

Develops algorithms to solve problems and uses python programming language to encode algorithms

- What is Problem Solving Process?
- Why need Problem Solving?
- Who Solve the problem?
- The Steps of Problem-Solving Process

What is Problem Solving Process?

- **Problem solving is the sequential process** of analyzing information related to a given situation and generating appropriate response options.
- The **problem-solving process starts** with the **problem specification** and **ends** with a **concrete (and correct) program**.



Why need Problem Solving?

- A computer is a very powerful and versatile (multipurpose) machine capable of performing a multitude of different tasks
- It has no intelligence or thinking power.
- The intelligence Quotient (I.Q) of a computer is zero.
- This places responsibility on the user to instruct the computer in a correct and precise manner,
- So that the machine is able to perform the required job in a proper way.
- A wrong or ambiguous(unclear) instruction may sometimes prove disastrous (terrible).

Who Solve the problem?

- A computer cannot solve a problem on its own.
- One has to provide step by step solutions of the problem to the computer.



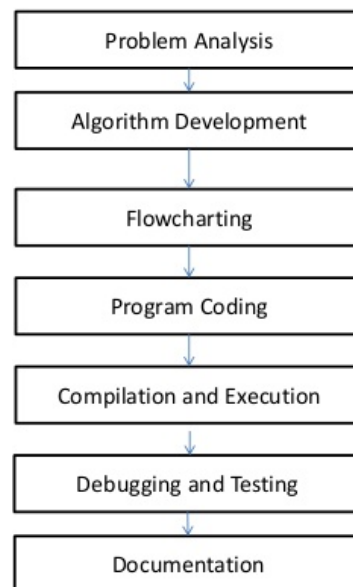
- In fact, the task of problem solving is not that of the computer.
- It is the programmer who has to write down the solution to the problem in terms of simple operations which the computer can understand and execute.

The Steps of Problem-Solving Process

In order to solve a problem by the computer, one has to pass through certain stages or steps.

They are,

1. Understanding the Problem
2. Defining the problem and boundaries
3. Planning solution
4. Implementation



Understanding the Problem

- Here we try to understand the problem to be solved in totally. Before with the next stage or step, we should be absolutely sure about the objectives of the given problem.
- Defining /Specifying the problem by answering to questions as:
 - What the computer program do?
 - What takes will it perform?
 - What kind of data will it use, and where will get its data from?
 - What will be the output of the program?
 - How will the program interact with the computer user?



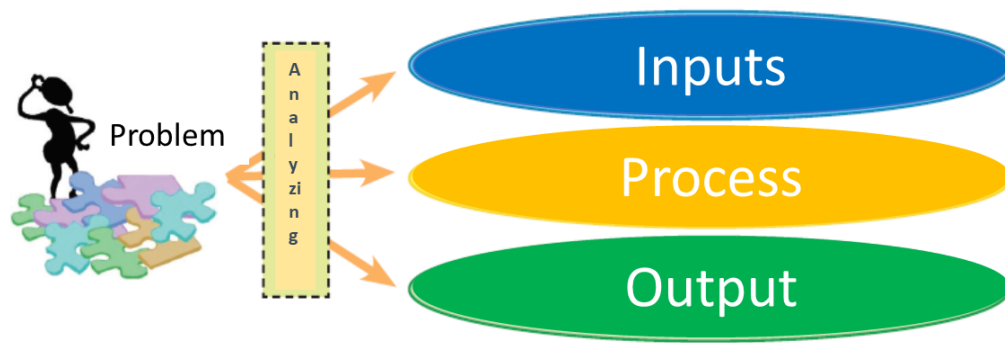
Analyze the problem

- to study or examine something carefully in a methodical way
- to separate (a thing, idea, etc.) into its parts
- Easy to understand the problem
- When analyzing the problem, the input, process and output are identified separately



Inputs, Outputs and Process ????

- Inputs – The raw materials that are used to solve a problem
- Outputs – The result obtained after solving a problem
- Process – Converting input into output
- A process takes place step by step and it is very important to understand the order of the process.



Problem 1 : Preparing a letter which can be posted.

Input : A sheet of paper suitable to write the letter on and a pen
An envelope and a stamp
Glue

Process :

1. Writing the letter
2. Folding the letter and putting it into the envelope
3. Pasting the envelope
4. Writing the recipient's address on the envelope
5. Sticking the stamp



Output : A letter ready to be posted.

Note: Steps No. 4 and 5 in this process can be interchanged.
However, the other steps should be followed in the order indicated.

Problem 2 : Making a cup of tea

Input : Tea leaves, sugar, hot water

Process :

1. Putting tea leaves in the strainer
2. Pouring hot water to the cup through the strainer
3. Adding some sugar to the cup
4. Stirring it well with a spoon
5. Testing for taste, taking a small sip from the cup
6. If the taste is not satisfactory, go to step 3 and repeat step Nos. 4 and 5

Output : A cup of tea



Problem 3 : Dividing 40 page and 80 page books from a parcel of books between two siblings - Sanduni and Anupama.

