# Mastering DSA With Java

Why Java ?

* Extensively used in industry
* Job Demand.
* Extensive Standard Library
* ***platform Independent.
* Community support.

Latest LTS version JDK.21 (Release date: 19th Sep 2023)

LTS stands for Long Term Support

*oracle (owner of Java) releases new version of Java in every 6-months.

## Introduction to programming

What is an Algorithm ?

Algorithm is a step by step procedure to solve a specific problem.

```
SET total = 0.

FOR i FROM 1 TO 5
  SET total = total + i.
  END FOR
PRINT total.
```

List of LTS versions: Java SE 8, Java SE 11, Java SE 17, Java SE 21.

There are so many updates in JDK21 but the Big update in JDK21 was

Then

```
public class HelloWorld
{
  public static void main (String[] args)
  {
System.out.println ("Hello welcome to monik
                     Learns");
  }
}
```

Now (after update)

```
void main()
{
System.out.print ("Hello welcome to monika Learns");
}
```

* In the Real life we Communicate through language like English, Kannada (or) any other langu

Eg:- I can say "Give me the pen"

So we need a language to Communicate.

* But if we want to Communicate with our Computer, laptop (or) phone, we can't use Languages like English (or) kannada.

→ We need a programming languages (C, C++, Java, Python, Javascript, PHP, R, Rust, Go, Swift, Scala etc)

→ Computer is basically an electronic device which understands only 0 & 1.

**Language of Computer is 0's & 1's**

* And we want our laptop / computer do many task for us as computers are very fast and can execute millions or trillions of instructions in seconds. we want to make some softwares to solve real world problems but first. Let's start with simple task.

adding two numbers

Eg:- 60 + 75

So how will we ask Computer to do this task?

Do we need to convert 75 + 60 into binary & then give it to computer?

you can do but its not that simple.

If we want to write a code / program in machine language (0's & 1's) then first we need to understand the CPU architecture because machine code is processor-specific. Different CPU's has their own instruction sets. then learn the instruction set & then Convert the instruction to binary.

So we use high level languages to write down our instructions.

Set of instruction is known as program

→ program is a sequence of instruction in a programming language that we give to computer to perform some specific task.

Why Computer understands only 0's & 1's?

→ Its because of their underlying hardware.

→ At the most fundamental level Computers are built from electronic components like transistors

* Transistors can be in 2 types on/off & this on/off state can easily be represented as '1' & '0'

→ Transistors are basic building blocks of modern electronics (laptop, smart devices, phones, computer memory chip, car engine & among other applications).

* A transistor can act as a switch & gate for electronic signals.

* ENIAC (Electronic Numerical Integrator And Computer) was the first programmable, electronic general purpose digital computer completed in 1945.

→ Computer basically process information using electronic switches known as transistors. which can be either ON(1) or OFF(0). Binary system uses only 2 digits 0 & 1. that's why binary system is ideal for computers because it can be easily represented using electronic switches.

* In a computer each transistor can be thought of as a switch that can be either ON(1) or OFF(0). These switches are combined to form logic gates, which are basic components of digital circuits. Logic gates perform operations such as AND, OR and NOT which are used to process information.

* It is technically possible to build computers that use other number system such as decimal, hexadecimal but Binary is perfect choice because:

① Simplicity :- logic gates can be designed using simple electronic circuits.

② efficiency :- decimal or hexadecimal would require more complex circuits to represent same amount of info which would increase the size and power consuption of computers

③ Scalability :- It can easily be extended to represent larger amount of information. This is because binary can be easily combined to form larger units such as bytes (8 bits), words (16, 32, 64 bits)

④ Universality:- Binary is a universal language that can be understood by all computers regardless of their architecture & manufacturer

* There are billions of transistor (switches) in computer / laptop. when combined these on/off switches represents data and instructions for how to compute complex task & calculations.

But the problem is for humans its really tough to tell computer to do some task in machine code (binary language)

Eg:- add 5 & 10.

Binary 1011 1000 0001 - - -
011 0000 - - - - - -

we can not write all over data and instruction in 0's & 1's.

=> But Back Then old times, to input data and instructions to computers, switches, punched card, paper tape were used.

Instructions were entered directly to the computer by setting switches to specific positions (if require precise manual manipulation)

So this comes under machine language (direct instruction, low-level, difficult to use) Then Assembly language came.

* mnemonics were used eg: ADD, SUB...
* Assembler.
* closely tied to specific Hardware architecture

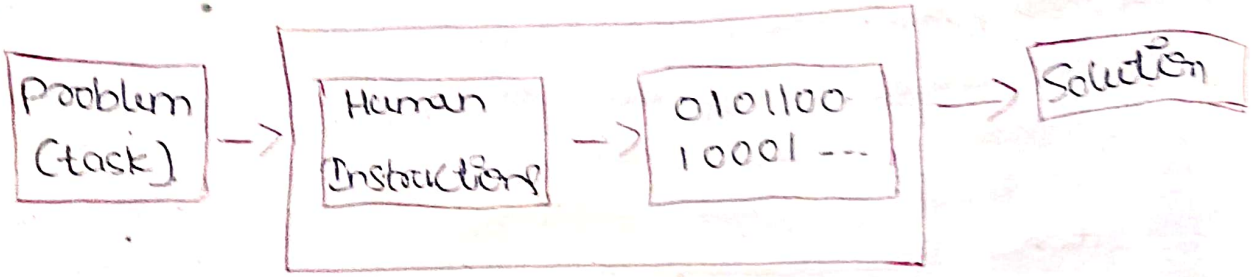
After that High Level Language came:-
* Not tied to Hardware architecture
* Compiler / Interpreter
* C, C++, Java, Python, JS, etc.
* more readable and maintainable.
* cross platform Compatibility.

