UNIT 2

For XI CSc:

COMPUTATIONAL THINKING AND PROGRAMMING-1

Introduction to Problem Solving

CONTENTS

(Learning Outcomes) ☺

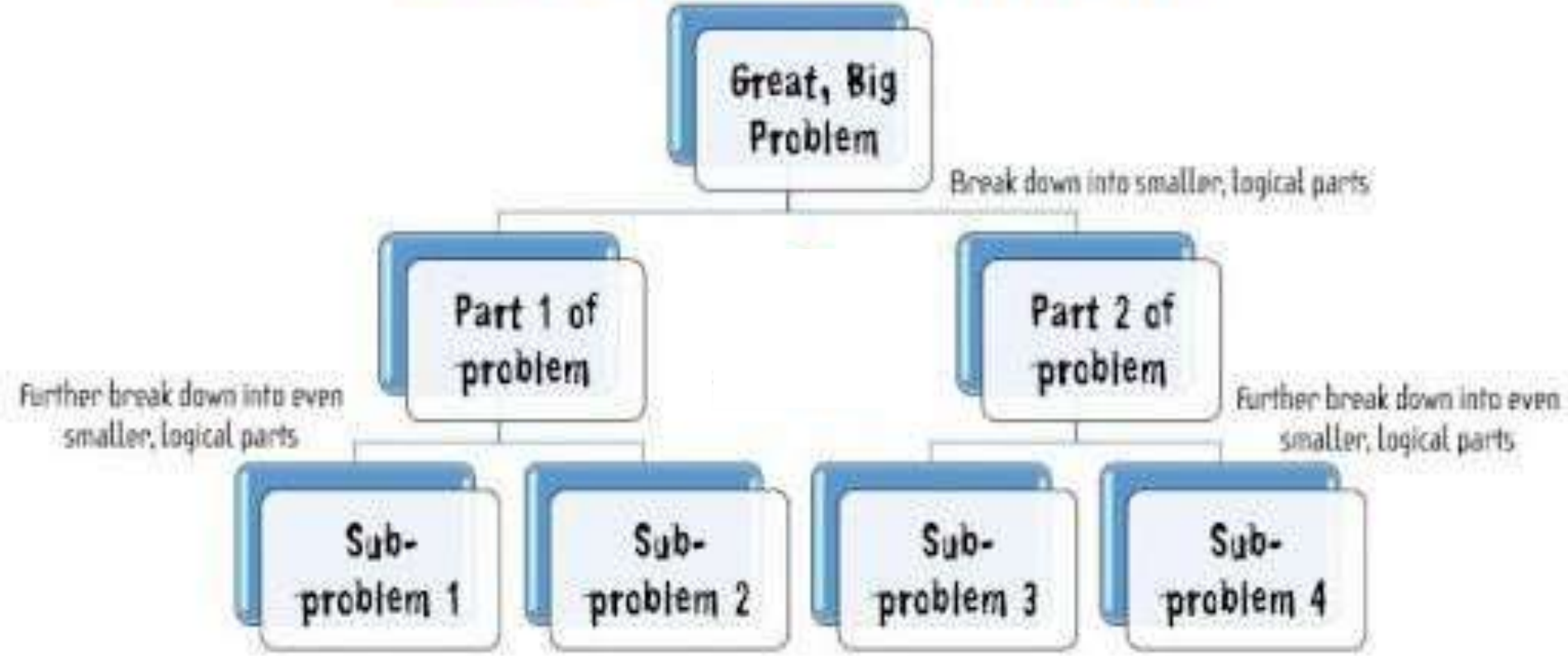
- Introduction to Computational Thinking and Programming-1
- Problem solving
- Problem Solving Cycle
- Algorithm, Flowchart, Pseudo-code
- Concept of Problem Decomposition
- Examples of Problem Decomposition

What is Computational Thinking?

