**//List Example**

var listData = mutableListOf("Abhishek", "Rai", "CK", "PK", "DK")

listData.add("Sandeep Singh")

**//Set Example**

var setData = mutableSetOf("Ck", "PK", "Dk", "CK")

**//Map Example**

var map = mutableMapOf<String, Int>("Name" to 1, "Address" to 100)

map.put("phoneNO", 24243)

**//ForEach loop on list**

listData.forEach{ e ->

println("forEach Iteams is $e")

}

**//For loop on list**

for(items in listData){

println("For loop Items is $items")

}

**//Iterate items from list using Iterator**

var iteratorData: Iterator<String> = listData.iterator()

while(iteratorData.hasNext()){

println("Iterator Next Data items is ${iteratorData.next()}")

}

// **Iterate items from map**

**for(key in map.keys){**

println("Items key= $key and value is ${map[key]}}")

}

**//Deblicate Element find or remove from the list**

val concrete = listOf(1, 3, 5, 7, 2, 8, 2, 1);

var uniqList = mutableListOf<Int>();

for(element in concrete){

if(uniqList.contains(element)){

println("Dublicate element found: $element")

}else{

uniqList.add(element)

}

}

**OR**

val concrete2 = listOf(1, 3, 5, 7, 2, 8, 2, 1);

var uniqList2 = mutableListOf<Int>();

for(elementIndex in 0.. concrete2.size-1){

if(uniqList2.contains(concrete2 [elementIndex])){

println("Dublicate element found: ${ concrete2 [elementIndex]}")

}else{

uniqList2.add(concrete2 [elementIndex])

}

}

//**Given number is odd or Even**

var no1 = 12

if(no1 % 2 == 0){

println("Given number is even")

}else{

println("Given number is odd")

}

//**Given number is Prime or not**

var no2 = 43

var isPrime = true;

for(i in 2..<no2){

if(no2 % i == 0){

isPrime = false

println("Multiplication found $i and $no2")

break;

}

}

if(isPrime){

println("Given number $no2 is prime")

}else{

println("Given number $no2 is not prime")

}

//**Fectorial of Given number**

val no3 = 10;

var factorialResult = 1

for(item in 1..no3){

factorialResult = factorialResult \* item

}

println("factorialResult of $no3 is $factorialResult")

//**Reverse the given number**

var no4 = 124

var temp = no4

var revNo = 0

while(temp != 0){

val rem = temp % 10

revNo = revNo \* 10 + rem

temp = temp / 10;

}

println("Actual No is: $no4 and revNo is: $revNo")

//**Reverse the given number**

var no5 = 121

var temp5 = no5

var revNo5 = 0

while(temp5 != 0){

val rem = temp5 % 10

revNo5 = revNo5 \* 10 + rem

temp5 = temp5 / 10;

}

if(no5 == revNo5){

println("Given No is: $no5 palindrome")

}else{

println("Given No is: $no5 is not palindrome")

}

//**Given String is palindrome or not**

val strVal = "RADAR"

var reverseStr = ""

for (i in strVal.length - 1 downTo 0) {

reverseStr += strVal[i]

}

if(strVal == reverseStr){

println("Given String is: $strVal palindrome")

}else{

println("Given String is: $strVal is not palindrome")

}

//Reverse the String

val str1 = "Abhishek"

val revstr = StringBuilder(str1).reverse().toString()

println("Reverse of $str1 String is $revstr")

//Some of Natural number: all o....n(1, 2, 3, 4, ...100) is natural number

val no6 = 100

var sum = 0

for(i in 1..no6){

sum += i;

}

println("Sum of natural number: $no6 is $sum")

//Deblicate Element find or remove from the list

val concrete = listOf(1, 3, 5, 7, 2, 8, 2, 1);

var uniqList = mutableListOf<Int>();

for(element in concrete){

if(uniqList.contains(element)){

println("Dublicate element found: $element")

}else{

uniqList.add(element)

}

}

var uniqList2 = mutableListOf<Int>();

for(elementIndex in 0..concrete.size-1){

if(uniqList2.contains(concrete[elementIndex])){

println("Dublicate element found: ${concrete[elementIndex]}")

}else{

uniqList2.add(concrete[elementIndex])

}

}

val arrr :MutableList<Int> = mutableListOf(3,76,8,5,7,9,2)

val sizex = arrr.size -1

for (i in 0..sizex){

var lowstIndex = i;

for (j in i..sizex){

if(arrr[j] < arrr[lowstIndex]){

lowstIndex = j

}

}

val temp = arrr[i];

arrr[i] = arrr[lowstIndex]

arrr[lowstIndex] = temp

}

println("sortest list ${arrr}")

println("second Lowest ${arrr[1]}")

println("Second highest Items is ${arrr[sizex-1]}")

**open class SuperParrent{**

var Address = "Abhishek" //Can not be access from chil class

open var phoneNo = "9988766" // will be accessable from child class

//will be accessable from child class

open fun addOne(){

println("SuperParrent One")

}

}

**open class Parrent: SuperParrent(){**

override var phoneNo = "29694"

open var name = "Abhishek"

override fun addOne(){

println("Parrent One")

}

open fun addTwo(){

println("Parrent two")

}

}

**class Child: Parrent(){**

override var name = "CK"

override fun addTwo(){

println("Child two")

}

}

**abstract class MyAbtract{**

abstract fun addTwoNo()

fun MyAbtract(){

println("ChildAbtract class addThreeNo")

}

}

**class ChildAbtract: MyAbtract(){**

override fun addTwoNo(){

println("ChildAbtract class addTwoNo")

}

}

**interface MyInterface {**

//Mandatory method to override in child class

fun bar()

//Option method to override in child class

fun foo() {

// optional body

println("MyInterface class bar")

}

}

**class C : MyInterface**{

override fun bar(){

println("child class bar")

}

override fun foo(){

super.foo()

println("child class foo")

}

}