**R Vectors**

A vector is the basic data structure in R that stores data of similar types. In R, we use the c() function to create a vector. For example,

**# create vector of string types**

employees <- c("Sabby", "Cathy", "Lucy")

print(employees)

# Output: [1] "Sabby" "Cathy" "Lucy"

**Access Vector Elements in R**

In R, each element in a vector is associated with a number. The number is known as a vector index.

We can access elements of a vector using the index number (1, 2, 3 …). For example, **# a vector of string type**

languages <- c("Swift", "Java", "R")

**# access first element of languages**

print(languages[1]) # "Swift"

**# access third element of languages**

print(languages[3]). # "R"

**Modify Vector Element**

To change a vector element, we can simply reassign a new value to the specific index. For example,

dailyActivities <- c("Eat","Repeat")

cat("Initial Vector:", dailyActivities)

**# change element at index 2**

dailyActivities[2] <- "Sleep"

print(dailyActivities)

**Output**

Initial Vector: Eat Repeat

Updated Vector: Eat Sleep

**# a vector with number sequence from 1 to 5**

numbers <- 1:5

print(numbers)

**Output**

[1] 1 2 3 4 5

**Repeat Vectors in R**

In R, we use the rep() function to repeat elements of vectors. For example, **# repeat sequence of vector 2 times**

numbers <- rep(c(2,4,6), times = 2)

print("Using times argument:", numbers)

**Output**

Using times argument: 2 4 6 2 4 6

**# repeat each element of vector 2 times**

numbers <- rep(c(2,4,6), each = 2)

Print("\nUsing each argument:", numbers)

**Output**

Using each argument: 2 2 4 4 6 6

**Loop Over a R Vector**

numbers <- c(1, 2, 3, 4, 5)

for (number in numbers) {

print(number)

}

**Output**

[1] 1

[1] 2

[1] 3

[1] 4

[1] 5

**Length of Vector in R**

We can use the length() function to find the number of elements present inside the vector. For example,

languages <- c("R", "Swift", "Python", "Java")

print("Total Elements:", length(languages))

**Output**

Total Elements: 4

**R MATRIX**

A matrix is a two-dimensional data structure where data are arranged into rows and columns. For example,

**Create a Matrix in R**

In R, we use the matrix() function to create a matrix.

**# create a 2 by 3 matrix**

matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE) print(matrix1)

**# access element at 1st row, 2nd column**

print("\nDesired Element:", matrix1[1, 2])

**Output**

**[,1] [,2]**

**[1,] "Sabby" "Larry"**

**[2,] "Cathy" "Harry"**

**Access Entire Row or Column**

In R, we can also access the entire row or column based on the value passed inside []. [n, ] - returns the entire element of the nth row.

[ ,n] - returns the entire element of the nth column.

For example,

matrix1 <- matrix(c("Sabby", "Cathy", "Larry", "Harry"), nrow = 2, ncol = 2) print(matrix1)

**# access entire element at 1st row**

print("\n1st Row:", matrix1[1, ])

**# access entire element at 2nd column**

print("\n2nd Column:", matrix1[, 2])

**Output**

[,1] [,2]

[1,] "Sabby" "Larry"

[2,] "Cathy" "Harry"

**# access entire element of 1st and 3rd row** cat("\n1st and 2nd Row:", matrix1[c(1,3), ])

**# access entire element of 2nd and 3rd column** cat("\n2nd and 3rd Column:", matrix1[ ,c(2,3)]) **Modify Matrix Element in R**

matrix1[1,2] = 140

print(matrix1)

**Combine Two Matrices in R**,

**# create two 2 by 2 matrices**

even\_numbers <- matrix(c(2, 4, 6, 8), nrow = 2, ncol = 2) odd\_numbers <- matrix(c(1, 3, 5, 7), nrow = 2, ncol = 2) **# combine two matrices by column**

total1 <- cbind(even\_numbers, odd\_numbers) print(total1)

**# combine two matrices by row**

total2 <- rbind(even\_numbers, odd\_numbers) print(total2)

**Check if Element Exists in R Matrix**

matrix1 <- matrix(c("Sabby", "Cathy", "Larry", "Harry"), nrow = 2, ncol = 2) "Larry" %in% matrix1 # TRUE

"Kinsley" %in% matrix1 # FALSE

**R FACTORS**

A Factor is a data structure that is used to work with categorizable datas.

