1. Write a Kotlin program to calculate LCM using while Loop and if Statement

fun main() {

val a = 9

val b = 27

val lcm = findLCM(a, b)

println("LCM of $a and $b: $lcm")

}

private fun findLCM(a: Int, b: Int): Int {

val biggerNum = if(a > b) a else b

var lcm = biggerNum

while(true) {

if(((lcm % a) == 0) && ((lcm % b) == 0)) {

break

}

lcm += biggerNum

}

return lcm

}

Output: LCM of 9 and 27: 27

2. Write a Kotlin program to calculate the Sum of Natural Numbers Using Recursion

a. The number whose sum is to be found is stored in a variable number.

b. Initially, the addNumbers() is called from the main() function with 20 passed as an argument

fun main() {

var number = 20

println("Sum of $number numbers ${addNumbers(number)}")

}

fun addNumbers(number:Int) : Int{

var output = 0;

if(number>0){

output = number + addNumbers(number-1)

}

return output;

}

Output : Sum of 20 numbers 210

3. Write a Kotlin program to calculate the average of all the elements in an Array using for loop.

fun main() {

val numArray = doubleArrayOf(4.3, 37.5, -25.6, 30.0, 68.40, 20.3,10.2)

var sum = 0.0

for (num in numArray) {

sum += num

}

val average = sum / numArray.size

println("The average is: %.2f".format(average))

}

Output: The average is: 20.73

4. Write a Kotlin program to Find Factorial of a number using for loop

fun main() {

val num = 20

var total = 0

for (i in 1..num) {

total = num\*i

}

println("The factorial is: $total")

}

Output: The factorial is: 400

5. Write a program to print month in words, based on input month in numbers.(using switch case)

fun main() {

val num = 12

when(num){

1 -> { println("January")}

2 -> { println("February") }

3 -> { println("March") }

4 -> { println("April") }

5 -> { println("May") }

6 -> { println("June") }

7 -> { println("July") }

8 -> { println("August") }

9 -> { println("September") }

10 -> { println("October") }

11 -> { println("November") }

12 -> { println("December") }

else -> { println("Enter Valid Number") }

}

}

Output : December

6. Write a Kotlin program to print the result of the following operations. Declare variables and initialize them with given values .

a. -5 + 8 \* 6

b. (55+9) % 9

c. 20 + -3\*5 / 8

d. 5 + 15 / 3 \* 2 - 8 % 3

fun main() {

val a =-5 + 8 \* 6

val b = (55+9) % 9

val c = 20 + -3\*5 / 8

val d = 5 + 15 / 3 \* 2 - 8 % 3

println("a = $a")

println("b = $b")

println("c = $c")

println("d = $d")

}

Output:

a = 43

b = 1

c = 19

d = 13

7. Write a Kotlin program to find greatest number in an array

fun main() {

val intArray = arrayOf(10,29,30)

var largest = intArray[0]

for (num in intArray) {

if (largest < num)

largest = num

}

println("largest number is $largest")

}

Output: largest number is 30

8. Write a Kotlin program to calculate difference between two time periods where Time is a user defined class

**a. class Time(internal var hours: Int, internal var minutes: Int, internal var seconds: Int)**

b. Write a function called “difference” which takes two arguments for start and stop time and returns the difference between the two

**class Time(internal var hours: Int, internal var minutes: Int, internal var seconds: Int)**

fun main() {

val start = Time(12, 40, 15)

val stop = Time(8, 12, 15)

val diff: Time

diff = difference(start, stop)

print("TIME DIFFERENCE: ${start.hours}:${start.minutes}:${start.seconds} - ")

print("${stop.hours}:${stop.minutes}:${stop.seconds} ")

print("= ${diff.hours}:${diff.minutes}:${diff.seconds}")

