Flowcharts and Pseudocode →

Flowcharts are diagram to represent solution of Problem

Graphical Representation of solution of a Problem.

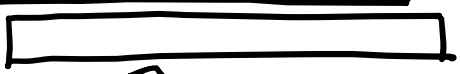
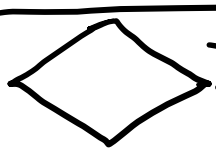
Algorithm is a finite step by step procedure to solve a particular problem

Flowchart is graphical Representation of Algorithm / Problem Solving.

To Solve →

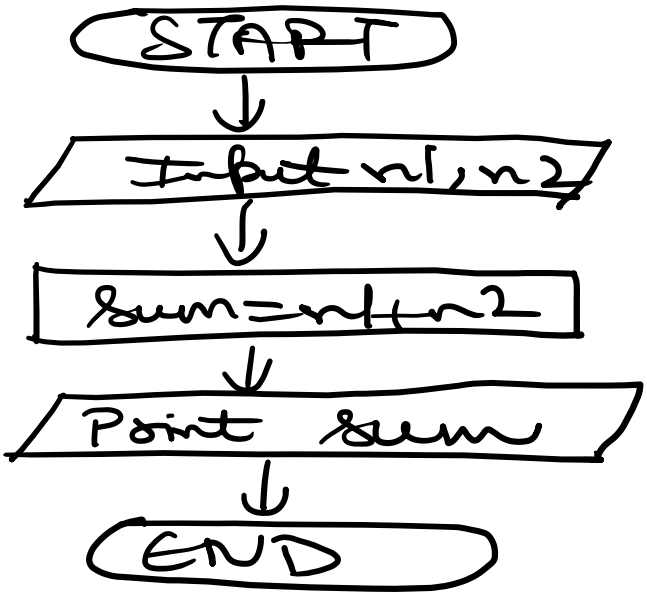
1. Divide into small parts.
2. logically arranged
3. Draw Flowchart
4. Solve Problem in Coding.

Components →

1.  → Oval → Start / END or EXIT
2.  → Parallelogram → Input / Output
3.  → Rectangle → Processing.
4.  Decision Box (Boolean → true or False) (Branches)
5.  → Arrows (Direction)

* Arrows acts as connector b/w diff shapes of Diagram *
* Branches Originates from Decision box either true branch or false branch *

Sum of Two No →



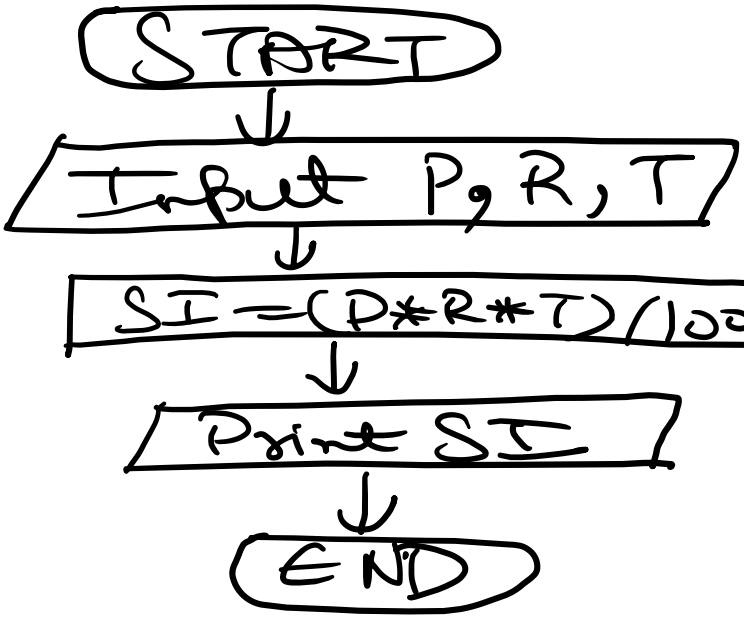
Input n1, n2
Output → Sum

PseudoCode →

Input n1, n2
Sum = n1 + n2
print Sum

Pseudocode → Pseudocode is explanation of code written in english language so that even a layman can understand working of code.