**Create a Factor in R**

students\_gender <- factor(c("male", "female", "male", "transgender", "female")) # print the marital\_status factor

print(students\_gender)

**# access 1st element of students\_gender**

print(students\_gender[1])

**# access 4th element of students\_gender**

print(students\_gender[4])

**Modify Factor Element**

marital\_status[1] <- "divorced"

print(marital\_status[1])

**DATA FRAME**

# Create a data frame

dataframe1 <- data.frame ( Name = c("Juan", "Alcaraz", "Simantha"), Age = c(22, 15, 19),

Vote = c(TRUE, FALSE, TRUE))

print(dataframe1)

**#Accessing Element in the DataFrame**

**# pass index number inside [ ]**

print(dataframe1[1])

**# pass column name inside [[ ]]**

print(dataframe1[["Name"]])

**# use $ operator and column name**

print(dataframe1$Name)

**Combing two columns**

**# create a data frame**

dataframe1 <- data.frame (

Name = c("Juan", "Alcaraz"),

Age = c(22, 15))

**# create another data frame**

dataframe2 <- data.frame ( Name = c("Yiruma", "Bach"), Age = c(46, 89))

**# combine two data frames vertically**

updated <- rbind(dataframe1, dataframe2) print(updated)

**# combine two data frames horizontally** updated <- cbind(dataframe1, dataframe2) print(updated)

**R LIST**

A List is a collection of similar or different types of data.

**# list with similar type of data**

list1 <- list(24, 29, 32, 34)

# list with different type of data

list2 <- list("Ranjy", 38, TRUE)

**Access List Elements in R**

list1 <- list(24, "Sabby", 5.4, "Nepal")

**# access 1st item**

print(list1[1]) # 24

**# access 4th item**

print(list1[4]) # Nepal

**Modify a List Element in R**

list1 <- list(24, "Sabby", 5.4, "Nepal") # change element at index 2

list1[2] <- "Cathy"

# print updated list

print(list1)

**Add Items to R List**

list1 <- list(24, "Sabby", 5.4, "Nepal") # using append() function

append(list1, 3.14)

**Remove Items From a List in R** [-1] - removes 1st item

[-2] - removes 2nd item and so on. list1 <- list(24, "Sabby", 5.4, "Nepal") **# remove 4th item**

print(list1[-4]) # Nepal

**Length of R List**

list1 <- list(24, "Sabby", 5.4, "Nepal") # find total elements in list1 using length() Print("Total Elements:", length(list1))

**Loop Over a List**

items <- list(24, "Sabby", 5.4, "Nepal")

# iterate through each elements of numbers

for (item in items) {

print(item)

}

**Check if Element Exists in R List**

list1 <- list(24, "Sabby", 5.4, "Nepal")

"Sabby" %in% list1 # TRUE

"Kinsley" %in% list1 # FALSE

**R ARRAY**

An Array is a data structure which can store data of the same type in more than two dimensions.

**Create an Array in R**

array(vector, dim = c(nrow, ncol, nmat)

**# create two 2 by 3 matrix**

array1 <- array(c(1:12), dim = c(2,3,2))

print(array1)

**Access Array Elements**

array[n1, n2, mat\_level]

Here,

n1 - specifies the row position

n2 - specifies the column position

mat\_level - specifies the matrix level

**# create two 2 by 3 matrix**

array1 <- array(c(1:12), dim = c(2,3,2))

print(array1)

**# access element at 1st row, 3rd column of 2nd matrix** print("\nDesired Element:", array1[1, 3, 2])

**Access Entire Row or Column**

**# create a two 2 by 3 matrix**

array1 <- array(c(1:12), dim = c(2,3,2))

print(array1)