So programing languages allow humans to communicate with computer by providing a structured way to give instructions and solve problems.

Using programing language we can write our instruction and data in human readable format

what is programing language?
A PL is a system of notation for writing computer program.
* Everything would be converted into 0's and 1's whether it a number (0, 1, 2, 3 --), (or) a characters (a, b, c, A, B, C --- ) (or) any special characters (@, $, ! --).

How to talk with computer ⟶ using programing languages




→ These instructions are written in any programming languages (C, C++, C#, Java etc) then Compiler convert into High level language (HLL) into Computer understandable format and then Computer gives us solution.

But it is compulsory to give clear and step by step instructions to computer, you can't give some vague instructions why?

Because computer has no brain. It is fast accurate, efficient and doesn't feel any tiredness but it is dumb. So we have to tell clearly what to do.

For Example :-

your mom told you to buy mangos. So you go to market and if you don't get mangos on one shop, then you go to other shop. As human you have your brain so you can decide on the fly to go to other shop to buy mangos. But Computer doesn't have its own brain.

* A programming language is a formal system of communication used to write instruction that a computer can execute. It provides a set of rules, syntax & semantics that allow humans to create program to perform specific task on a computer.

* programming language are used to develop software, automatic tasks, manipulate data & interact with H/w,

HLL (C++, Python, Java) :→ closer to human language, easy to read & write.

LLL (machine code, Assembly) :→ closer to computer's hardware.

## Programming and Coding :-

### Programming

* A process of designing, planning, writing, testing and maintaining instructions that a computer can follow to perform specific tasks.

key aspects :- (tasks)

* Defining the logic and flow of program
* writing algorithm & flowchart
* Choosing right algorithm & DS
* writing code.
* Debugging, testing, maintaing software

### Coding

* coding is specifically set of lines of code in any
* programming language follows syntax rules to create working programs.
* Coding is one part of programming.

## Real life example

programming

* Sara mathur decides what dishes to make, ensure she has all the ingredients for that at home & think about how each dish will complement the others
* Then make a plan to cook everything efficiently & on time
* Then serve.

* Coding is like following a recipe.
  i) She chop vegetables, boil water, frying, making dough (writing specific parts of the program).
  ii) She measure out ingredients & follow step by step instruction (syntax) to create the dishes.

## Algorithms & program

### Algorithm

* Set of steps to accomplish a task.
  Eg:- making a tea.
* In formal way, algorithm is a finite set of steps @ finite set of instructions to accomplish a task

### program

A program is a concrete implementation of an algorithm written in a programming language.
* A program is a sequence of instruction written in a programming language that perform a specific task.

@ In more precise way, an algorithm is a finite sequence of steps/ instructions to solve a program @ to solve a computational task.

* we write a algorithm in a simple english language
* should be finite and unambiguous & correct.

@ Solve a problem based on algorithm.

we write program in programing language.

Example for an Algorithm.

Making Tea.

Step 1 : Start.
Step 2 : pour water into a saucepan
Step 3 : Turn the gas on and put saucepan on gas flame.
Step 4 : Add spices (ginger, cardamom) & boil it.
Step 5 : Add a Tea powder to boiling water
Step 6 : Let it boil for about 1-2 minutes
Step 7 : pour milk to the saucepan.
Step 8 : Bring the mixture to boil again. ensuring if doesn't overflow.
Step 9 : Add sugar to taste & stir for 1-min
Step 10 : Let it boil for about 2-3 minutes on low flame.
Step 11 : pour the tea through a strainer into cup to remove tea powder & spices
Step 12 : Serve hot tea.

Algorithm is not specific to Computer science. When we write step-by-step solution of any problem for any field, that is known as algorithm.

Example: 2.

Sum of two numbers

Step 1: Start
Step 2: Declare variables num1, num2 & sum
Step 3: Take first input in num1.
Step 4: Take second input in num2
Step 5: Calculate sum by adding num1 & num2. assign the result to variable sum
sum = num1 + num2.
Step 6: Display sum.
Step 7: Stop.

Pseudocode:-

* Pseudocode is a way of representing an algorithm using a combination of plain english language and programming constructs.
* It is a bridge between the algorithm and actual programing code.

Algorithm → pseudocode → program.

Example:-1

pseudocode for Sum of 2 numbers:-

```
BEGIN
READ 'num1' & 'num2'
COMPUTE  sum = num1 + num2
PRINT 'sum'
END.
```

It is more structured than plain English but less formal than programming language.

Example 2:-

Check number is even (or) odd

```
BEGIN.
READ num.
If num %2 == 0
   PRINT 'even.
ELSE
   PRINT odd.
END IF.
END.
```

Example 3:- print Nos from 1 to 10.

```
BEGIN.
  SET num = 1.
WHILE num <= 10. DO.
     PRINT NO.
     num = num + 1.
END WHILE.
```

(or)

```
BEGIN
SET num = 1
REPEAT
    PRINT num.
    num = num + 1
UNTIL num <= 10.
END.
```

* It helps the programmer in planning the solution to the problem as well as readers in understanding the approach to the problem.

## Flow Chart

It is a visual representation of a process (or) an algorithm. using standardized symbols. Each step in a process/algorithm is represented by different symbol and contains a short description of the step.

Flow charts are used in various fields like computer programming, business process, modelling & system design to visualize workflows & process.

## Key elements

1. Start/End (Terminator):-

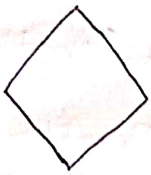
(or)



Oval — rounded rectangle.