Input : The parcel of books

Process :

1. Opening the book parcel
2. Taking a book out from the parcel
3. If it is a 40 page book, giving it to Sanduni
4. If it is a 80 page book, giving it to Anupama
5. Go to Step No. 2 till all the books are taken out of the parcel

Output : Sanduni getting 40 page books
Anupama getting 80 page books



Problem 4 : Adding two numbers

Input : Two numbers

Process : Adding the two numbers

Output : Total

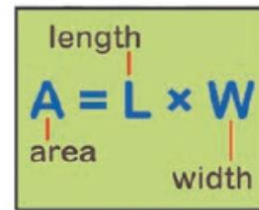


Problem 5 : Finding the area of a rectangle

Input : Length and width of the rectangle

Process : Area = Length x Width

Output : Area



Problem 6 : Finding the larger number between two numbers

Input : Two numbers

Process : Comparing the two numbers, finding the larger one

Output : Larger number

Problem 7 : Finding whether a number is odd or even

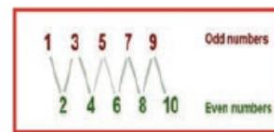
Input : Number

Process : Dividing the number by 2

Deciding that the number is even if the remainder = 0

Deciding that the number is odd if the remainder = 1

Output : Indicating whether the number is odd or even



Defining the problem and boundaries

- If we have any problem, have to developed computerized Information System.
- For that Need to identify -
 - What are the data insert into the system?
 - What Information that taken from the system?
 - Who are the parties need those information?
- For that need to identify boundaries of our system
 - Identify external parties which are related to the system



Planning solution

- If there is more than one solution to a given problem, such solutions are called **alternative solutions**.
- Such solutions depend on the nature of the problem.
- Thus, if there are many solutions (set) to a particular problem, it is suitable to consider these and select most appropriate solution.
- All the solutions pertaining to a problem are called **solution space**.



Example 1

Let us examine the solution space to find the perimeter of a rectangle.

Let us analyze the input, process and output related to this problem.

Input : Length and width of the rectangle

Process : Calculating the perimeter

Output : Indicating the perimeter

Let us examine the solution space to calculate the perimeter.

1st solution Perimeter = length + width + length + width

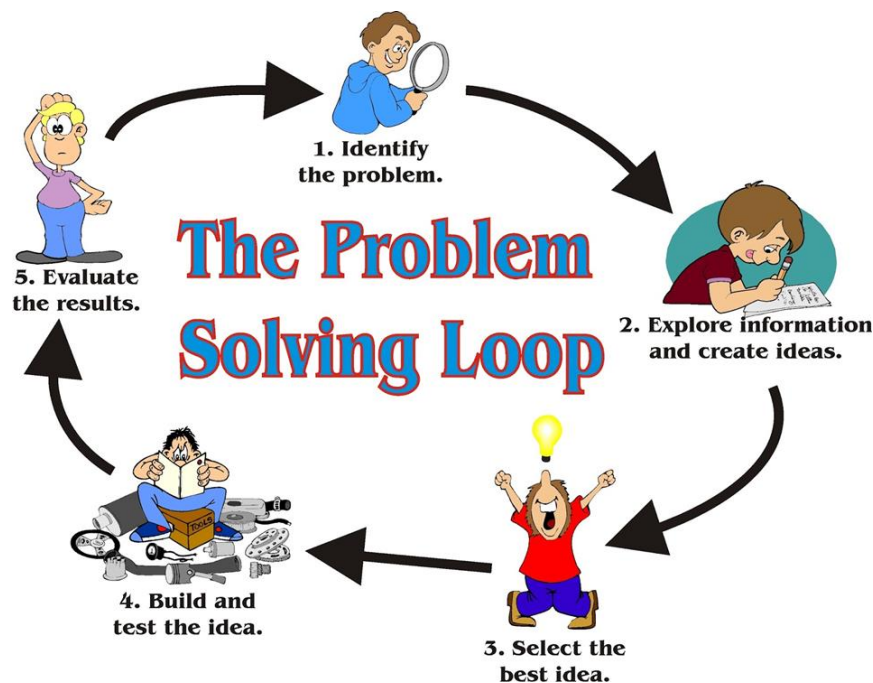
2nd solution Perimeter = length \times 2 + width \times 2

3rd solution Perimeter = (length + width) \times 2

Out of these solutions, a person who has knowledge only of addition, can select the 1st solution as the most appropriate. A person who has knowledge of multiplication and addition can select the 3rd solution out of the 2nd and 3rd solutions as the most appropriate. The reason for this is, it has the minimum number of additions and multiplications.