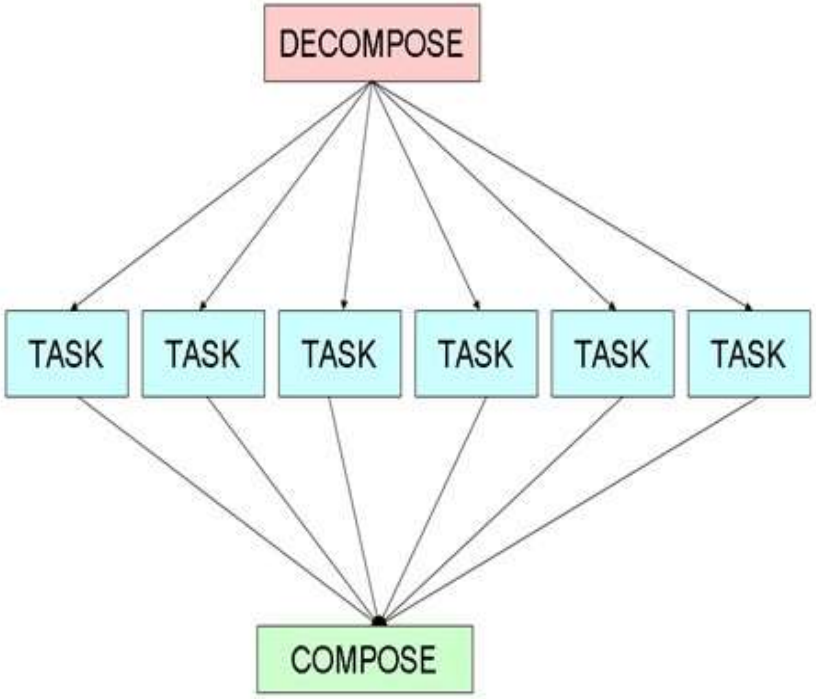
Computational Thinking (CT) is a set of problem-solving methods that involve expressing problems and their solutions in ways that a computer could also execute. It involves the mental skills and practices for designing computations that get computers to do jobs for people, and explaining and interpreting the world as a complex of information processes.