② Input/output :-  (parallelogram)

③ process :-  (rectangle)

④ Decision :-  (Diamond) or (Rhombus)

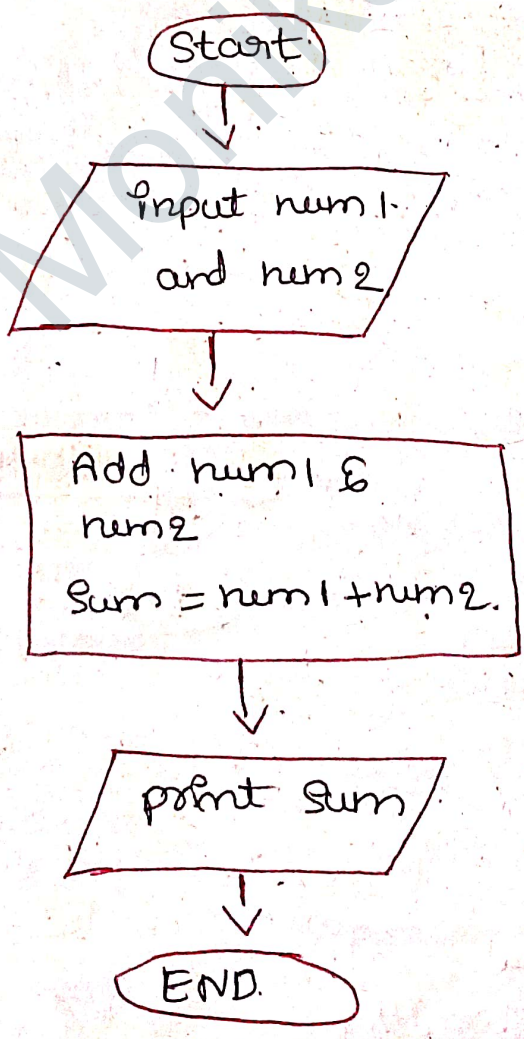
⑤ Flow Lines :- ⟶.

⑥ Connector. :- ○ → on-page Connector.
(small circle).

 (off page connector).

⑦ Document. 

Example :- Add two nos flowchart.



## Practice Questions

① Calculate the area of rectangle.

Algorithm

Step 1 : Start

Step 2 : Declare variables length & breadth & area

Step 3 : Take input from the user and store values for variable length & breadth.

Step 4 : Calculate the area using the formula

area = length × breadth.

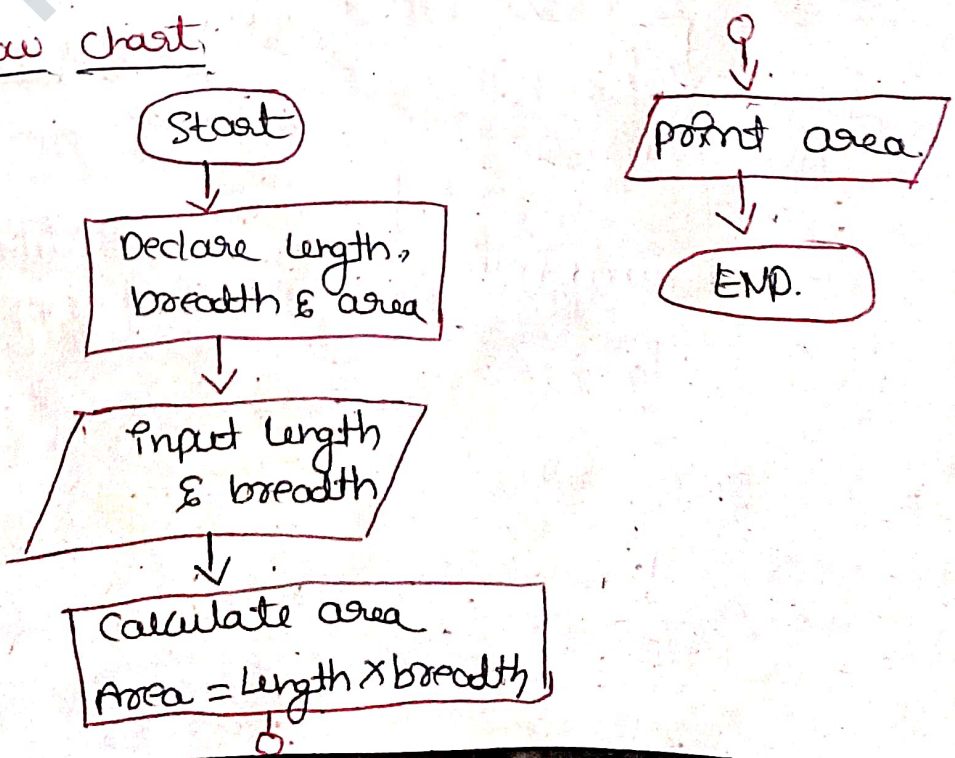
Step 5 : print area.

Step 6 : Stop

Pseudo Code

```
BEGIN.
READ length, breadth.
Compute area = length x breadth
PRINT area.
END.
```

Flow Chart:



② Swap two numbers without using third variable.

## Algorithm

Steps:

Step 1 : Start

Step 2 : Declare variable a & b

Step 3 : Read values of a & b

Step 4 : a = a+b.

Step 5 : b = a - b.