**# access entire elements at 2nd column of 1st matrix** print ("\n2nd Column Elements of 1st matrix:", array1[,c(2),1]) **# access entire elements at 1st row of 2nd matrix** print ("\n1st Row Elements of 2nd Matrix:", array1[c(1), ,2])

**Check if Element Exists**

# create a two 2 by 3 matrix

array1 <- array(c(1:12), dim = c(2,3,2))

11 %in% array1 # TRUE

13 %in% array1 # FALSE

**Length of Array in R**

# create a two 2 by 3 matrix

array1 <- array(c(1:12), dim = c(2,3,2))

# find total elements in array1 using length()

print("Total Elements:", length(array1))

**//LOOPING STATEMENTS**

**WRITE A PROGRAM TO IMPLEMENT USING FOR LOOP** numbers = c(2, 3, 12, 14, 5, 19, 23, 64)

# for loop with break

for (i in numbers) {

# break the loop if number is 5

if( i == 5) {

break

}

print(i)

}

**WRITE A PROGRAM TO COUNT THE NUMBER OF EVEN NUMBERS** # vector of numbers

num = c(2, 3, 12, 14, 5, 19, 23, 64)

# variable to store the count of even numbers

count = 0

# for loop to count even numbers

for (i in num) {

# check if i is even

if (i %% 2 == 0) {

count = count + 1

}

}

print(count)

**WRITE A PROGRAM TO CALCULATE THE SUM OF THE FIRST TEN NATURAL NUMBERS USING WHILE LOOP.**

# variable to store current number

number = 1

# variable to store current sum

sum = 0

# while loop to calculate sum

while(number <= 10) {

sum = sum + number

number = number + 1

}

print(sum)

|  |
| --- |

The syntax of repeat loop is:

| repeat { # statements  if(stop\_condition) {  break  }  } |
| --- |

**WRITE A PROGRM TO PRINT NUMBERS USING A REPEAT LOOP AND WILL EXECUTE UNTIL THE BREAK STATEMENT IS EXECUTED.**

x = 1

# Repeat loop

repeat {

print(x)

# Break statement to terminate if x > 4

if (x > 4) {

break

}

# Increment x by 1

x = x + 1

}

**WRTIE A PROGRAM TO CHECK NEGATIVE AND POSITIVE NUMBER** x <- 0

**# check if x is positive or negative or zero**

if (x > 0) {

print("x is a positive number")

} else if (x < 0) {

print("x is a negative number")

} else {

print("x is zero")

}

**// DATA ANALYSIS USING IRIS DATASET**

data(iris)

str(iris)

summary(iris)

mean(iris $ Sepal.Length)

median(iris $ Sepal.Width)

sd(iris$Sepal.Width)

var(iris$Sepal.Width)

summary(a)

SL=iris $ Sepal.Length

hist(SL)

**//WRITE A PROGRAM TO IMPLEMENT HIERARCHICAL CLUSTERING** ds<- USArrests

# Dissimilarity matrix

d <- dist(ds, method = "euclidean")

# Hierarchical clustering using Complete Linkage

hcl <- hclust(d, method = "complete" )

# Plot the obtained dendrogram

plot(hcl, cex = 0.8, hang = -1)

# Cut tree into 4 groups

sub\_grp <- cutree(hcl, k = 4)

# Number of members in each cluster

table(sub\_grp)

#draw the dendrogram with a border around the 4 clusters.

rect.hclust(hcl, k=4, border = 7:10)

**//WRITE A PROGRAM TO IMPLEMENT K-MEANS CLUSTERING ALGORITHM** ds<-iris[,1:4]

summary(ds)

km.res<-kmeans(ds,3 ,nstart=50)

km.res

table(iris$Species, km.res$cluster)

plot(ds[c("Sepal.Length", "Sepal.Width")], col = km.res$cluster, main = " K-means with 3 clusters")

points(km.res$centers[, c("Sepal.Length", "Sepal.Width")], col = 4, pch = 8, cex = 3) install.packages("factoextra")

library(factoextra)

#Finding optimal no of cluster

fviz\_nbclust(ds, km.res, method = 'wss')

#Visualizing K-means Clusters

fviz\_cluster(km.res, data = ds)