Implementation

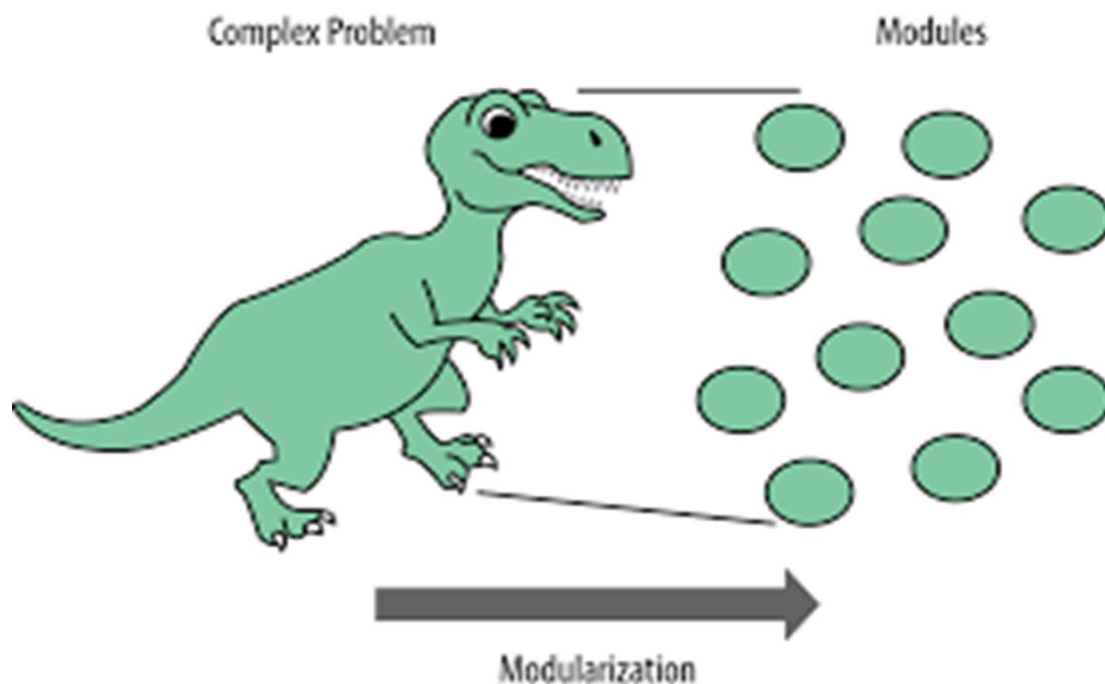
- In computer programming, A **programming language implementation** is a system for executing computer programs.
- Once the programmer has properly finished developing, coding and testing the program, the task is almost over. Now the project team will release the program to play its part in the overall system – the programmer has just some tidying up to do



The Top Down and Stepwise Refinement Methodologies in Solving Problems

- Modularization
- Top Down Design and Stepwise Refinement
- Structure Charts

Modularization

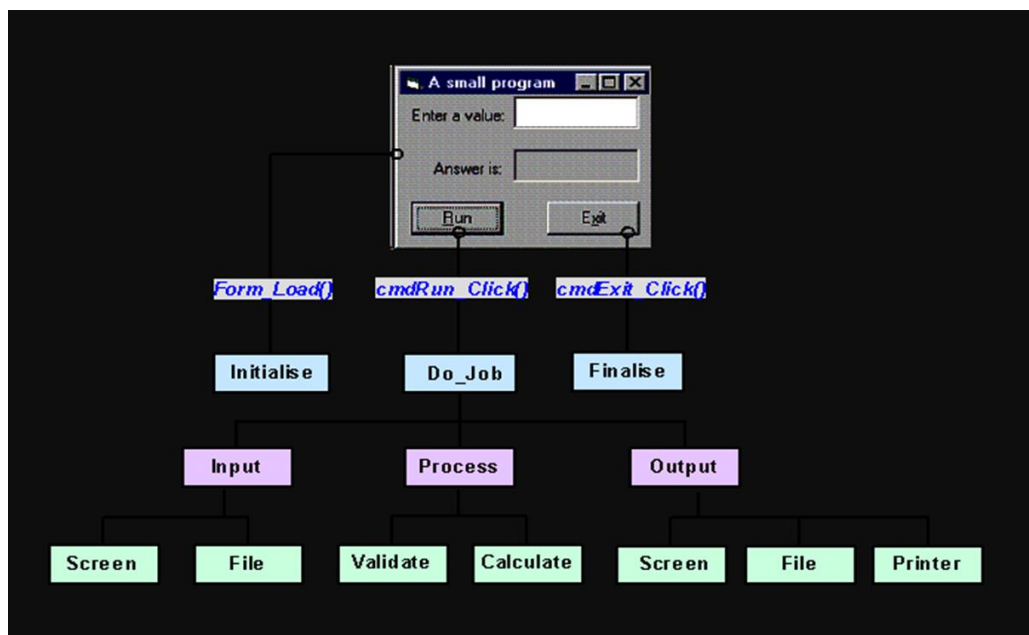
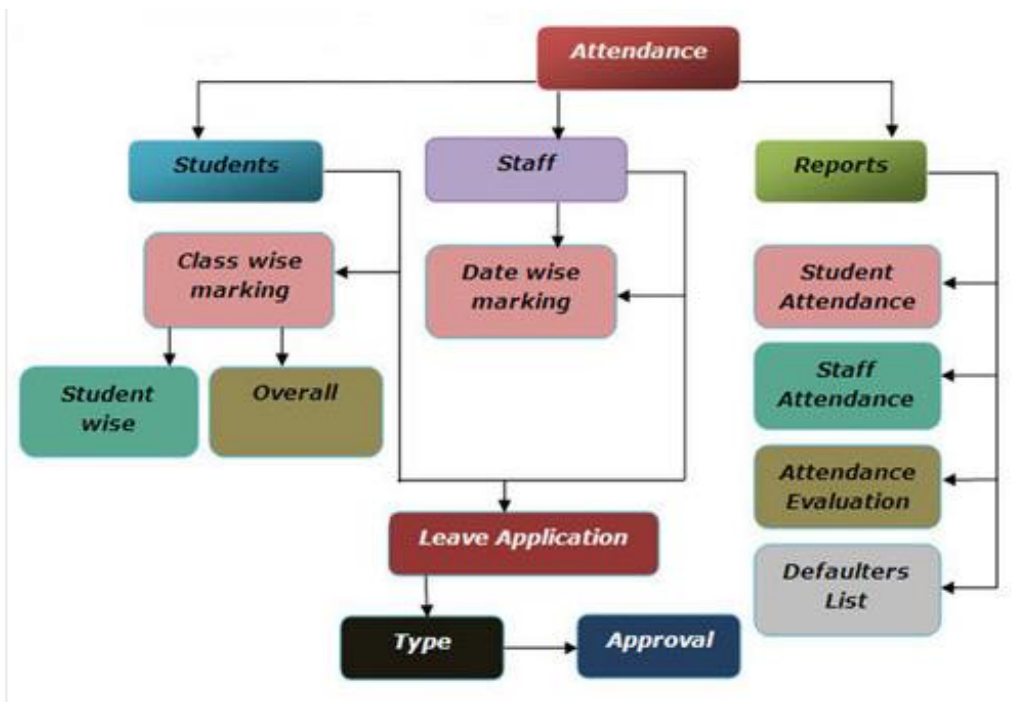