}

fun difference(start: Time, stop:Time) : Time{

val diff = Time(0, 0, 0)

if (stop.seconds > start.seconds) {

--start.minutes

start.seconds += 60

}

diff.seconds = start.seconds - stop.seconds

if (stop.minutes > start.minutes) {

--start.hours

start.minutes += 60

}

diff.minutes = start.minutes - stop.minutes

diff.hours = start.hours - stop.hours

return diff

}

Output: TIME DIFFERENCE: 12:40:15 - 8:12:15 = 4:28:0

Write a Kotlin Program to display a transpose of a matrix

Matrix

a11 a12 a13

a21 a22 a23

Transposed Matrix

a11 a21

a12 a22

a13 a23

fun main() {

val row = 2

val column = 3

val matrix = arrayOf(intArrayOf(2, 3, 4), intArrayOf(5, 6, 4))

// Display current matrix

display(matrix)

// Transpose the matrix

val transpose = Array(column) { IntArray(row) }

for (i in 0..row - 1) {

for (j in 0..column - 1) {

transpose[j][i] = matrix[i][j]

}

}

// Display transposed matrix

display(transpose)

}

fun display(matrix: Array<IntArray>) {

println("The matrix is: ")

for (row in matrix) {

for (column in row) {

print("$column ")

}

println()

}

}

Output:

The matrix is:

2 3 4

5 6 4

The matrix is:

2 5

3 6

4 4

10. Implement indexOfMax() function so that it returns the index of the largest element in the array, or null if the array is empty

fun main() {

val arr = arrayOf(12,4,5,6,17,8,1)

indexOfMax(arr)

}

fun indexOfMax(array: Array<Int>){

var largest = array.first()

var index = 0

array.forEachIndexed{ i, element ->

if(largest<element){

largest = element

index = i

}

}

println("Index of Largest number $largest is : $index")

}

Output : Index of Largest number 17 is : 4

11. Any array may be viewed as a number of "runs" of equal numbers.

For example, the following array has two runs:

1, 1, 1, 2, 2

Three 1's in a row form the first run, and two 2's form the second.

This array has two runs of length one:

3, 4

And this one has five runs:

1, 0, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0

Implement the runs() function so that it returns the number

of runs in the given array.

fun main() {

val array = arrayOf(1,0,1,0,0,0,3,3,5,6,6,6,6)

val total = runs(array)

println("Total runs : $total")

}

fun runs(array: Array<Int>): Int{

var run = 1

var oldData=array[0]

array.forEach{element->

if(element!=oldData){

run++

oldData=element

}

}

return run

}

Output: Total runs : 7

12. Palindrome: Implement a function which returns true if the given string is a palindrome or false if not

fun main() {

val name = "Arora"

val array = name.toList()

val size = array.size-1

var isPalindrome = true

for(i in 0..size/2){

if(array[i].toLowerCase()!=array[size-i].toLowerCase()){

isPalindrome = false

}

}

if(isPalindrome)

print("$name is Palindrome")

else

print("$name is not Palindrome")

}

Output: Arora is Palindrome

13. Write a function for sum, difference, multiplication and division of two numbers which takes the operator as an additional argument. This function is to be implemented using when in Kotlin

fun main() {

operation(20,15,"\*")

}

fun operation(num1:Int,num2:Int,opr:String){

when(opr){

"+" -> println(num1+num2)

"-" -> println(num1-num2)

"\*" -> println(num1\*num2)

"/" -> println(num1/num2)

else -> println("Invalid Operatation")

}

}

Output: 300

14. Write a lambda function which will filter all the even numbers for the input List<Int>

fun main() {

val array = arrayOf(22,31,45,54,62,76,88,98,14,89)

val evenList : (Array<Int>) -> ArrayList<Int> = {

var evenList = ArrayList<Int>()

array.forEach{

if(it%2==0)

evenList.add(it)

}

evenList

}

println(evenList.invoke(array))

}

Output: [22, 54, 62, 76, 88, 98, 14]