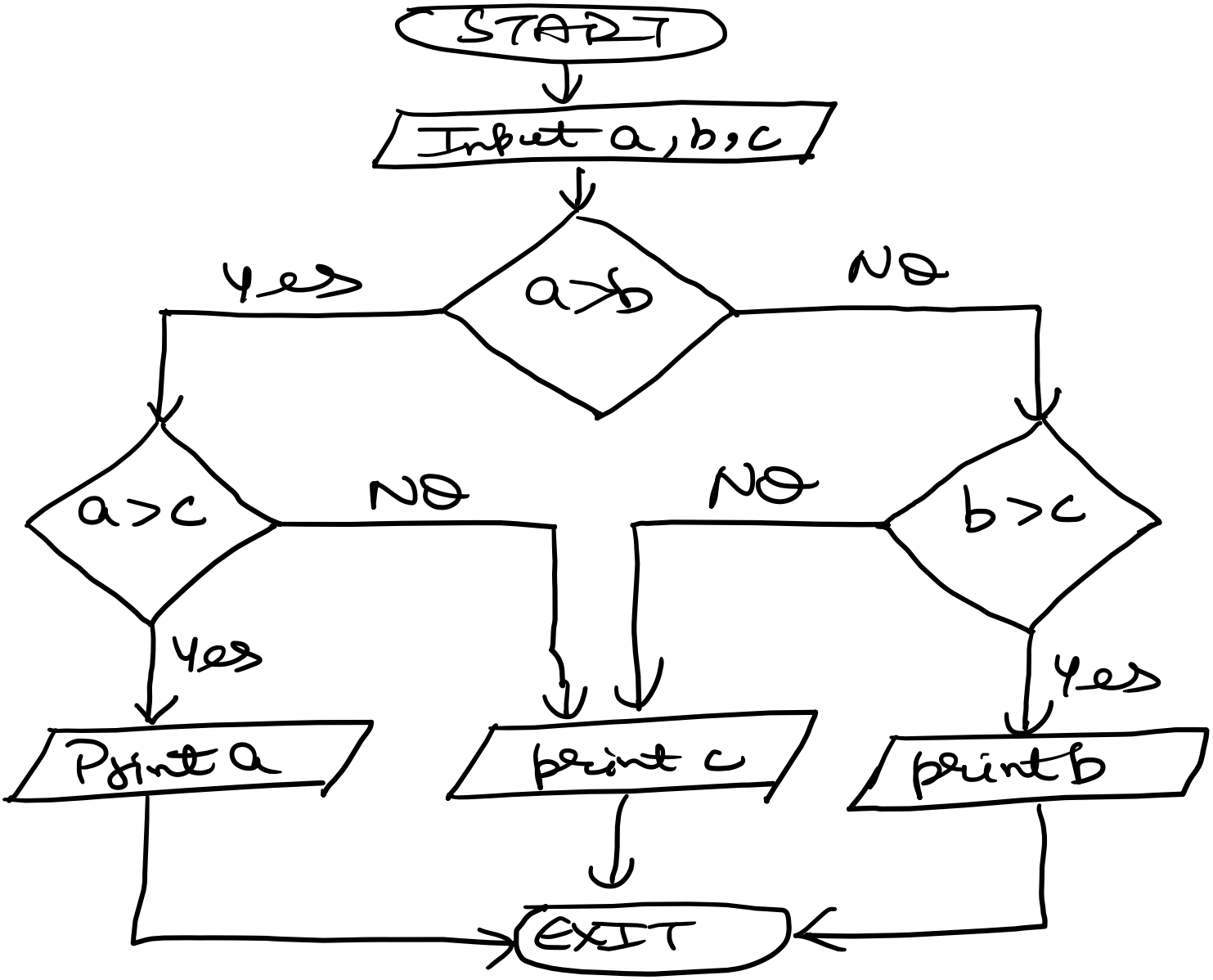
Calculate Simple Interest (SI) →



PseudoCode →

Input P, R, T
Calculations (P*R*T)/100
Output SI

Max of 3 No →



Input 3 No. | Output → Greatest No

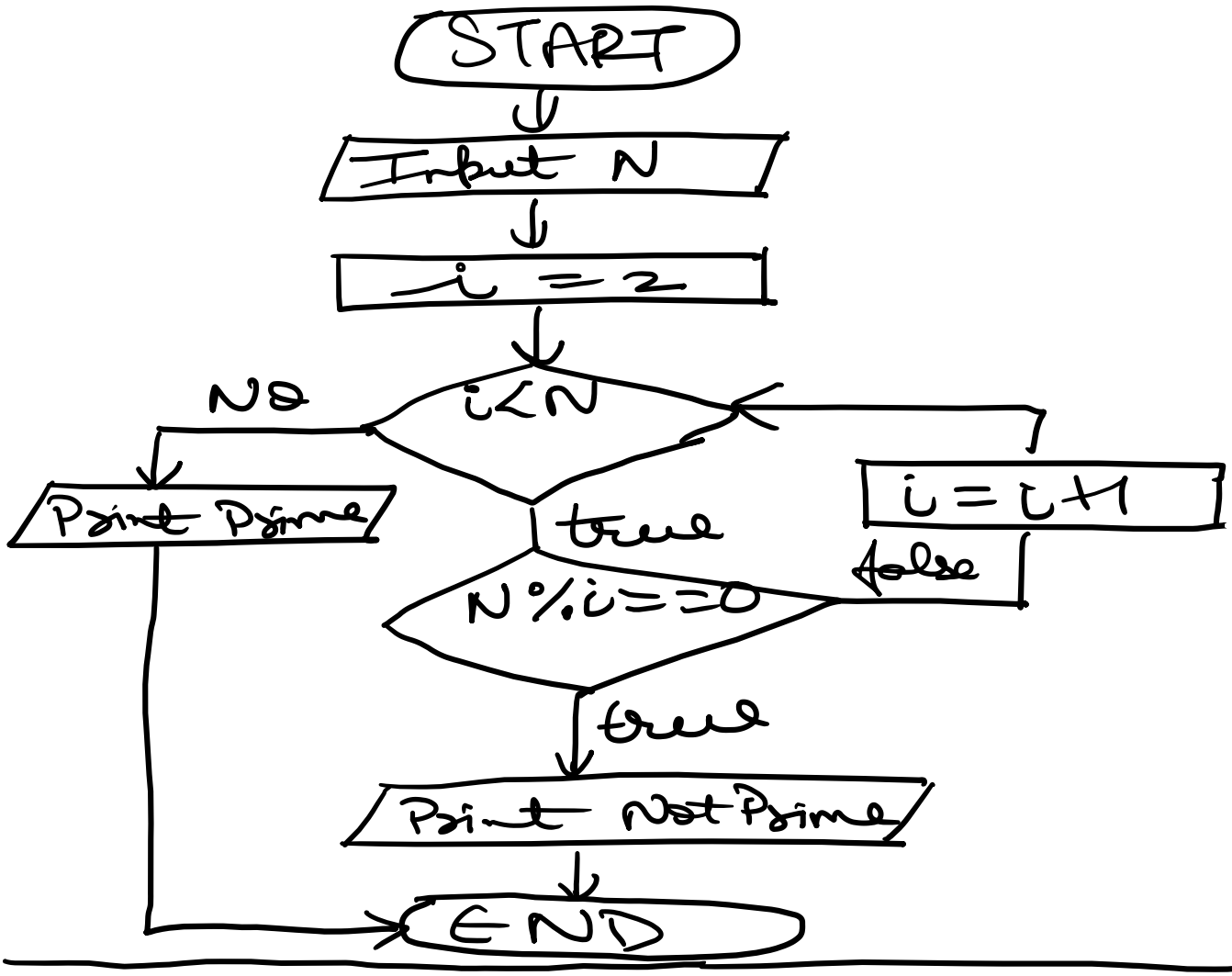
Pseudocode →

1. Start
2. Input a, b, c
3. If a > b do
   - If a > c do
     - print a
   - Else
     - print c

Else
If b>c do
print b
Else
print c

④ Exit

Find if a No is Prime or Not →



Non Prime are Composite Numbers which have more than two Factors
Eg 15 — 1, 3, 5, 15
Prime → Only Two Factors. 1 and Itself