- Modularization is breaking a large Program into modules (Smaller chunks).
- Breaking a large problem into understandable chunks, cannot reduce the overall complexity of a system.
- But breaking it down into smaller parts means that the programmer has to understand one module is less effort than if dealing with the system as a whole.

Ex: Consider a Payroll Management System of a Private Company

- A separate unit for enter the attendance of employees
- A separate unit for enter Overtime details and calculate OT.
- Like wise after creating separate units for separate tasks,
- Finally links all separate units together and possible to run as single task on a screen

Ex: Think about your School Management System



Advantages of Modularizations

- Makes code manageable.
 - "Safer" code.
 - Code is easier to read & understand.
 - Reduces maintenance overheads.
 - Allows re-cycling/re-use of code.
-
- Can **reuse modules**, which contain standard procedures throughout the program, **saving development time**.
 - Testing of individual modules, in isolation, makes tracing mistakes easier.
 - Can be done **amendments** to single modules which do not affect the rest of the program.
 - Even **can assigned more than one programmers** to developed a program.
 - The checking of conditions and errors in the modules, **ensuring consistency (quality) and completeness**.

Disadvantages of Modularizations

- Too many modules can reduce performance (speed).
- Excessive and/or poor modularization can lead to code that is difficult to read.

Top Down Design and Stepwise Refinement

- Top-down design (also known as stepwise design) - in which design begins by specifying complex pieces and then dividing them into successively smaller pieces and repeat until no further breakdown is possible.
- The result is a hierarchically distinct, manageable component, whose relationships and levels are clear and simple to see.

Example: Making Pancakes

- To show you an example of Top-Down design I'll take the act of making pancakes.
- Starting from the top we have the task to:
- Make some pancakes
- But saying that by itself isn't enough to actually make any pancakes, we need to break the task down:

Make some pancakes

- Organize Kitchen
- Make Pancakes
- Serve

Each of these tasks can then be broken down further:

Organize Kitchen

1. Clean surfaces
2. Get out mixing bowl, whisk, spoon, sieve
3. Get out plain flour, salt, eggs, full fat milk, butter
4. Put on apron