**Tuple & Destructuring Declaration**

15. Create a class to hold the members for runs, age, nom, name and a primary key as player\_number. Create a List of players with appropriate values for each of the members and also ensure unique player\_number in the List.

a) Write a function to return a tuple with all the members values for the given player\_number

b) Write a program to destructure the return value of the function and print the individual values

data class Player( var runs : Int,

var age : Int,

var nom : Int,

var name : String,

var player\_number : Int ){}

fun main() {

val players = ArrayList<Player>()

players.add(Player(58,23,25,"P1",1))

players.add(Player(60,22,13,"P3",3))

players.add(Player(54,21,20,"P4",4))

players.add(Player(57,24,21,"P5",5))

players.add(Player(56,25,22,"P6",6))

val player = getTuple(4,players)

println(player)

// destructuring

val runs = player?.component1()

val age = player?.component2()

val nom = player?.component3()

val name = player?.component4()

val player\_number = player?.component5()

println("Destructured Data: runs=$runs, age=$age, nom=$nom ,name=$name, player\_number=$player\_number")

}

fun getTuple(number:Int,players:ArrayList<Player>) : Player?{

var player : Player? = null

players.forEach{

if(number.equals(it.player\_number)){

player = it

}

}

return player

}

Output : Player(runs=54, age=21, nom=20, name=P4, player\_number=4)

Destructured Data: runs=54, age=21, nom=20 ,name=P4, player\_number=4

**Extensions:**

16. Write an extension function to the existing string class which will replace all the spaces with “\_” for the given string

fun main() {

val name = "Poonam Mangotra"

println(name)

println(name.replaceSpaces())

}

fun String.replaceSpaces() : String{

return this.replace(" ","\_")

}

Output: Poonam Mangotra

Poonam\_Mangotra

17. Write an extension function to the existing List class to reverse the list and return into a new list.

fun main() {

val list = (1..10).toList()

println("list : $list")

val reverse: List<Int> = list.replaceSpaces()

println("reverse : $reverse")

}

fun List<Int>.replaceSpaces() : List<Int>{

return this.reversed()

}

Output: list : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

reverse : [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

**Class**

18. Design and develop an OO program in Kotlin to implement an application to update/search Cricket Statistics. [use data class]

The abstract class named cricket is created. Two classes bat and bowl are derived from the cricket.

Data Members :

 player\_ name

 number\_of\_matches

 runs

Member functions :

 update():- this function takes input statistics from the admin for Batsman and Bowler.

 write\_to():- this function writes the objects' data to a List [ List<Bat>, List<Bowl>]

 Display():-this function is used to display the statistics for batsman and bowler.

 Search():-this function is used to search for a particular player and then display to standard output.

 Fifties():- this function is used to return number of fifties made by a specific player.

 Hundreds():-this function is used to return number of hundreds made by a particular player.

 Wickets():-this function is used to return number of wickets taken by a particular player.

 Overs():- this function is used to return number of overs bowled by a particular player.

 Use constructors to initialize the bat and bowler objects.

 Implement INHERITANCE by inheriting Cricket class into two new classes

- bat

- bowler

- Use appropriate data members and member functions inside derived classes.

 Create a List and store the records of the various players and display the stats on user's request.

 Make your program more interactive.

 Test thoroughly for various inputs.

abstract class Cricket(var player\_name: String, var number\_of\_matches: Int,var runs: Int){

fun update(cricket : Cricket){

player\_name = cricket.player\_name

number\_of\_matches = cricket.number\_of\_matches

runs = cricket.runs

}

fun display(){

println("player\_name: $player\_name number\_of\_matches: $number\_of\_matches runs $runs")

}

abstract fun fifties()

abstract fun hundreds()

abstract fun wickets()

abstract fun overs()

fun write\_to(cricket : Cricket){

player\_name = cricket.player\_name

number\_of\_matches = cricket.number\_of\_matches

runs = cricket.runs

}

fun search(name: String){

}

}

class Bat(player\_name: String, number\_of\_matches: Int, runs: Int) : Cricket(player\_name,number\_of\_matches,runs){

override fun fifties(){

println("Fifties : ${runs/50}")

