**Programming Logic 1**

Write a function that accepts three positive integers as input and check if they form the sides of a triangle. Print YES if they form one, and NO if they do not. The output should be a single line containing one of these two strings: YES or NO.

package AllPractice;

import java.util.Scanner;

public class Ass3 {

public static void main(String[] args) {

Scanner sc=new Scanner(System.***in***);

System.***out***.println("Enter Triangle 1st side");

int a = sc.nextInt();

System.***out***.println("Enter Triangle 2nd side");

int b = sc.nextInt();

System.***out***.println("Enter Triangle 3rd side");

int c = sc.nextInt();

System.***out***.println(*checkTriangle*(a, b, c) ? "YES" : "NO");

}

public static boolean checkTriangle(int a, int b, int c) {

if (a <= 0 || b <= 0 || c <= 0)

return false;

if (a + b > c && a + c > b && b + c > a)

return true;

return false;

}

}

Write a function which accepts two positive integers M and N as input. There are two cases to consider:  
            (1) If M < N, then print M as output.  
            (2) If M >= N, subtract N from M. Call the difference M1. If M1 >= N, then subtract N from M1 and call the difference M2. Keep doing this operation until you reach a value k, such that,  Mk < N. You have to print the value of Mk as output.

a.      Test case 1 :

                                                        i.      Input – M = 10, N = 15

                                                       ii.      Output : 10

b.      Test case 2:

                                                        i.      Input – M = 15, N = 15

                                                       ii.      Output : 0

c.       Test case 3 :

                                                        i.      Input – M = 15, N = 10

                                                       ii.      Output : 5

package AllPractice;

import java.util.Scanner;

public class Ass4 {

public static void main(String[] args) {

Scanner sc=new Scanner(System.***in***);

System.***out***.println("Enter 1st number");

int M = sc.nextInt();

System.***out***.println("Enter 2nd number");

int N = sc.nextInt();

System.***out***.println(*findFinalValue*(M, N));

}

public static int findFinalValue(int M, int N) {

if (M < N)

return M;

while (M >= N) {

M -= N;

}

return M;

}

}

Assume a table with the following banking table for savings account with the fields:

 Customer\_Account\_Number, pan\_number, txn\_date, credit\_amount, debit\_amount, balance. There will be one record for every transaction.

Write a pseudocode to get all customers whose withdrawals are more than the deposits over the last 6 months cumulatively. State your assumptions if any and write the program (or a pseudocode). Create test data as needed.

Java Approach:

1. Initialize an empty dictionary or hashmap to store the cumulative total amounts for each customer over the last 6 months.

2. Iterate over each record in the banking table:

a. Extract the customer account number, transaction date, credit amount, and debit amount from the record.

b. Check if the transaction date falls within the last 6 months.

c. If yes,

if deposit transaction then

update the amount as amount + deposit amount for the corresponding customer account number in the dictionary.

else

update the amount as amount - withdrawal amount for the corresponding customer account number in the dictionary.

3. Iterate over the dictionary:

a. For each customer account number, check if the cumulative total amount is less than 0

b. If yes, output the customer account number.

4. End.

SQL Approach:-

SELECT Customer\_Account\_Number

FROM transactions

WHERE txn\_date >= DATE\_SUB(CURRENT\_DATE, INTERVAL 6 MONTH)

GROUP BY Customer\_Account\_Number

HAVING SUM(debit\_amount) > SUM(credit\_amount);

Assumptions:

- The "txn\_date" field is in a date format that can be easily compared to determine whether it falls within the last 6 months.

- The transactions are sorted by transaction date in ascending order.

- The banking table contains accurate and complete data for each transaction.

- The balance field is not necessary for this analysis.

Example Test Data:

Customer\_Account\_Number | txn\_date | credit\_amount | debit\_amount

1 | 2023-09-01 | 5000 | 0

1 | 2023-09-15 | 0 | 2000

2 | 2023-10-02 | 3000 | 0

2 | 2023-10-20 | 0 | 3500

3 | 2023-11-05 | 8000 | 0

3 | 2023-11-25 | 0 | 7000

4 | 2023-12-10 | 2000 | 0

4 | 2023-12-20 | 0 | 2500