Step 6 : a = a - b

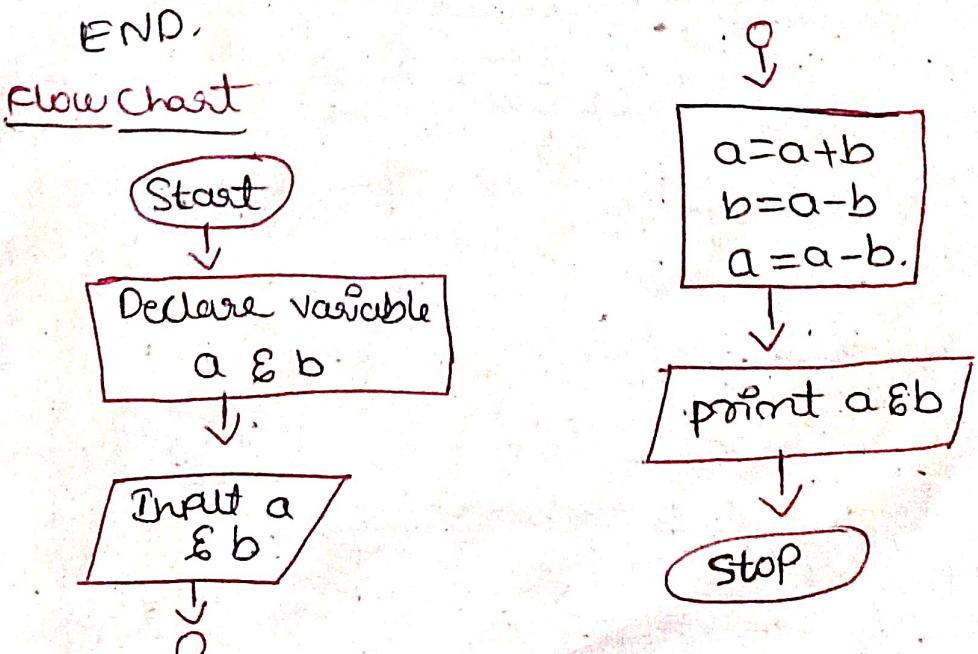
Step 7 : Display values of a & b.

Step 8 : Stop.

## Pseudocode

```
BEGIN
READ a
READ b
COMPUTE   a = a+b
          b = a-b.
          a = a-b
PRINT a.
PRINT b.
END.
```

## Flow Chart



③ Calculate the Area of Circumference of a Circle

Algorithm

Step 1 : Start

Step 2 : Declare variable for, radius (r) area, and Circumference.

Step 3 : Take radius as input

Step 4 : Calculate the area using the formula

area = 3.14 * r * r

Step 5 : calculate the Circumference using the formula

Circumference = 2 * 3.14 * r

Step 6 : Display values of area & Circumference

Step 7 : Stop.

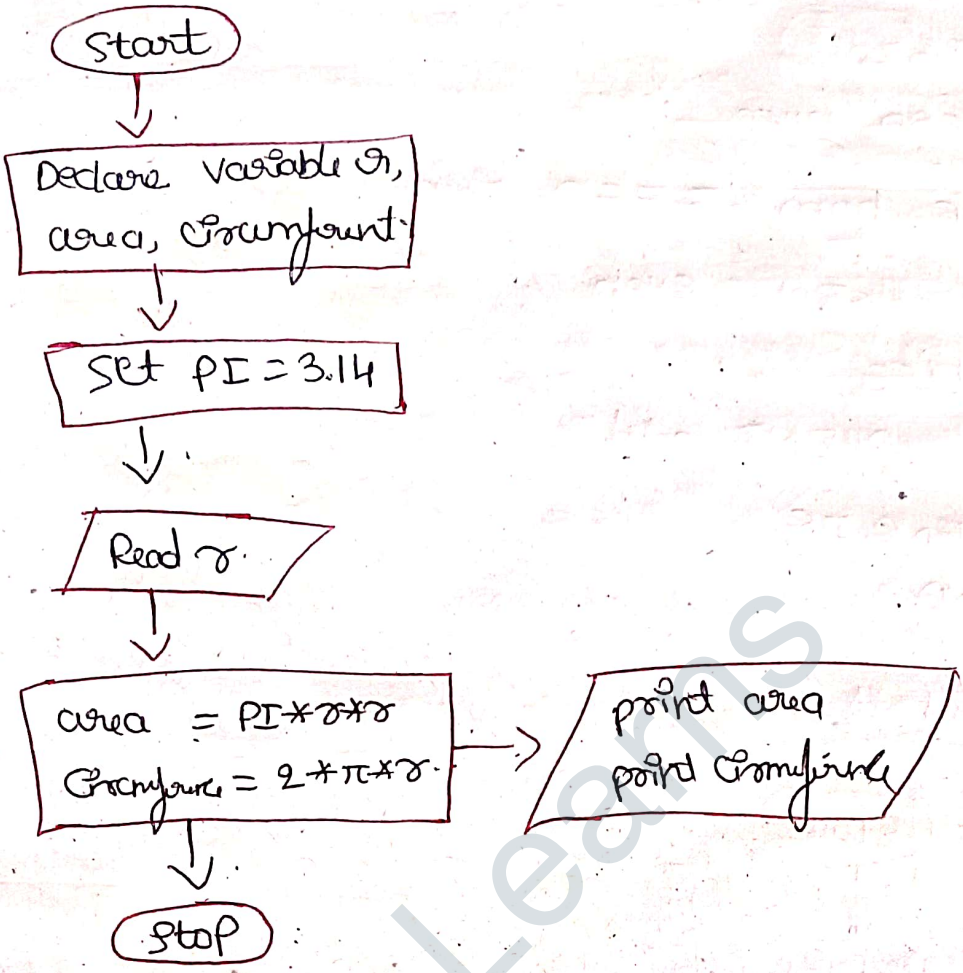
Pseudo Code

```
BEGIN
  Read r.
  SET PI = 3.14
  COMPUTE area = PI * r * r.
          Circumference = 2 * PI * r.

  PRINT area
  PRINT Circumference

END.
```

Flowchart




4. Given no is even (or) odd.

Algorithm

Step 1 : Start

Step 2 : Declare a variable num.

Step 3 : Take a input & store it in num.