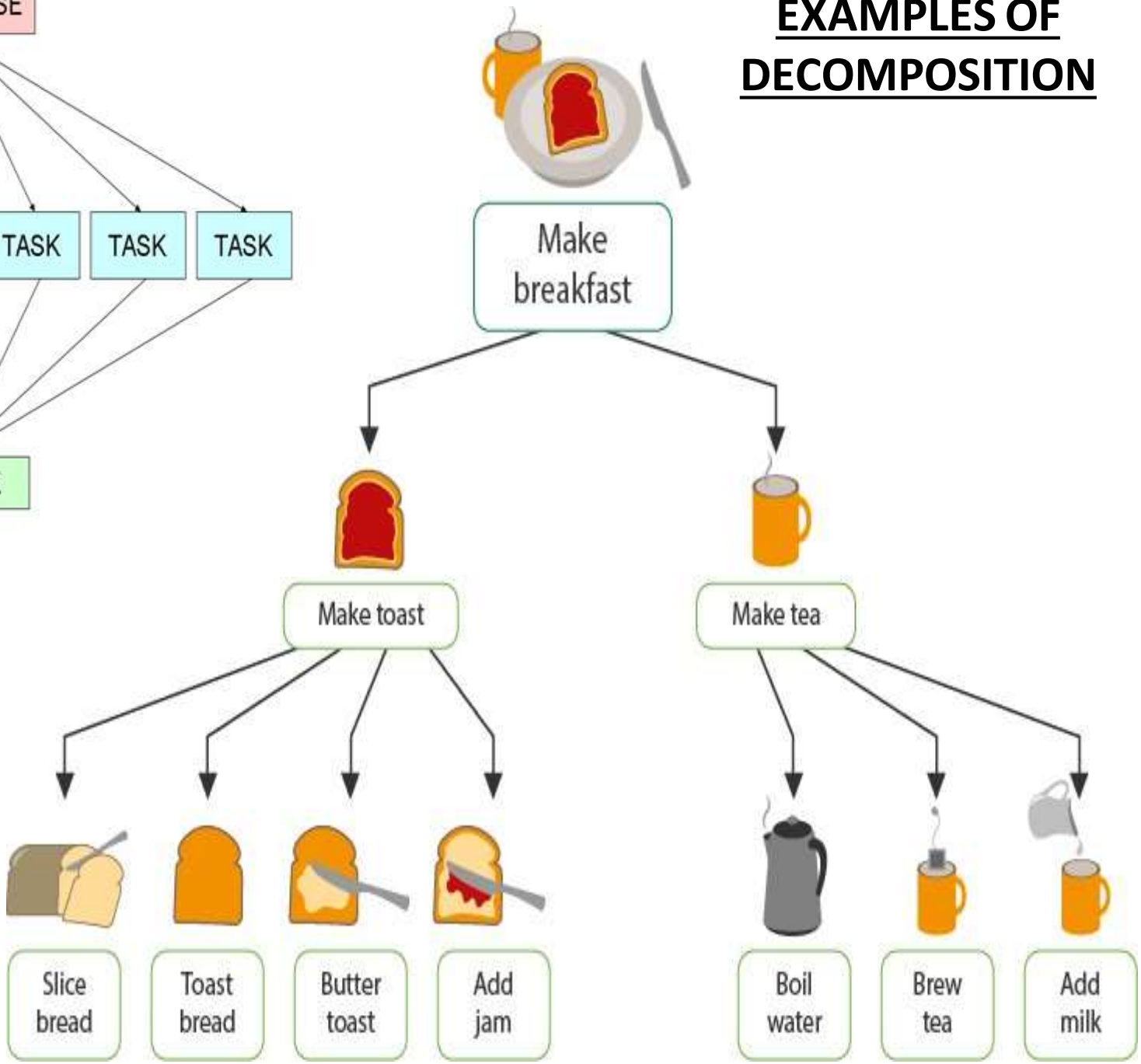
DECOMPOSITION

- Decomposition involves **breaking down a complex problem or system into smaller parts** that are more manageable and easier to understand.
- The smaller parts can then be examined and solved, or designed individually, as they are simpler to work with.



