**DSA ASSIGNMENT**

**Q.1**

import java.util.Scanner;

public class DSA1 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int target = sc.nextInt();

        int n = sc.nextInt();

        int a[] = new int[n];

        int temp[] = new int[2];

        for (int i = 0; i < a.length; i++) {

            a[i] = sc.nextInt();

        }

        for (int i = 0; i < a.length - 1; i++) {

            for (int j = i + 1; j < a.length; j++) {

                if (a[i] + a[j] == target) {

                    temp[0] = i;

                    temp[1] = j;

                    System.out.println("[" + temp[0] + ", " + temp[1]+"]");

                    sc.close();

                    return;

                }

            }

        }

        System.out.println("Pair not found.");

        sc.close();

    }

}

**Q.2.**

import java.util.\*;

class DSA2{

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

          int n = sc.nextInt();

          int temp = 0;

          while(n>0){

            int m = n%10;

            temp = temp\*10+ m;

            n/=10;

          }

          System.out.println(temp);

           sc.close();

        }

    }

**Q.3.**

import java.util.\*;

class DSA3 {

    public boolean isValid(String s) {

        Stack<Character> stack = new Stack<Character>();

        for (char c : s.toCharArray()) {

            if (c == '(' || c == '[' || c == '{') {

                stack.push(c);

            } else {

                if (stack.isEmpty()) {

                    return false;

                }

                char top = stack.peek();

                if ((c == ')' && top == '(') || (c == ']' && top == '[') || (c == '}' && top == '{')) {

                    stack.pop();

                } else {

                    return false;

                }

            }

        }

        return stack.isEmpty();

    }

}

**Q.4.**

class DSA4 {

    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

        if (list1 != null && list2 != null) {

            if (list1.val < list2.val) {

                list1.next = mergeTwoLists(list1.next, list2);

                return list1;

            } else {

                list2.next = mergeTwoLists(list1, list2.next);

                return list2;

            }

        }

        if (list1 == null)

            return list2;

        return list1;

    }

}

class ListNode {

    int val;

    ListNode next;

    ListNode(int val) {

        this.val = val;

    }

}

**Q.5**

class DSA5 {

    public int maxSubArray(int[] nums) {

        int maxSum = Integer.MIN\_VALUE;

        int currentSum = 0;

        for (int i = 0; i < nums.length; i++) {

            currentSum += nums[i];

            if (currentSum > maxSum) {

                maxSum = currentSum;

            }

            if (currentSum < 0) {

                currentSum = 0;

            }

        }

        return maxSum;

    }

}

**Q.6.**

import java.util.Scanner;

public class DSA6 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer: ");

        int x = sc.nextInt();

        sc.close();

        boolean result = isPalindrome(x);

        System.out.println("Output: " + result);

    }

    public static boolean isPalindrome(int x) {

        if (x < 0) {

            return false;

        }

        int original = x;

        int reversed = 0;

        while (x != 0) {

            int digit = x % 10;

            reversed = reversed \* 10 + digit;

            x /= 10;

        }

        return original == reversed;

    }

}

**Q.7.**

import java.util.\*;

public class DSA7{

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of steps: ");

        int n = sc.nextInt();

        sc.close();

        int distinctWays = climbStairs(n);

        System.out.println("Output: " + distinctWays);

    }

    public static int climbStairs(int n) {

        if (n <= 0) {

            return 0;

        } else if (n == 1) {

            return 1;

        }

        int[] dp = new int[n + 1];

        dp[0] = 1;

        dp[1] = 1;

        for (int i = 2; i <= n; i++) {

            dp[i] = dp[i - 1] + dp[i - 2];

        }

        return dp[n];

    }

}

**Q.8.**

import java.util.\*;

public class DSA8{

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of strings: ");

        int n = sc.nextInt();

        sc.nextLine();

        String[] strs = new String[n];

        System.out.println("Enter the strings:");

        for (int i = 0; i < n; i++) {

            strs[i] = sc.nextLine();

        }

        sc.close();

        String commonPrefix = longestCommonPrefix(strs);

        System.out.println("Output: " + commonPrefix);

    }

    public static String longestCommonPrefix(String[] strs) {

        if (strs == null || strs.length == 0) {

            return "";

        }

        String prefix = strs[0];

        for (String str : strs) {

            while (!str.startsWith(prefix)) {

                prefix = prefix.substring(0, prefix.length() - 1);

            }

        }

        return prefix;

    }

}

**Q.9**

import java.util.\*;

class DSA9 {

    public boolean isAnagram(String s, String t) {

        if (s.length() != t.length()) {

            return false;

        }

        char[] sChars = s.toCharArray();

        char[] tChars = t.toCharArray();

        Arrays.sort(sChars);

        Arrays.sort(tChars);

        return Arrays.equals(sChars, tChars);

    }

}

**Q.10.**

Import java.util.\*;

class DSA10 {

    public List<List<Integer>> levelOrder(TreeNode root) {

        List<List<Integer>> ans = new ArrayList<>();

        if(root == null) return ans;

        Queue<TreeNode> queue = new LinkedList<>();

        queue.add(root);

        while(!queue.isEmpty()) {

            int size = queue.size();

            List<Integer> level = new ArrayList<>();

            for(int i = 0; i < size; i++) {

                TreeNode node = queue.peek();

                queue.remove();

                if(node.left != null) queue.add(node.left);

                if(node.right != null) queue.add(node.right);

                level.add(node.val);

            }

            ans.add(level);

        }

        return ans;

    }

}