Step 4 : Check if the num is Completely divisible by 2 using the formula

num %. 2 == 0. then the given number is even

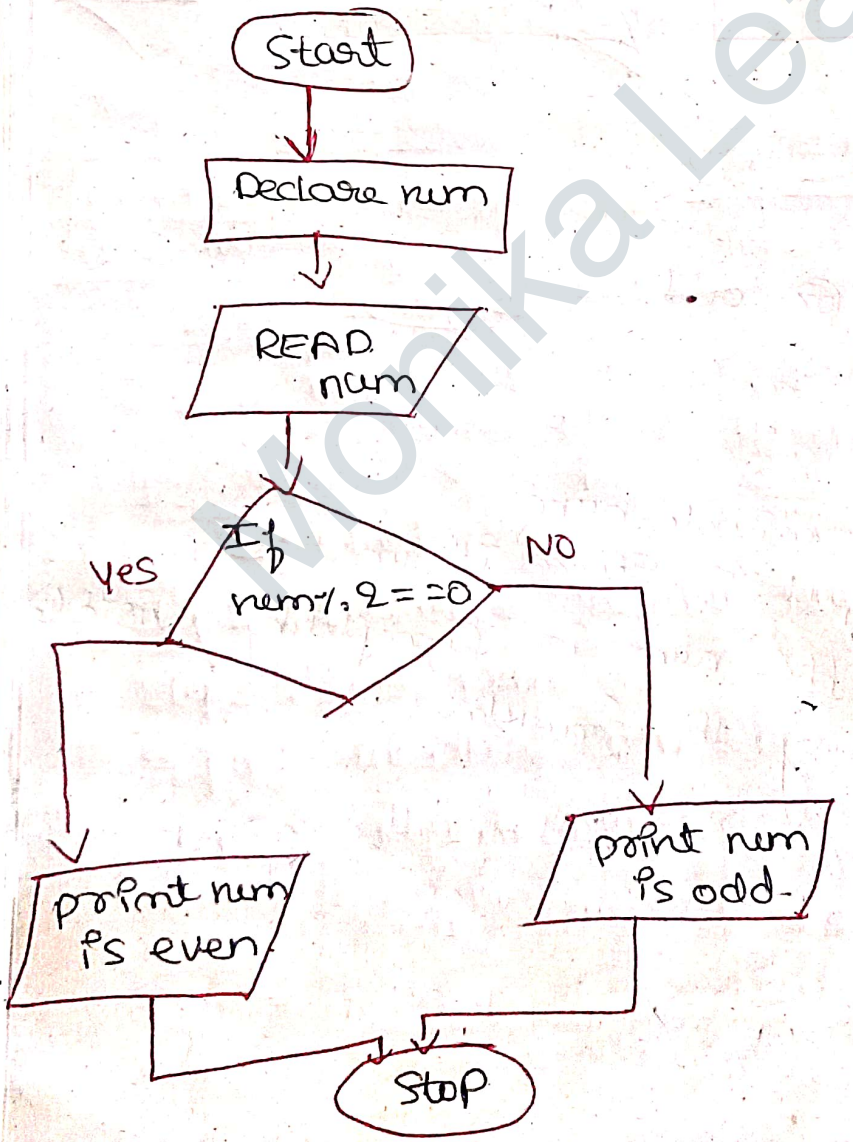
Step 5 : otherwise print number is odd.

Step 6 : Stop.

## Pseudocode

```
BEGIN.
READ num.
IF num % 2 == 0. Then,
    PRINT even
ELSE
    PRINT odd.
END IF.
END.
```

## Flowchart



5. Greatest of three numbers using ternary operator

Algorithm

Step 1 : Start

Step 2 : Declare variables A, B and C

Step 3 : use a nested ternary.

```
if A>B
    if A>C. ⟶ max=A.
    else
        max=C
else (A<B)
    if B>C ⟶ max=B
    else
        max=C.
```

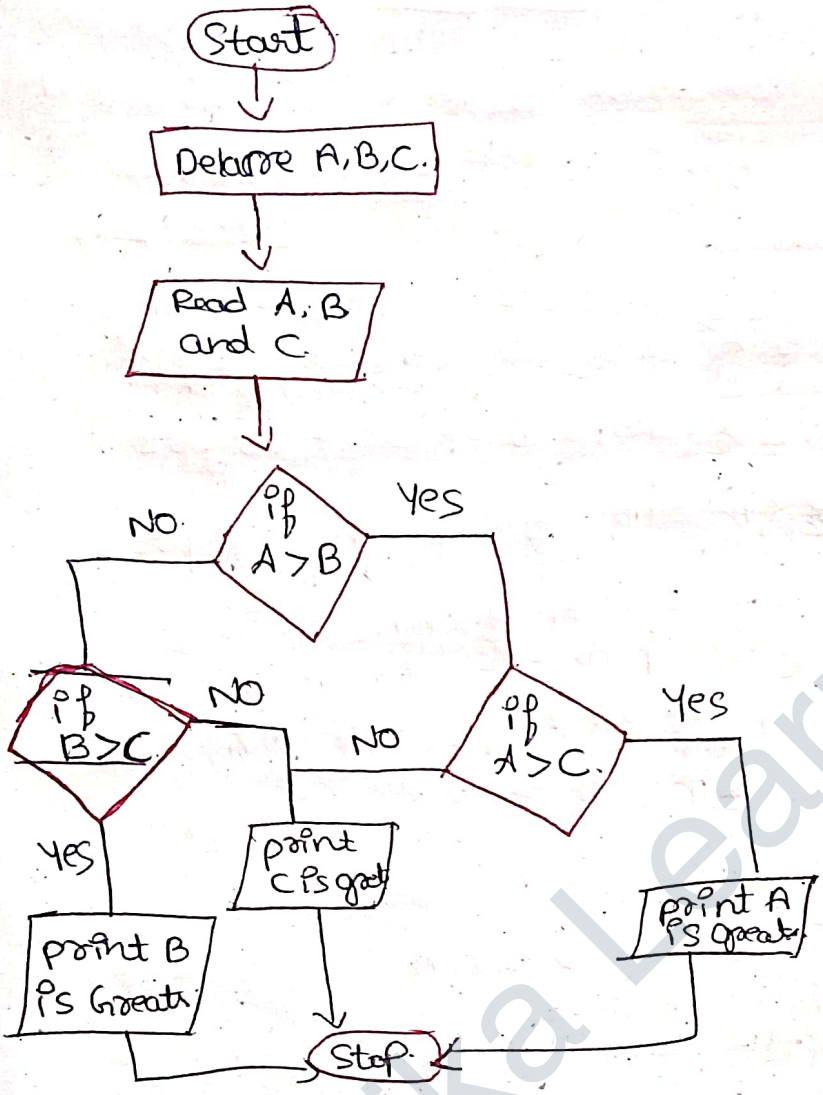
Step 4 : output max.