EXAMPLES OF DECOMPOSITION

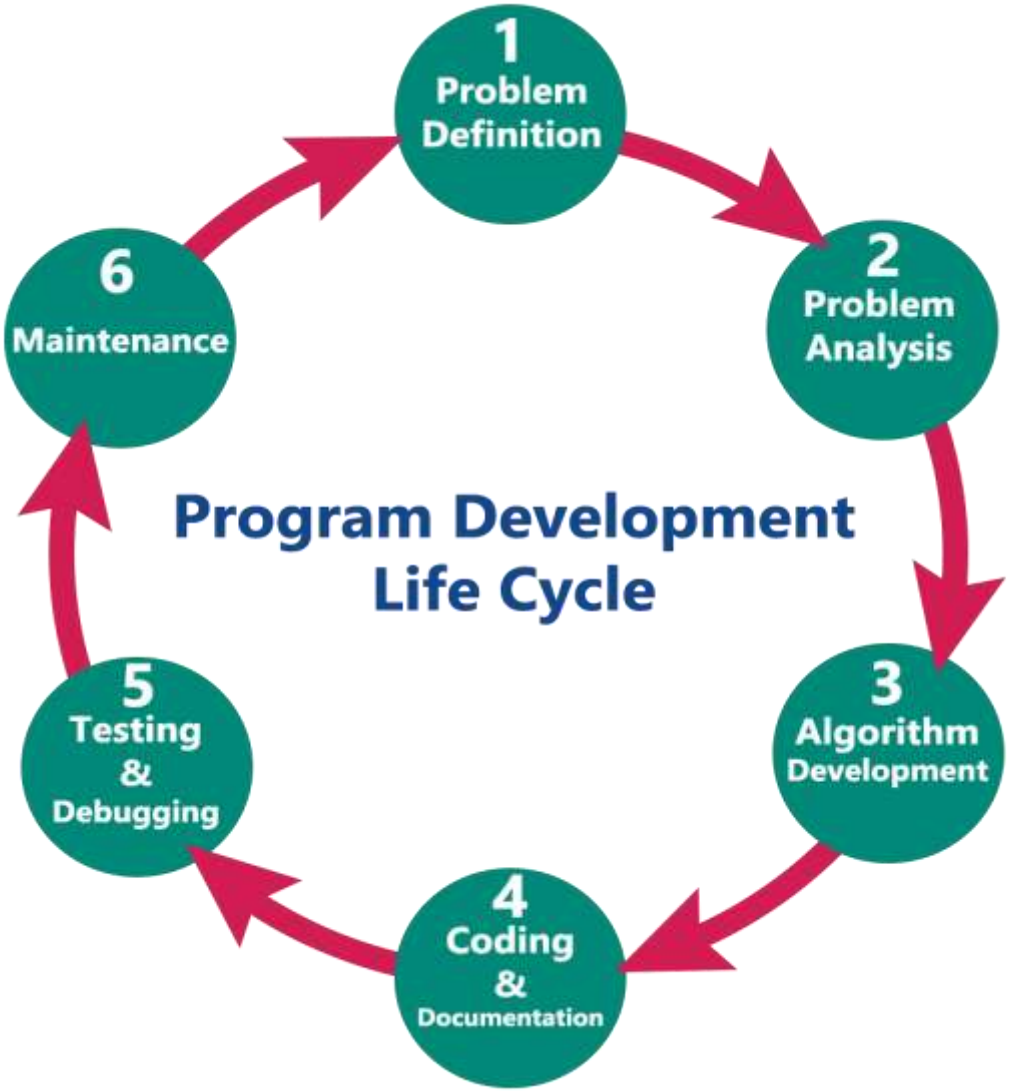




WHAT IS PROBLEM SOLVING ?

Precise step-by-step instructions given by us to solve a problem is known as Problem Solving.

Thus, **problem solving is the process of identifying a problem, developing an algorithm for the identified problem and finally implementing the algorithm to develop a computer program to solve the problem.**



PROBLEM SOLVING METHODOLOGY AND TECHNIQUES

Steps to create a working program

1. Understand the problem well
2. Analyze the problem to
   a) Identify minimum number of inputs required for output
   b) Identify processing components.
3. design the program by
   a) Deciding step by step solution.
   b) Breaking down solution into simple steps.
4. Code the program by
   a) Identifying arithmetic and logical operation required for solution.
   b) Using appropriate control structure such as conditional or looping control structure.
5. Test and Debug your program by
   a) Finding error in it.
   b) Rectifying the error.
6. Complete your documentation.
7. Maintain your program.

PROGRAM LOGIC DEVELOPMENT TOOLS

ALGORITHM:- An algorithm is a step-by-step procedure (well-defined instructions) to solve a given problem.

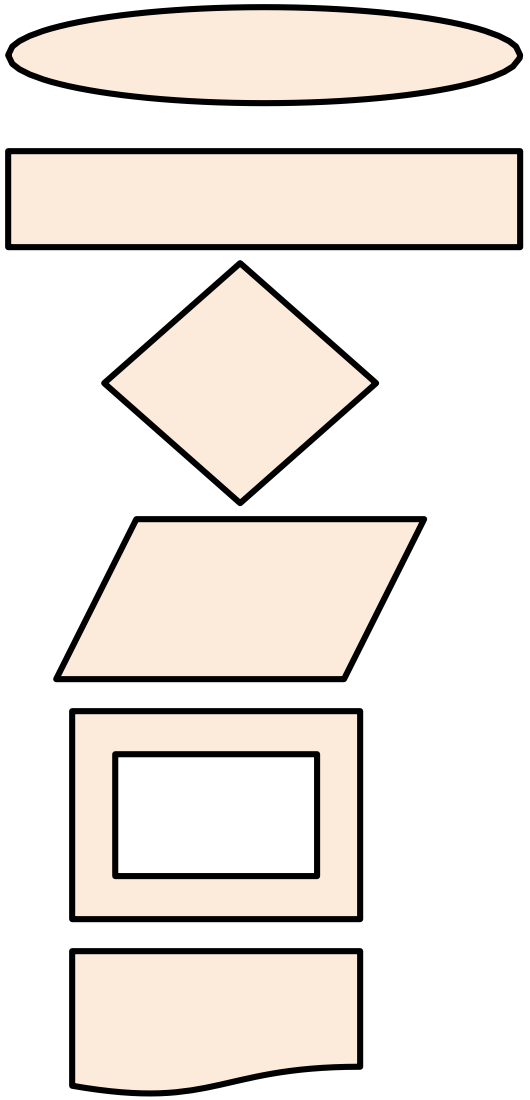
Eg.

Algorithms are written out with tools like

pseudocode, flowcharts, decision trees.

Flowcharts

A flowchart is a graphical representation of steps of an algorithm to solve a given problem.