Prime No Program in Java →

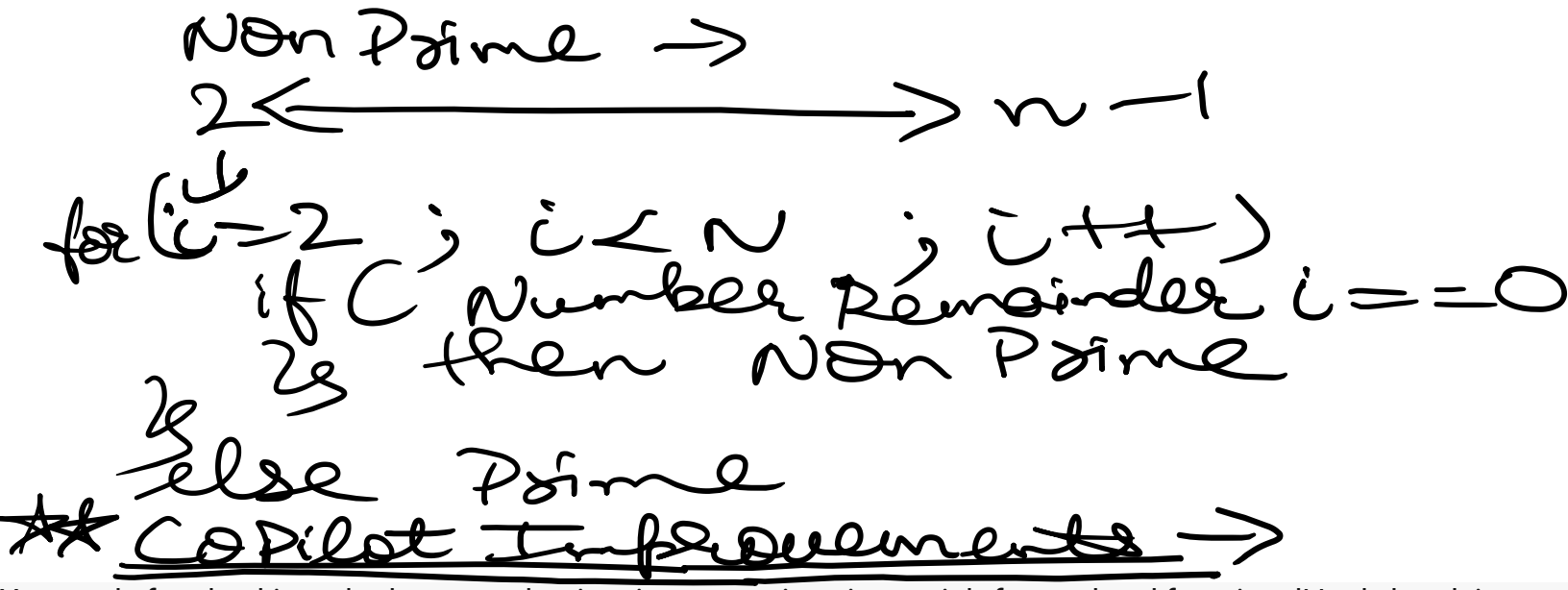
```java
import java.util.Scanner;

public class Prime {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the Value of Number = ");
        boolean Prime = true;
        int N = input.nextInt();
        for(int i = 2 ; i < N ; i++){
            if(N % i == 0){
                Prime = false;
                break;
            }
        }
        if(Prime==false){
            System.out.println(N + " is Not a Prime Number");
        }else{
            System.out.println(N + " is a Prime Number");
        }
        input.close();
    }
}
```

Non Prime is Composite No having more than 2 factors.

Prime → having 2 factors only first 1 and itself.

To check Prime or Not →

```
for ( i=2; i<N ; i++) {
  if ( N % i == 0 ) {
      Non Prime
  }
}
Else Prime
```



Your code for checking whether a number is prime or not is quite straightforward and functional! Let's break it down:

1. You've used a `Scanner` to read input from the user, which is a good approach.
2. The variable `Prime` is used to track whether the number is prime or not.
3. The loop iterates from 2 to `N-1`, checking if `N` is divisible by any number in that range.
4. If `N` is divisible by any number, you set `Prime` to `false` and break out of the loop.
5. Finally, you print whether the number is prime or not based on the value of `Prime`.

However, there are a couple of improvements you can make:

1. **Optimization**: You can reduce the loop range from `2` to `Math.sqrt(N)` to improve efficiency. If `N` is divisible by any number greater than its square root, it will also be divisible by a smaller factor.
2. **Edge Cases**: Consider handling cases where `N` is less than 2 (since prime numbers start from 2) and printing an appropriate message.

Here's an updated version of your code with these improvements:

```
import java.util.Scanner;

public class Prime {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Enter a positive integer: ");
      int N = input.nextInt();
      input.close();

      if (N < 2) {
         System.out.println(N + " is not a prime number.");
```

```
        return;
    }

    boolean isPrime = true;
    for (int i = 2; i <= Math.sqrt(N); i++) {
        if (N % i == 0) {
            isPrime = false;
            break;
        }
    }

    if (isPrime) {
        System.out.println(N + " is a prime number.");
    } else {
        System.out.println(N + " is not a prime number.");
    }
  }
}
```

% (modulous) (Arithmetic operator) → for Remainder.

== (Equality operator) to check two no equal or not.