Make Pancakes

1. Sift salt and flour into bowl
2. Break eggs into bowl
3. Whisk
4. Add water and milk
5. Add butter
6. Whisk
7. Cook

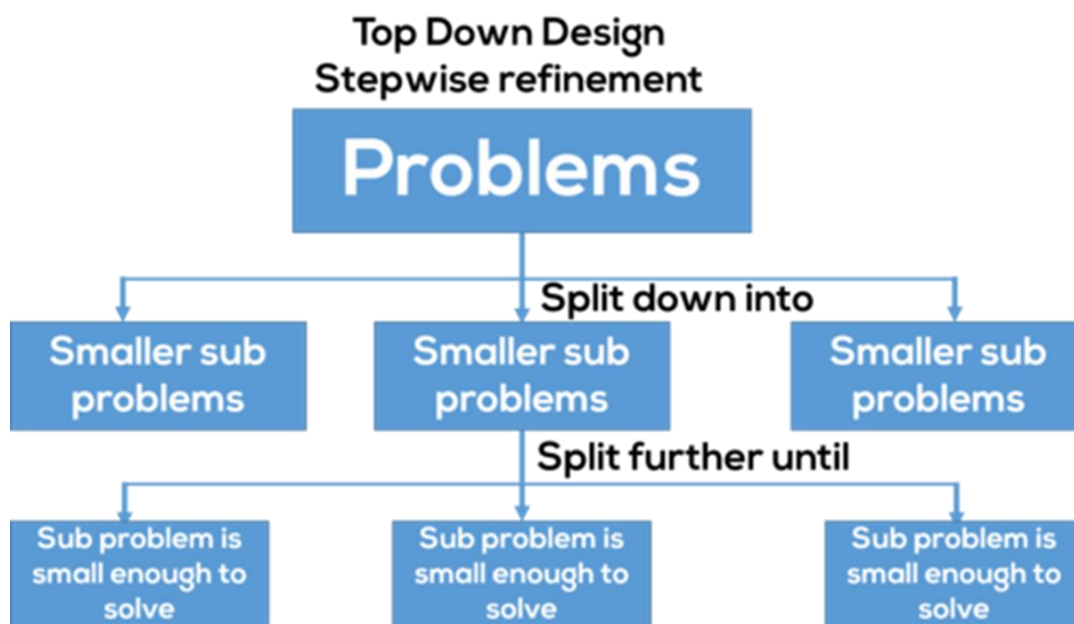
Serve

And each of these tasks can be broken down further, let us take a look at the Cook:

Cook

1. Get pan to temperature
2. Pour batter in
3. Spread batter to edges
4. Use plastic spatula to check bottom of pancake
5. When brown, flip
6. Use plastic spatula to check bottom of pancake
7. When brown finish

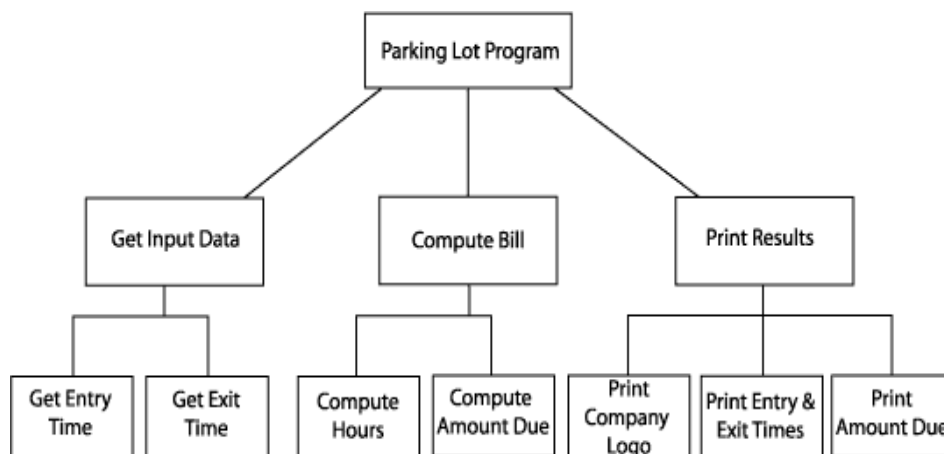
- We can break down some of these tasks even further. So starting at a single point, the creation of a pancake, we have broken down the task into its individual parts.
- Top-down design involves looking at the whole task and breaking it down into smaller, more manageable sub-problems which are easier to solve. These sub-problems can be divided even further into smaller steps. This is called stepwise refinement.



Structure Charts

- A structured chart **graphically illustrates the structure of a program** by showing independent hierarchical steps.
- Since the components are pictured in hierarchical form, a drawing of this kind is also known as **hierarchy chart**.
- A structure chart is **easy to draw** and **easy to change**, and it is **often used to supplement or even to replace a logic flowchart**.

A structure chart of a simple parking lot billing program



Uses Algorithmic Approach to Solve Problems

- Algorithms
 - Flowcharts
 - Pseudo Codes
- Hand Traces



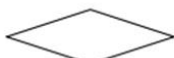


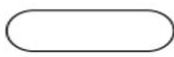

Algorithms

- **An algorithm is a detailed series of instructions for carrying out an operation or solving a problem.**
- This can be
 - a simple process, such as multiplying two numbers, or
 - a complex operation, such as playing a compressed video file (mp3).
 - Search engines use proprietary algorithms to display the most relevant results from their search index for specific queries.
- In a non-technical approach, we use algorithms in everyday tasks,
 - such as a recipe to bake a cake or a do-it-yourself handbook.
- In other word, **an algorithm is a Complete step-by-step procedure to solve a given problem.**
- In the problem-solving phase of computer programming, you will be designing algorithms. This means that you will have to be conscious of the strategies you use to solve problems in order to apply them to programming problems. These algorithms can be designed though the use of **flowcharts or pseudocode.**

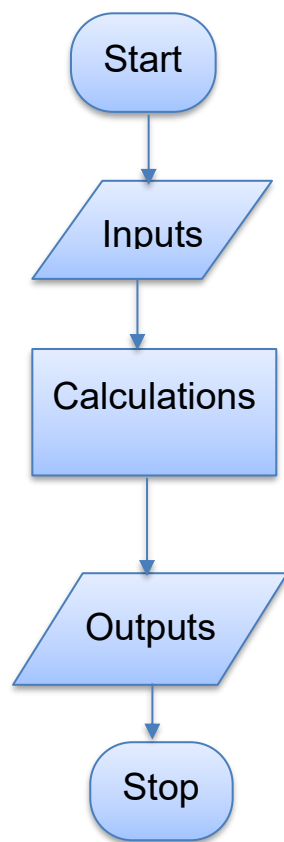
Flowcharts

- **A flowchart is a diagram** made up of boxes, diamonds and other shapes, connected by arrows.
- Each shape represents a step in the process, and the arrows show the order in which they occur.
- Flowcharting combines symbols and flowlines, to show figuratively the operation of an algorithm.
- **It is a step-by-step Diagrammatic representation of the program**
- **Each type of task is represented by a symbol**