START / END

PROCESS

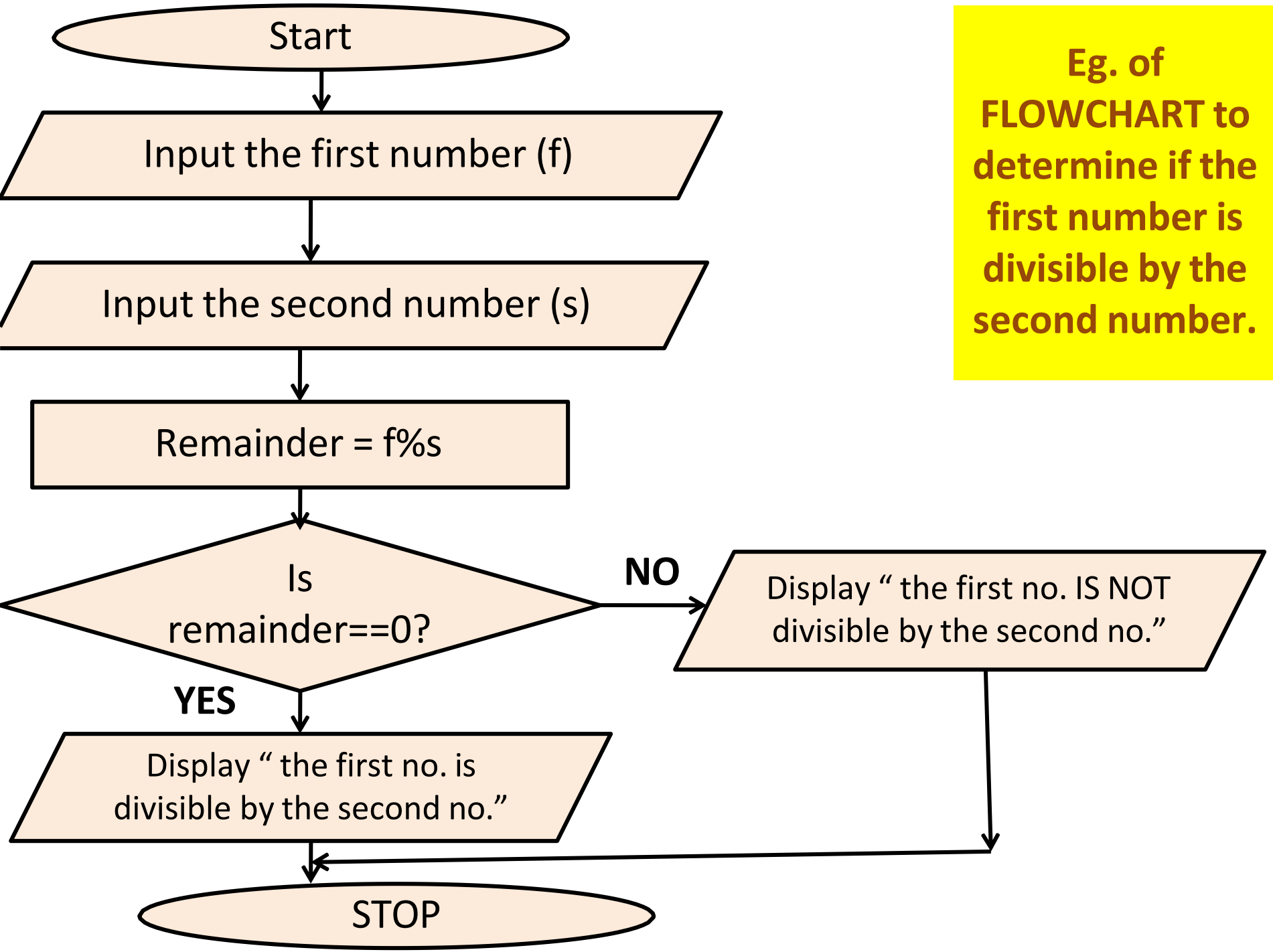
DECISION / Condition

DATA (Input/Output)

SUB PROCESS / USER DEFINED TASK

DOCUMENT

Eg. of FLOWCHART to determine if the first number is divisible by the second number.



PSEUDOCODE

- **Pseudocode is an informal way of describing the steps of a program's solution without using any strict programming language syntax or underlying technology considerations.**
- **It uses simple English and highlights the sequence of instructions.**

Eg. Write the pseudocode to determine is the first number is divisible by the second number or not.

Ans.

Input first number in variable firstnum.

Input second number in variable secondtnum.