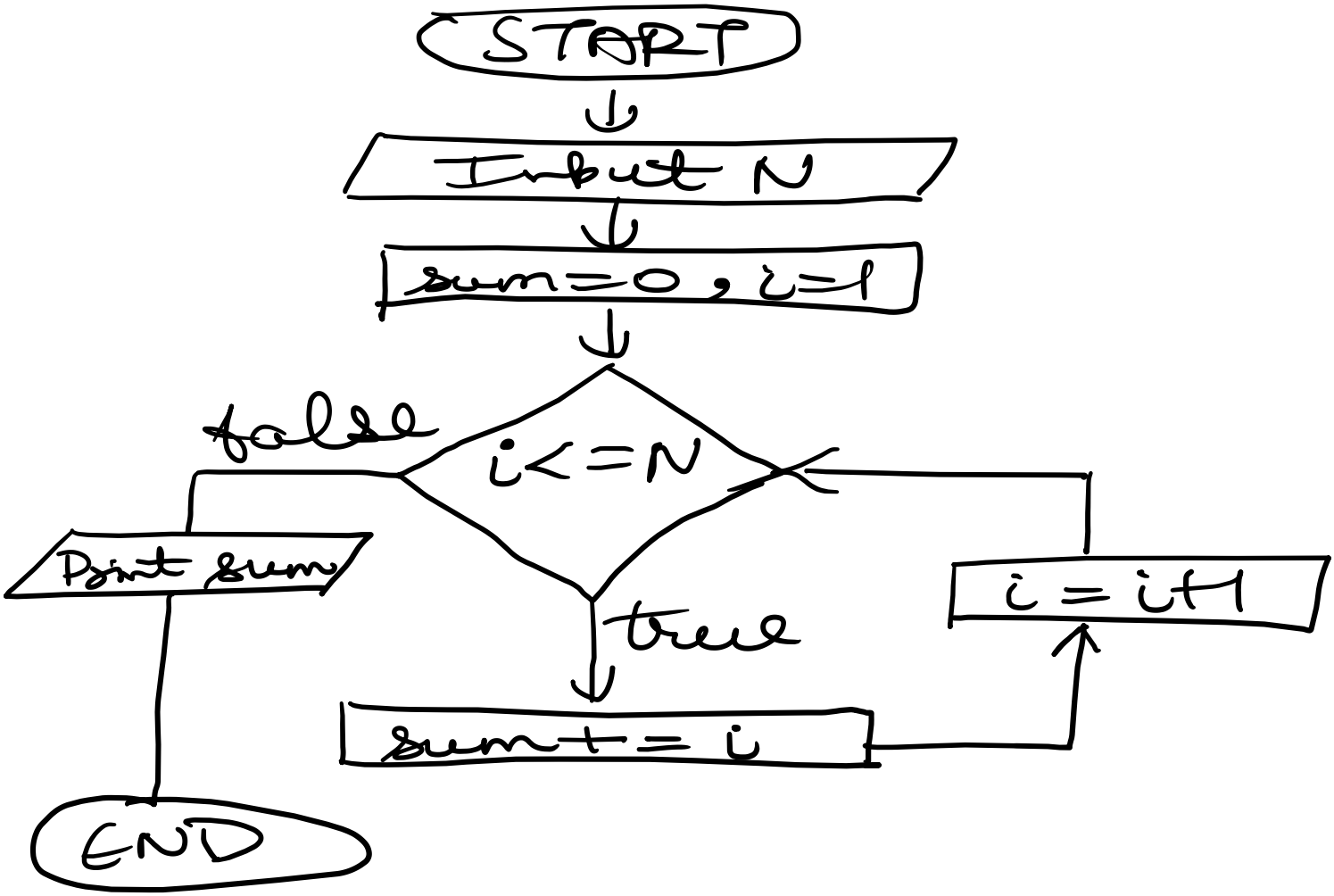
= Assignment operator used to assign value from right to left.

int a = 5;

Pseudocode - Prime =>

1. Input N
2. let i = 2
3. while (i < N) do
   if (N % i == 0) do
   Print NOT Prime
   Else
   else
   i++
4. Print Prime

Sum of First N Natural NO =>



Pseudo Code =>

1. Input N
2. i = 1, sum = 0
3. while (i <= N) do
   sum += i
   i++
4. Print sum

** Java Program of Sum of N Natural Numbers => *

```java
import java.util.Scanner;

public class SumofNaturalNo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter Value of Number = ");
        int N = input.nextInt();
        int sum = 0;
        // Loop to calculate sum of N Natural Numbers ->
        for(int i = 1 ; i<=N ; i++){
            sum += i;
        }
        System.out.println("Sum of N Natural Numbers = "+sum);
        input.close();
    }
}
```