sita | mithila | FALSE | 89 |
| Shiv | kailash | FALSE | 98 |

```
new_df <- rbind(df, c("Shyan", "UP", FALSE, 99))
new_df
```

A data.frame: 5 × 4

| Names | Addr | isFine | Marks |
|---|---|---|---|
| <chr> | <chr> | <chr> | <chr> |
| janak | dcl | TRUE | 67 |
| Ram | india | TRUE | 55 |
| Sita | mithila | FALSE | 89 |
| Shiv | kailash | FALSE | 98 |
| Shyan | UP | FALSE | 99 |

### Add Columns

Use the `cbind()` function to add new columns in a Data Frame

```
df
```

A data.frame: 4 × 4

| Names | Addr | isFine | Marks |
|---|---|---|---|
| <chr> | <chr> | <lgl> | <dbl> |
| janak | dcl | TRUE | 67 |
| Ram | india | TRUE | 55 |
| Sita | mithila | FALSE | 89 |
| Shiv | kailash | FALSE | 98 |

```
ddf <- cbind(df, Country = c("Nepal", "India", "Nepal", "India"))
ddf
```

A data.frame: 4 × 5

| Names | Addr | isFine | Marks | Country |
|---|---|---|---|---|
| <chr> | <chr> | <lgl> | <dbl> | <chr> |
| janak | dcl | TRUE | 67 | Nepal |
| Ram | india | TRUE | 55 | India |
| Sita | mithila | FALSE | 89 | Nepal |
| Shiv | kailash | FALSE | 98 | India |

**Remove Rows and Columns**

Use the `c()` function to remove rows and columns in a Data Frame

```
df
```

A data.frame: 4 × 4

| Names | Addr | isFine | Marks |
|---|---|---|---|
| <chr> | <chr> | <lgl> | <dbl> |
| janak | dcl | TRUE | 67 |
| Ram | india | TRUE | 55 |
| Sita | mithila | FALSE | 89 |
| Shiv | kailash | FALSE | 98 |

```
# remove first row
r_df <- df[-c(1), ]
r_df
```

A data.frame: 3 × 4

| | Names | Addr | isFine | Marks |
|---|---|---|---|---|
| | <chr> | <chr> | <lgl> | <dbl> |
| **2** | Ram | india | TRUE | 55 |
| **3** | Sita | mithila | FALSE | 89 |
| **4** | Shiv | kailash | FALSE | 98 |

```
# remove second col
c_df <- df[,-c(2)]
c_df
```

A data.frame: 4 × 3

| Names | isFine | Marks |
|---|---|---|
| <chr> | <lgl> | <dbl> |
| janak | TRUE | 67 |
| Ram | TRUE | 55 |
| Sita | FALSE | 89 |
| Shiv | FALSE | 98 |

Amount of Rows and Columns

```
dim(df)
```

```
4 · 4
```

```
length(df)
```

```
4
```

```
# no of column
ncol(df)
```

```
4
```

```
# no of rows
nrow(df)
```

```
4
```

**Combining Data Frames**

- Use the rbind() function to combine two or more data frames in R vertically
- And use the cbind() function to combine two or more data frames in R horizontally

**Structure Data**

```
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
```

```
$ qsec: num  16.5 17 18.6 19.4 17 ...
$ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
$ am  : num  1 1 1 0 0 0 0 0 0 0 ...
$ gear: num  4 4 4 3 3 3 3 4 4 4 ...
$ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

## 5. Factors

Factors are used to categorize data. Examples of factors are:

- Gender: Male/Female
- Music: Rock, Pop, Classic, Rap

To create a factor, use the `factor()` function and add a vector as argument

```
country <- c("Nepal", "India", "Nepal","Nepal", "Pakistan")
```

```
c_factor <- factor(country)
c_factor
```

```
Nepal · India · Nepal · Nepal · Pakistan
▶ Levels:
```

```
# we can see/access levels of this factor
levels(c_factor)
```

```
'India' · 'Nepal' · 'Pakistan'
```

```
is.factor(country)
```

```
FALSE
```

```
is.factor(c_factor)
```

```
TRUE
```