Remainder= firstnum modulus secondnum

If the remainder is 0

    display 'the first number is divisible by the second number'.

else

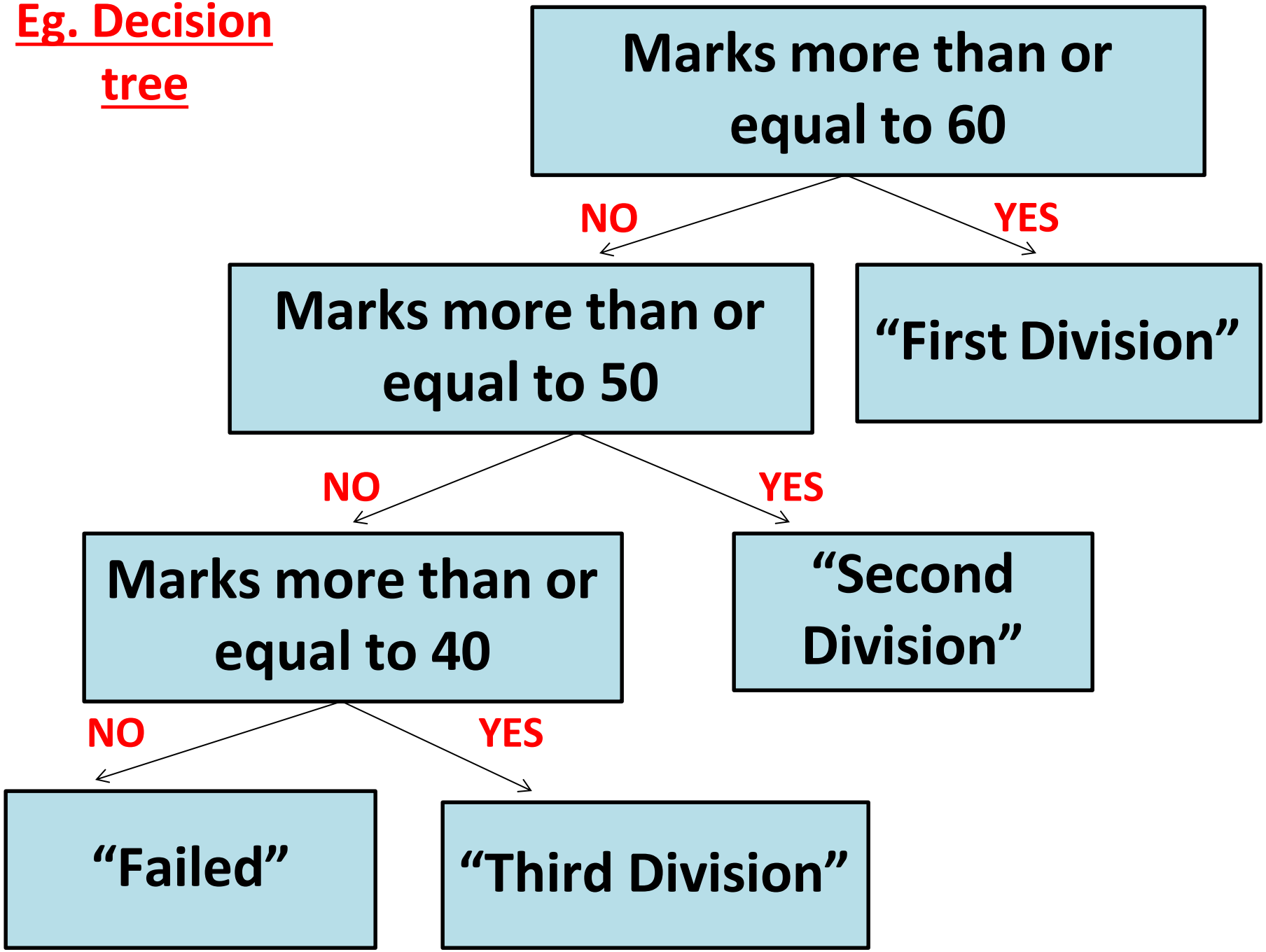
    display 'the first number IS NOT divisible by the second number'.

DECISION TREES

Decision trees are a way of presenting rules in a hierarchical and sequential structure, where based on the hierarchy of ***rules***, certain ***outcomes*** are predicted. Eg.