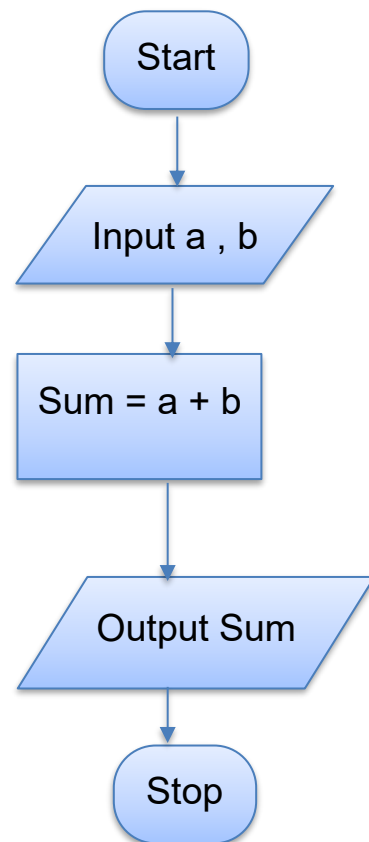
Symbols commonly used in flowcharting...

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program.
	Flow Lines	Shows direction of flow.

A Typical Flowchart

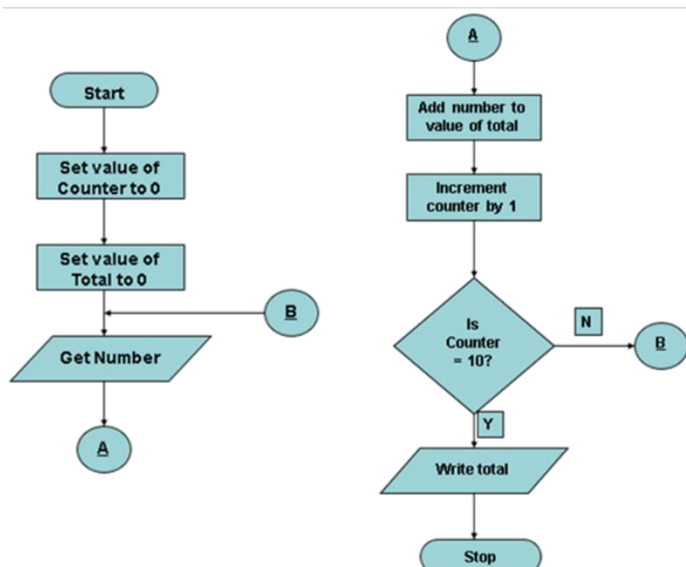


Eg :



Connectors

- When a flowchart is too long to fit on a page and is to be continued on another page a connector symbol is used



- A small circle is used to represent a connector in a flowchart
- An alphabet or a number is written inside the circle to identify the pairs of connectors to be joined

Pseudo Codes

- Pseudo codes use every day language...**to prepare a brief set of instructions**...in the order...in which they will appear in a finished program
- It is an **abbreviated version of actual computer code** (that's why it is called Pseudocode)
- A computer **cannot execute Pseudocode**
- Once pseudocode is created, **it is simple to translate into real programming code.**

Rules for writing Pseudocodes

- **Write one statement per line.**
- **Capitalize Initial Keywords:** Primarily **READ, WRITE, IF, ELSE, ENDIF, WHILE, ENDWHILE, REPEAT, UNTIL** are the keywords that must be written in **capital letters**.
- **Indent to show hierarchy:** Indent lines to make the Pseudocode easy to read and understand.
- **End multiline structures:** Every **IF** must be end with an **ENDIF**. Every **DO, DO WHILE** must end with **ENDDO**.
- **Keep statements language independent:** This rule does not mean that we can write our pseudocode in English, French, Russian, Chinese or whatever languages we know. **It refers to programming language.** Try to avoid the use of words peculiar (abnormal) to any programming language.

Example for a Pseudocodes

- Write a pseudo code that inputs two numbers (a and b) and calculates the sum of the numbers and output the sum

```
INPUT a
INPUT b
sum = a + b
OUTPUT sum
```

ADVANTAGES AND DISADVANTAGES

Pseudocode Disadvantages

- It's not visual
- There is no accepted standard, so it varies widely from company to company

Pseudocode Advantages

- Can be done easily on a word processor
- Easily modified
- Implements structured concepts well

Flowchart Disadvantages

- Hard to modify
- Need special software (not so much now!)
- Structured design elements not all implemented

Flowchart Advantages

- Standardized: all pretty much agree on the symbols and their meaning
- Visual (but this does bring some limitations)

The Logical Constructs

- Sequence
- Selection
- Iteration

SEQUENCE is a linear progression where one task is performed sequentially after another.