}

override fun hundreds(){

println("Hundreds : ${runs/100}")

}

override fun wickets(){

println("No Info for Batsman")

}

override fun overs(){

println("No Info for Batsman")

}

}

class Bowl( player\_name: String, number\_of\_matches: Int, runs: Int,var wickets : Int,var overs: Int) : Cricket(player\_name,number\_of\_matches,runs){

override fun fifties(){

println("No Info for Bowler")

}

override fun hundreds(){

println("No Info for Bowler")

}

override fun wickets(){

println("Wickets : $wickets")

}

override fun overs(){

println("Overs : $overs")

}

}

fun main() {

val bat : Bat = Bat("Player 1", 120, 3500)

val bowl : Bowl = Bowl("Player 2", 100, 1000,20,2000)

bat.display()

bat.fifties()

bat.hundreds()

bowl.display()

bowl.wickets()

bowl.overs()

}

Output : player\_name: Player 1 number\_of\_matches: 120 runs 3500

Fifties : 70

Hundreds : 35

player\_name: Player 2 number\_of\_matches: 100 runs 1000

Wickets : 20

Overs : 2000

19. Define a class to represent a bank account. Include the following:

**Data members:**

a. Name of the depositor b. Account number c. Type of account d. Balance amount in the acccount

**Member functions:**

a. To assign initial values. b. To deposit an amount. c. To withdraw an amount after checking the balance. d. To display name & balance.

Write a program to test your class. Modify the program for handling 10 customers.

class Bank{

var name : String = ""

var accountNumber : String =""

var type: String =""

var balance : Double = 0.0

fun init(name : String , number: String, type: String, bal : Double){

this.name = name

accountNumber = number

this.type = type

balance = bal

}

fun deposit(amount : Double){

balance = balance+amount

println("Amount deposited. Balance: $balance")

}

fun withdraw(amount : Double){

balance()

if(amount < balance){

balance = balance - amount

println("Amount Withdrawl. Balance: $balance")

}else{

println("Enter less amount Balance: $balance")

}

}

fun balance(){

println("Name : $name Balance: $balance")

}

}

fun main() {

val bank : Bank = Bank()

bank.init("ABC", "012356465754736","Savings",2000000.0)

bank.deposit(10000.0)

bank.withdraw(50000.0)

bank.withdraw(4445444345.0)

bank.balance()

}

Output: Amount deposited. Balance: 2010000.0

Name : ABC Balance: 2010000.0

Amount Withdrawl. Balance: 1960000.0

Name : ABC Balance: 1960000.0

Enter less amount Balance: 1960000.0

Name : ABC Balance: 1960000.0

20. Create a class called player with the properties named runs and number\_of\_matches. Overload the “+” operator to return the sum of total runs and number\_of\_matches for two player objects

data class Player(var runs:Int, var number\_of\_matches: Int)

fun main() {

val p1: Player = Player(50,2)

val p2: Player = Player(80,5)

println("Total runs ${p1.runs} + ${p2.runs} = ${p1.runs.plus(p2.runs)}")

println("Total number\_of\_matches ${p1.number\_of\_matches} + ${p2.number\_of\_matches} = ${p1.number\_of\_matches.plus(p2.number\_of\_matches)}")

}

Output: Total runs 50 + 80 = 130

Total number\_of\_matches 2 + 5 = 7

21. Define a class named Payment that contains a member variable of type double that stores the amount of the payment and appropriate getter and setter methods. Also create a method named paymentDetails that outputs an English sentence to describe the amount of the payment.

Next, define a class named CashPayment that is derived from Payment. This class should redefine the paymentDetails method to indicate that the payment is in cash. Include appropriate constructor(s).