Step 5 : Stop.

Pseudocode

```
BEGIN.
  READ A,B,C.
COMPUTE
    IF A>B.
        If A>C
            max=A
        ELSE
            max=C.
    ELSE
        If B>C
            max=B
        ELSE
            max=C.
    ENDIF.
PRINT Greatest no
END.
```

Flowchart




6. Sum of digits of a given number.

Step 1 : Start
Step 2 : Declare a variable num.
Step 3 : Read num
Step 4 : Initialize sum = 0.
Step 5 : Repeate num > 0.

digit = num % 10.
sum = sum + digit
num = num / 10

step 5 : print sum

Step 6 : Stop

add of it digits

Ex :- 1234 => 10

to get the last digit of number

d = 1234 % 10 = 4

S = 0 + 4 = 4.

n = 123 (ignore decimal part)

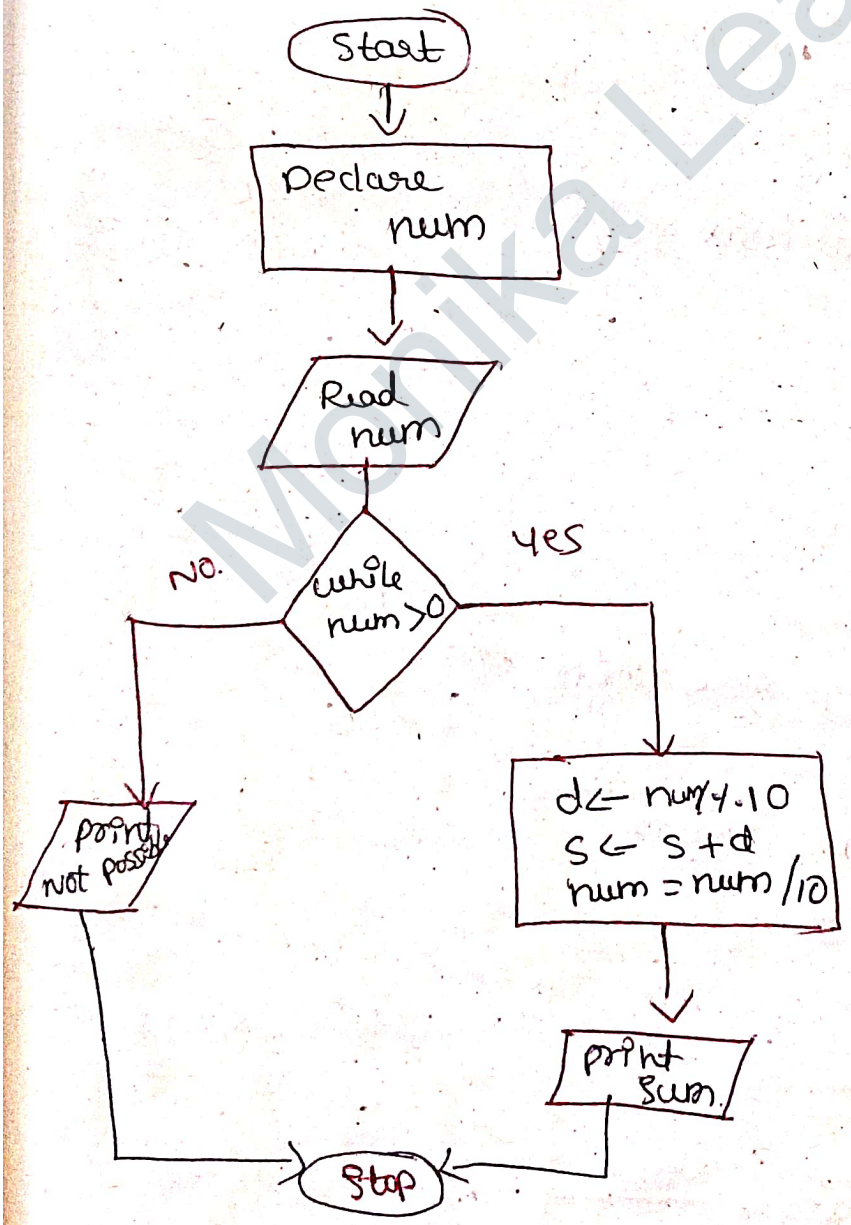
10)1234(123
10
23
20
34
30/4

123.4
1234
40

## Pseudocode

```
BEGIN
 READ num
  Sum ← 0
  while num > 0 DO
      digit ← num % 10
      Sum ← Sum + digit
      num ← num / 10
   END WHILE
   PRINT Sum
  END.
```

## Flowchart



# Assignment Questions

1. Maximum of three numbers without terrary Operator

## Algorithm

Step 1: Start

Step 2: Declare variables A, B, and C.