Pseudocode:

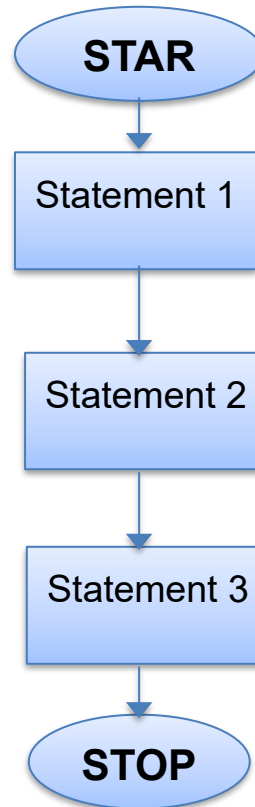
BEGIN

statement 1

statement 2

statement 3

END



Find the sum and average of two numbers.

BEGIN

INPUT a

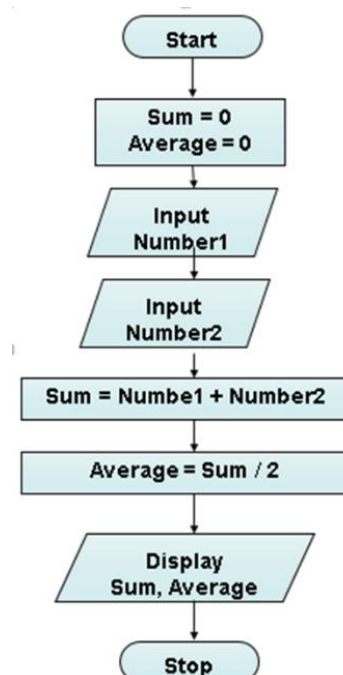
INPUT b

Sum = a + b

Average = Sum/2

OUTPUT Sum, Average

END



SELECTION - there may be alternative steps that could be taken subject to a particular condition

IF-THEN-ELSE is a decision (selection) in which a choice is made between two alternative courses of action

Select one path according to the condition

IF THEN

- If the condition is true do the statements inside IF
- No operation if the condition is false

IF THEN ELSE

- If the condition is true do the statements inside IF
- If the condition is false do the statements inside ELSE

IF condition

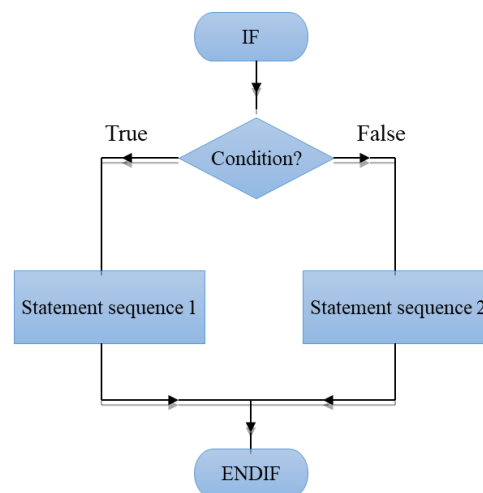
THEN

sequence-1(statements)

ELSE

sequence-2(statements)

ENDIF



- *Example1:*
IF $a > 0$ THEN
 Print a
END IF

- *Example2:*
IF $a > b$ THEN
 Print a
ELSE
 Print b
END IF

Ex: Find the largest among three different numbers entered by user.

BEGIN

DECLARE Variables a, b, and c

READ Variables a, b, and c

IF a>b

IF a>c

DISPLAY a is the largest number

ELSE

DISPLAY c is the largest

ELSE

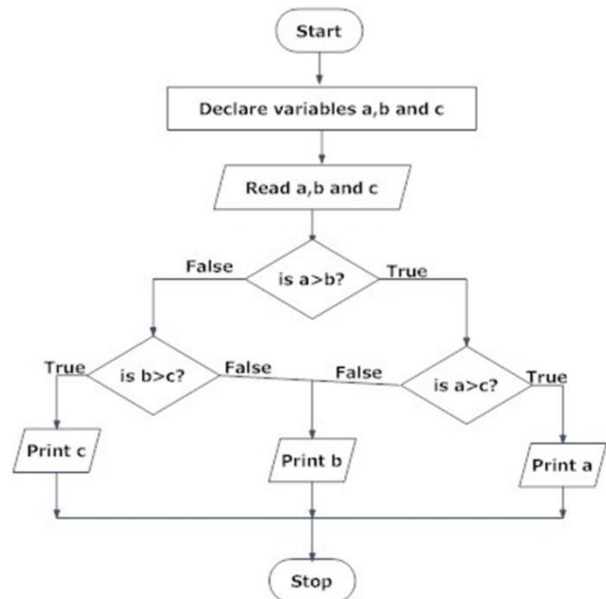
IF b>c

DISPLAY b is the largest

ELSE

DISPLAY c is the largest

END



ITERATION - certain steps may need to be repeated while, or until, a certain condition is true