MARKS	DIVISION
>=60	First
50-60	Second
40-50	Third
<40	Failed

Eg. Decision tree



PSEUDOCODE

VERSUS

FLOWCHART

PSEUDOCODE	FLOWCHART
An informal high-level description of the operating principle of an algorithm	A diagrammatic representation that illustrates a solution model to a given problem
Written in natural language and mathematical notations help to write pseudocode	Written using various symbols

DEBUGGING PROGRAMS

What is Testing and Debugging?

Testing refers to check for an error or bug in the program code.

Debugging refers to the process of locating the place of error, cause of error, and correcting the code accordingly.

→**Errors**

→**Exceptions**

Errors and Exceptions

ERRORS

Error is a bug in the code that causes irregular output or stops a program from executing.

Error prevents the program from compiling and running correctly.

Errors in a program can be fixed by making corrections in the code.

Types of Errors:-

1) **Compile time Errors**
   - **Syntax Error**

2) **Runtime Errors**
3) **Logical Errors / Semantics Error (In few of the programming Languages)**

EXCEPTIONS

Exception is an irregular unexpected situation occurring during execution on which programmer has no control.

Fixing exception is tougher than fixing errors. Exceptions are written in try and except block.

Some Built-in Exceptions are:-

1. **EOF Error**
2. **IOError**
3. **NameError**
4. **IndexError**
5. **ImportError**
6. **TypeError**
7. **ValueError**
8. **ZeroDivisionError**
9. **OverflowError**
10. **KeyError**

Types of Error

- **Compile Time Error-** Occurs during compile time. When a program compiles, its source code is checked for whether it follows the programming language's rules or not.

Its types are:-

1. **Syntax Errors :** it occurs when a grammatical rule of the Programming language is violated

Eg.

```
if a=>b          # Instead of ==(relational operator) =>
print(a)         #has been used, which is syntactically wrong.
                 #Moreover, : has not been given after the if condition.
```

2. **Semantic errors:** it occurs when statements are not meaningful.

Semantics refers to the set of rules which give the meaning of a statement.

```
a*b=c        # Statement is not meaningful…… Semantic Error !!!!!
```

➢ Run Time Error:

It occurs during the execution of the program. These errors are harder to detect (trap).

Some run time errors stop the execution of the program which is known as program “crashed” or “abnormally terminated”.

Eg.

An infinite loop or wrong input value (giving string instead of int, which is required)

➢ Logical Error:

It occurs due to wrong logic of a program. We don’t encounter any error during compile-time or run-time, but still program does not provide the correct result. This is done by the programmer’s mistake.

Eg.

Trying to print the multiplication table , but instead of a*i, you have written a+i , which shows a wrong result by adding the nos. rather than multiplying.

Some Built – in Exceptions	Description
EOFError	Raised when one of the built-in functions (input()) hits an end-of-file condition (EOF) without reading any data.
IOError	Raised when an I/O operation (such as a **print**(), the built-in open() function or a method of a file object) fails for an I/O-related reason, e.g., *"file not found"* or *"disk full".*
NameError	Raised when an identifier name is not found.
IndexError	Raised when a sequence subscript or index is *out of range*, e.g., from a string of length 4 if you try to read a value of index like 4 or more i.e., string[4], string[5], string[-5] etc. will raise exception as legal indexes for a string of length 4 are 0, 1, 2, 3 and -1, -2, -3, -4 only.
ImportError	Raised when an **import** statement fails to find the module definition or when a **from_import** fails to find a name that is to be imported.
TypeError	Raised when an operation or function is applied to an object of inappropriate type, e.g., if you try to compute a square-root of a string value.
ValueError	Raised when a built-in operation or function receives an argument with inappropriate value e.g., int("z10") will raise ValueError.
ZeroDivisionError	Raised when the second argument of a division or modulo operation is zero.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
KeyError	Raised when a mapping (dictionary) key is not found in the set of existing keys.

HOW TO DEBUG A PROGRAM

- Debugging involves correction of code so that the cause of errors is removed.
- **Testing is done to verify correct behavior i.e., with some sample values (test cases) whose output is already known, the code is tested – if it produces the anticipated output, code is correct and if it does not, there is some error. Once you know the errors, you can debug your program.**

Some useful debugging techniques.

Debugging Techniques

Debugging a program is a skill. There are many traditional debugging techniques that you can allow to debug your code. These are:

1. **Carefully spot the origin of error.**
2. **Print variables' intermediate values.**
3. **Code tracing and stepping.**