Step 3: Read A, B and C.

Step 4: if A > B and A > C then

max ← A.

else if B > C then

max ← B

else

max ← C.

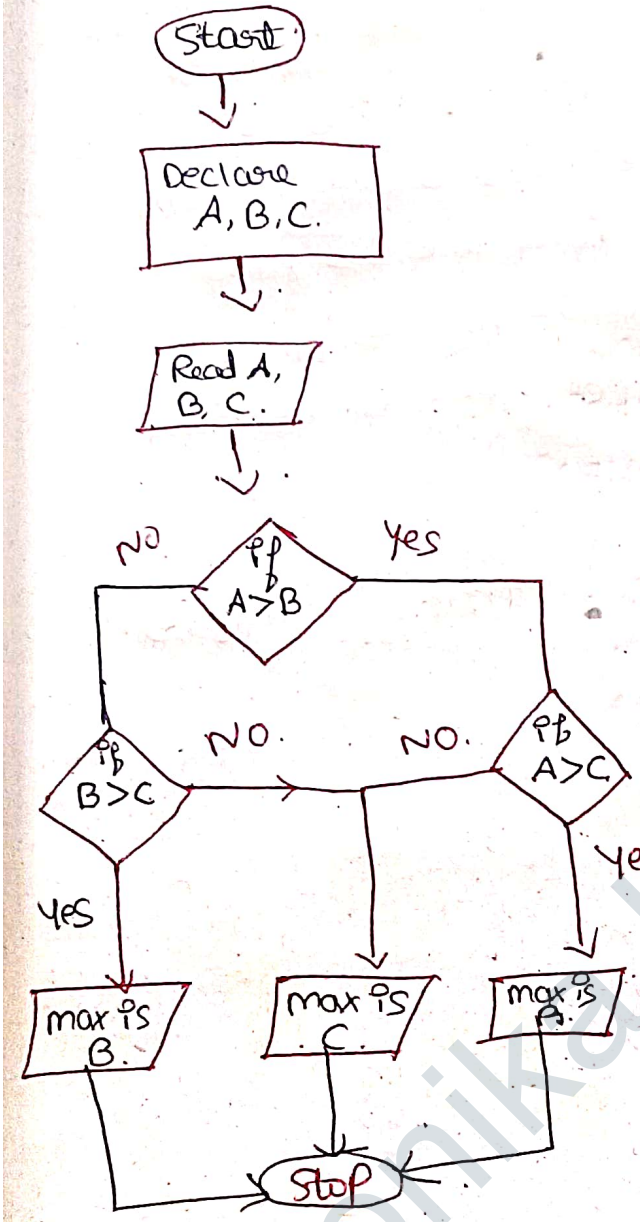
Step 5: print max

Step 6: Stop.

## Pseudocode

```
BEGIN.
  READ A, B, C
  IF A>B AND A>C THEN.
      max←A
  ELSE IF B>C THEN
      max ← B
  ELSE
      max←C
  ENDIF.
  PRINT max
END.
```

Flowchart




② Find out whether the given year is leap year or not

Step 1: Start
Step 2: Input the year.
Step 3: If (year is divisible by 4) Then
If (year is divisible by 100) Then
If (year divisible by 400) Then
It is a leap year
else
It is not a leap year
else
It is a leap year
else
not a leap year.

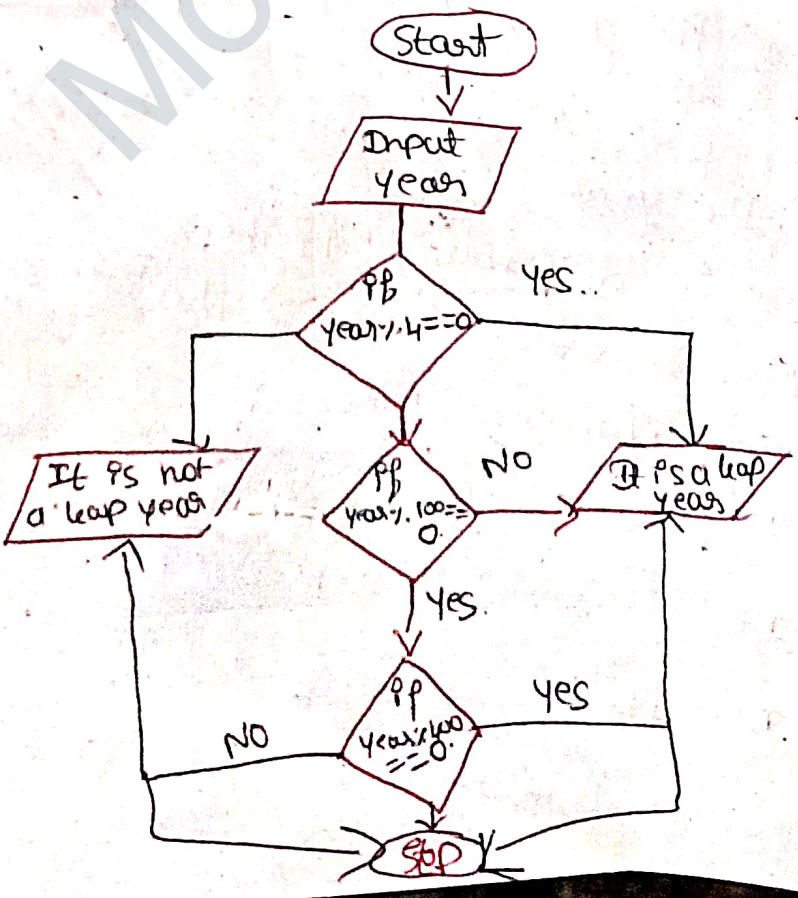
Step 4: print the result.