Define a class named CreditCardPayment that is derived from Payment. This class should contain member variables for the name on the card, expiration date, and credit card number. Include appropriate constructor(s). Finally, redefine the paymentDetails method to include all credit card information in the printout.

Create a main method that creates at least two CashPayment and two

CreditCardPayment objects with different values and calls paymentDetails for each.

open class Payment(){

var payment: Double = 0.0

get() = field

set(value) { field = value }

constructor(payment:Double) : this(){

this.payment = payment

}

open fun paymentDetails(){

println("Payment : $payment")

}

}

class CashPayment(payment : Double) : Payment(payment){

override fun paymentDetails(){

println("Cash : $payment")

}

}

class CreditCardPayment (var name : String,var number : String,var expiryDate: String) : Payment(){

override fun paymentDetails(){

println("Card Details: name = $name, number = $number, expiryDate = $expiryDate")

}

}

fun main() {

val p1 : CashPayment = CashPayment(550.0)

p1.paymentDetails()

val p2 : CashPayment = CashPayment(10050.0)

p2.paymentDetails()

val p3 : CreditCardPayment = CreditCardPayment("ABC", "1232432534","02/23")

p3.paymentDetails()

}

Output: Cash : 550.0

Cash : 10050.0

Card Details: name = ABC, number = 1232432534, expiryDate = 02/23

22. Write a program to create a class Book with the following data members: isbn, title and price. Inherit the class Book to two derived classes: Magazine and Novel with the following data members:

Magazine: type

Novel: author

Populate the details using constructors.

Create a magazine and Novel instance and display the details.

open class Book(var isbn : String,

var title: String,

var price: Double){

open fun displayDetails(){

println("Book Details : isbn = $isbn, title = $title, price = $price")

}

}

class Magazine(var type: String, isbn : String,

title: String,

price: Double) : Book(isbn,title,price){

override fun displayDetails(){

println("Magazine Details :type = $type, isbn = $isbn, title = $title, price = $price")

}

}

class Novel(var author: String, isbn : String,

title: String,

price: Double) : Book(isbn,title,price){

override fun displayDetails(){

println("Novel Details :author = $author, isbn = $isbn, title = $title, price = $price")

}

}

fun main() {

val magazine = Magazine("Business", "2ew3", "Entrepreneur", 550.0)

val novel = Novel("Motivation", "1325", "The Secret", 550.0)

magazine.displayDetails()

novel.displayDetails()

}

Output: Magazine Details :type = Business, isbn = 2ew3, title = Entrepreneur, price = 550.0

Novel Details :author = Motivation, isbn = 1325, title = The Secret, price = 550.0

23. Create an interface called RBI with functions

bool createAccount()

float getBalance()

bool withDraw()

float Deposit(var amount : Int)

Create four classes SBI, ICICI, BOB, HSBC which implements the interface RBI.

interface RBI{

fun createAccount() : Boolean

fun getBalance() : Float

fun withDraw() : Boolean

fun Deposit(amount : Int) : Float

}

class SBI : RBI{

override fun createAccount() : Boolean{ return true}

override fun getBalance() : Float{return 0.0f}

override fun withDraw() : Boolean{return true}

override fun Deposit(amount : Int) : Float{return amount.toFloat()}

}

class ICICI : RBI{

override fun createAccount() : Boolean{ return true}

override fun getBalance() : Float{return 0.0f}

override fun withDraw() : Boolean{return true}

override fun Deposit(amount : Int) : Float{return amount.toFloat()}

}

class BOB : RBI{

override fun createAccount() : Boolean{ return true}

override fun getBalance() : Float{return 0.0f}

override fun withDraw() : Boolean{return true}

override fun Deposit(amount : Int) : Float{return amount.toFloat()}

}

class HSBC : RBI{

override fun createAccount() : Boolean{ return true}

override fun getBalance() : Float{return 0.0f}

override fun withDraw() : Boolean{return true}

override fun Deposit(amount : Int) : Float{return amount.toFloat()}

}

**Collections**

24. Take a nested list and return a single flattened list with all values except nil/null.

The challenge is to write a function that accepts an arbitrarily-deep nested list-like structure and returns a flattened structure without any nil/null values.

