

# Rajalakshmi Engineering College

Name: varsha s  
Email: 241501237@rajalakshmi.edu.in  
Roll no:  
Phone: 9342191041  
Branch: REC  
Department: AI & ML - Section 1  
Batch: 2028  
Degree: B.E - AI & ML

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 2\_CY

Attempt : 1  
Total Mark : 40  
Marks Obtained : 40

#### Section 1 : Coding

##### 1. Problem Statement

Noah is analyzing numbers within a given range  $[A, B]$  and wants to calculate a special sum. For each number in the range, he calculates the product of its odd digits (ignoring even digits). If the number contains no odd digits, it is skipped. The sum of these products for all numbers in the range is the result.

Write a program to compute this sum.

Example

Input:

10 12

Output:

3

Explanation:

For 10, odd digits = 1, product = 1.

For 11, odd digits = 1, 1, product =  $1 * 1 = 1$ .

For 12, odd digits = 1, product = 1.

Total sum =  $1 + 1 + 1 = 3$

### ***Input Format***

The input consists of two space-separated integers A and B, representing the inclusive range boundaries.

### ***Output Format***

The output prints a single integer representing the sum of the products of odd digits for all numbers in the range.

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 10 12

Output: 3

### ***Answer***

```
// You are using Java
import java.util.Scanner;
class OddDigitProductSum {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Read input range A and B
        int A = scanner.nextInt();
        int B = scanner.nextInt();

        int totalSum = 0;

        // Iterate through the range
```

```

for (int num = A; num <= B; num++) {
    int temp = num;
    int product = 1;
    boolean hasOddDigit = false;

    // Process each digit
    while (temp > 0) {
        int digit = temp % 10;

        if (digit % 2 != 0) {
            product *= digit;
            hasOddDigit = true;
        }

        temp /= 10;
    }

    if (hasOddDigit) {
        totalSum += product;
    }
}

// Output the result
System.out.println(totalSum);

scanner.close();
}
}

```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Ram wants to evaluate the time required to break even on an investment based on initial costs, monthly profits, and monthly expenses. Write a program to calculate the break-even point in months and categorize the return on investment.

Compute the break-even point by using the formula:  $\text{initial cost} / (\text{monthly profit} - \text{monthly expenses})$  Based on the break-even point, classify the

return on investment into one of the following categories: Quick Return: If the break-even point is 3 months or fewer. Average Return: If the break-even point is between 4 and 12 months, inclusive. Long-term Return: If the break-even point exceeds 12 months.

Ram is new to programming, so he seeks your assistance in creating the program.

Note: monthly profit is always greater than monthly expenses.

### ***Input Format***

The first line of input consists of a double value representing the initial cost.

The second line consists of a double value representing the monthly profit.

The third line consists of a double value representing the monthly expenses.

### ***Output Format***

The first line prints "Break-even Point:", followed by the break-even point as a decimal number (of double datatype), formatted to two decimal places.

The second line prints "Category: ", followed by the investment return as a String, which can be one of:

- "Quick Return" if break-even point  $\leq 3$
- "Average Return" if break-even point  $\leq 12$
- "Long-term Return" if break-even point  $> 12$

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 10000.50

5000.75

1000.10

Output: Break-even Point: 2.50

Category: Quick Return

### ***Answer***

```

// You are using Java
import java.util.Scanner;
class BreakEvenCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Read input values
        double initialCost = scanner.nextDouble();
        double monthlyProfit = scanner.nextDouble();
        double monthlyExpenses = scanner.nextDouble();

        // Calculate net monthly income
        double netMonthly = monthlyProfit - monthlyExpenses;

        // Calculate break-even point
        double breakEvenPoint = initialCost / netMonthly;

        // Print break-even point formatted to two decimal places
        System.out.printf("Break-even Point: %.2f\n", breakEvenPoint);

        // Determine category
        String category;
        if (breakEvenPoint <= 3.0) {
            category = "Quick Return";
        } else if (breakEvenPoint <= 12.0) {
            category = "Average Return";
        } else {
            category = "Long-term Return";
        }

        // Print category
        System.out.println("Category: " + category);

        scanner.close();
    }
}

```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

Ted, the computer science enthusiast, has accepted the challenge of writing a program that checks if the number of digits in an integer matches the sum of its digits.

Guide Ted in designing and writing the code to solve this problem using a 'do-while' loop.

### ***Input Format***

The input consists of an integer N, representing the number to be checked.

### ***Output Format***

If the sum is equal to the number of digits, print "The number of digits in N matches the sum of its digits."

Else, print "The number of digits in N does not match the sum of its digits."

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 20

Output: The number of digits in 20 matches the sum of its digits.

### ***Answer***

```
// You are using Java
import java.util.Scanner;
class DigitSumChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Read the input number
        int N = scanner.nextInt();
        int num = N;

        int digitCount = 0;
        int digitSum = 0;

        // Use do-while loop to process digits
```

```

do {
    int digit = num % 10;
    digitSum += digit;
    digitCount++;
    num /= 10;
} while (num > 0);

// Check condition and print result
if (digitSum == digitCount) {
    System.out.println("The number of digits in " + N + " matches the sum of
its digits.");
} else {
    System.out.println("The number of digits in " + N + " does not match the
sum of its digits.");
}

scanner.close();
}
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Raj is solving a physics problem involving projectile motion, where he needs to calculate the time a ball hits the ground using a quadratic equation of the form  $ax^2 + bx + c = 0$ . Depending on the coefficients, the ball may hit the ground once, twice, or not at all in real time.

Help Raj find all real roots of the equation, if any.

Note: discriminant =  $b^2 - 4ac$

##### ***Input Format***

The input consists of three space-separated doubles a, b, and c, representing the coefficients of the quadratic equation.

##### ***Output Format***

If there are two real roots, print:

- "Two real solutions:"
- "Root1 = <value>"
- "Root2 = <value>"

If there is one real root, print:

- "One real solution:"
- "Root = <value>"

If there are no real roots, print:

- "There are no real solutions."

Note: values are rounded to two decimal places.

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 1 6 9

Output: One real solution:

Root = -3.00

### ***Answer***

```
// You are using Java
import java.util.Scanner;
class QuadraticSolver {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Read the coefficients
        double a = scanner.nextDouble();
        double b = scanner.nextDouble();
        double c = scanner.nextDouble();

        // Calculate the discriminant
        double discriminant = b * b - 4 * a * c;

        if (discriminant > 0) {
            // Two real solutions
```

```

double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);

// Sort roots for consistent output (smaller root first)
double minRoot = Math.min(root1, root2);
double maxRoot = Math.max(root1, root2);

System.out.println("Two real solutions:");
System.out.printf("Root1 = %.2f\n", maxRoot);
System.out.printf("Root2 = %.2f\n", minRoot);

} else if (discriminant == 0) {
    // One real solution
    double root = -b / (2 * a);
    System.out.println("One real solution:");
    System.out.printf("Root = %.2f\n", root);

} else {
    // No real solutions
    System.out.println("There are no real solutions.");
}

scanner.close();
}
}

```

**Status :** Correct

**Marks :** 10/10