Step 5: Stop

## Pseudocode

```
BEGIN
  READ year
  IF year %.4 ==0 THEN
  IF year %.100 ==0 THEN.
  IF year %.400 ==0 THEN.
     PRINT Leap Year
   ELSE
  PRINT Not a Leap year
  ELSE
    PRINT Leap year
  ELSE
    PRINT NOT a Leap year
  END IF
END
```

## Flowchart



3. Check weather the given number is a prime (23) not:

Algorithm

Step 1: Start

Step 2: Input a number n.

Step 3:- If $n <= 1$, it is not a prime

Step 4:- Set flag = 0

Step 5:- Loop from i = 2 to n-1.

if n % i == 0, set flag = 1 (not a prime)

break

Step 6:- if flag = 0.

print prime

else

not prime

Step 7 :- End.

Pseudocode

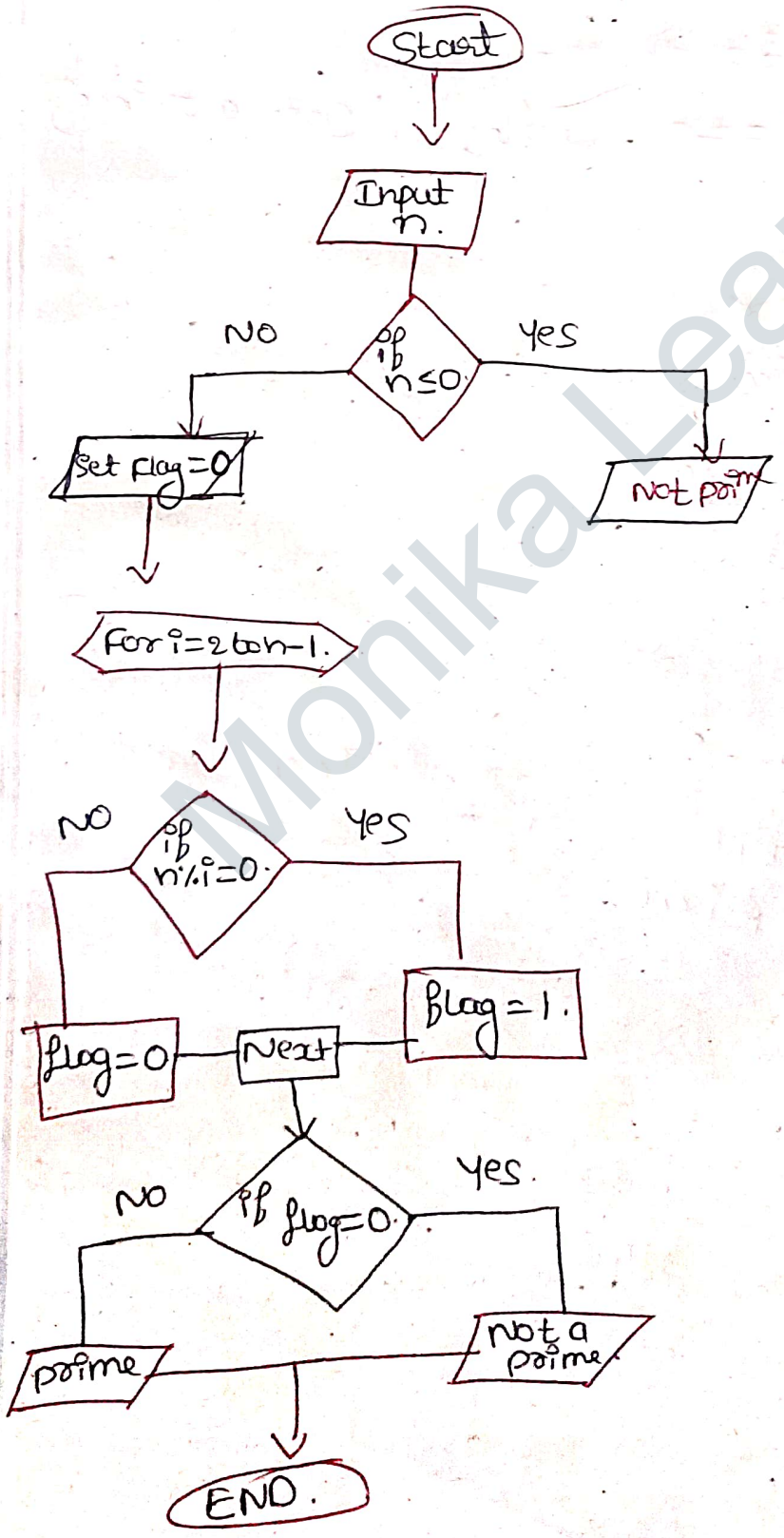
```
BEGIN.
 READ n
 If n<=0. THEN
    PRINT NOT prime.
    Stop
 END IF

 flag <-0
 for i<- 2 to n-1 DO.
   if n mod i = 0 THEN.
        flag <-1.
        BREAK.
   ENDIF.
 END FOR.
```

(not a prime.

```
IF flag=0 THEN
  PRINT "prime"
ELSE
  PRINT "NOT prime"
END IF
STOP.
```

Flowchart



(4) Calculate the factorial of a given number.

Algorithm

Step 1: Start
Step 2: Input n.
Step 3: Set fact = 1.
Step 4: Repeat from i = 1 to n.
→ fact = fact * i.
Step 5: print fact.
Step 6: Stop.

pseudocode

```
BEGIN
  READ n.
  fact ← 1.
  FOR i = 1 TO n DO
    fact ← fact * i.
  END FOR.
  PRINT fact
  END.
```

Flowchart