For Example

input: [1,[2,3,null,4],[null],5]

output: [1,2,3,4,5]

fun main() {

val list = listOf(

listOf(1),

listOf(2,3,null,4),

listOf(null),

listOf(5)

)

var flattenList = list.flatten()

println(list)

println(removeNull(flattenList))

}

fun removeNull(flattenList : List<\*>) : ArrayList<\*>{

var result = ArrayList<Any>()

flattenList.forEach{

if(it!=null){

result.add(it)

}

}

return result

}

Output : [[1], [2, 3, null, 4], [null], [5]]

[1, 2, 3, 4, 5]

25. Write a Kotlin Program to Convert Map to List

fun main() {

val hMap: MutableMap<String, Int> = HashMap()

hMap["A"] = 165

hMap["B"] = 616

hMap["C"] = 671

val entries: List<String> = hMap.entries.map { "(${it.key}, ${it.value})" }

entries.forEach { print(it) }

}

Output: (A, 165)(B, 616)(C, 671)

26. Write a Kotlin Program to Join Two Lists

fun main() {

val list1: List<Int> = listOf(1,2,3,4,5)

val list2 : List<Int> = listOf(6,7,8,9)

val entries = list1 + list2

entries.forEach { print("$it,") }

}

Output : 1,2,3,4,5,6,7,8,9,

27. Write a Kotlin Program to Sort a Map by Values

fun main() {

var capitals = hashMapOf<String, String>()

capitals.put("Nepal", "Kathmandu")

capitals.put("India", "New Delhi")

capitals.put("United States", "Washington")

capitals.put("England", "London")

capitals.put("Australia", "Canberra")

val result = capitals.toList().sortedBy { (\_, value) -> value}.toMap()

for (entry in result) {

print("Key: " + entry.key)

println(" Value: " + entry.value)

}

}

Output: Key: Australia Value: Canberra

Key: Nepal Value: Kathmandu

Key: England Value: London

Key: India Value: New Delhi

Key: United States Value: Washington

28. Write a Kotlin program to filter all the null values and odd numbers from a List

fun main() {

val numbers: List<Int?> = listOf(1, 2, null, 4,5,6,7,8,null,10)

val nonNullNumbers = numbers.filterNotNull().filter{ it % 2 != 0 }

println(nonNullNumbers)

}

Output: [1, 5, 7]

29. Write a Kotlin Program to Sort a List of Custom Objects by Property [ public class CustomObject(val customProperty: String) {}]

public class CustomObject(val customProperty: String) {}

fun main() {

var list = listOf(CustomObject("R"),CustomObject("D"),CustomObject("H"),CustomObject("J"),CustomObject("Z"),CustomObject("A"))

val sortedList = list.sortedWith(compareBy{it.customProperty})

sortedList.forEach{

print(it.customProperty)

}

}

Output: ADHJRZ

30. Write a Kotlin Program to Sort Elements in Lexicographical Order (Dictionary Order)

For Example:

Input Array should consist:

val words = arrayOf("Ruby", "C", "Python", "Java")

Output should be :

C

Java

Python

Ruby

public class CustomObject(val customProperty: String) {}

fun main() {

val words = arrayOf("Ruby", "C", "Python", "Java", "AI", "Kotlin", "Flutter")

for (i in 0..(words.size-2)) {

for (j in i + 1..(words.size-1)) {

if (words[i].compareTo(words[j]) > 0) {

// swap words[i] with words[j[

val temp = words[i]

words[i] = words[j]

words[j] = temp

}

}

}

println("In lexicographical order:")

for (i in 0..(words.size-1)) {

println(words[i])

}

}

Output : In lexicographical order:

AI

C

Flutter

Java

Kotlin

Python

Ruby