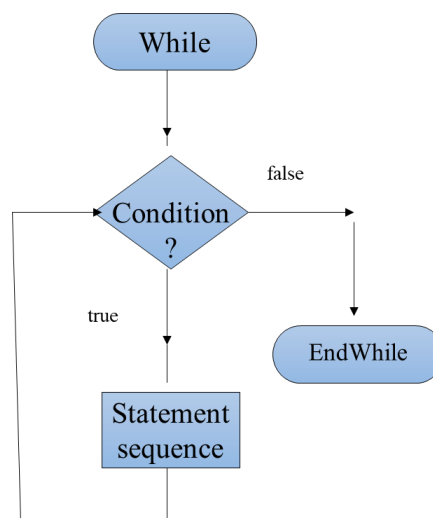
- While
- For
- Repeat

While

Pseudocode

```

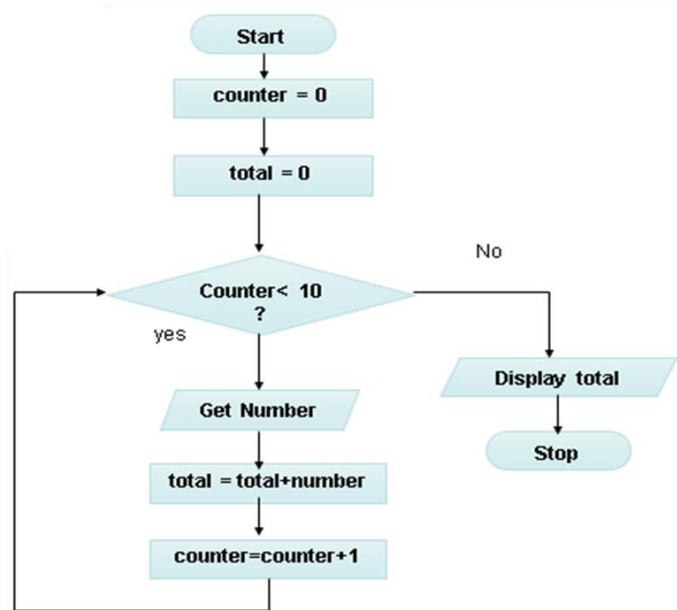
WHILE    <Condition>
    Statement- Sequence
END WHILE
  
```



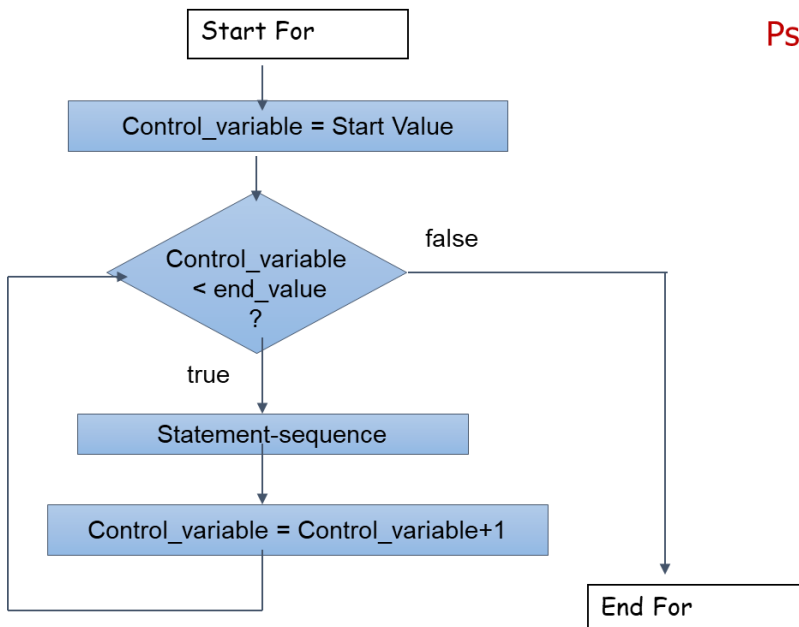
Find the sum of 10 numbers

```

count = 1
sum = 0
WHILE count < 10 Do
    INPUT num
    sum = sum + num
    count = count + 1
END WHILE
DISPLAY sum
    
```



FOR



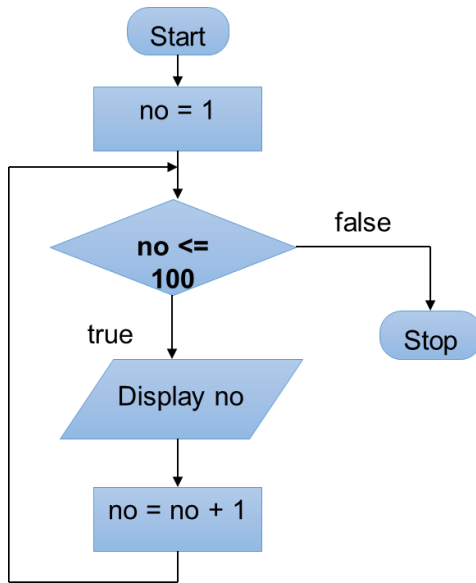
Pseudocode

FOR iteration bounds

sequence

NEXT COUNT

Display the numbers 1, 2, 3, 4, 5,, 100

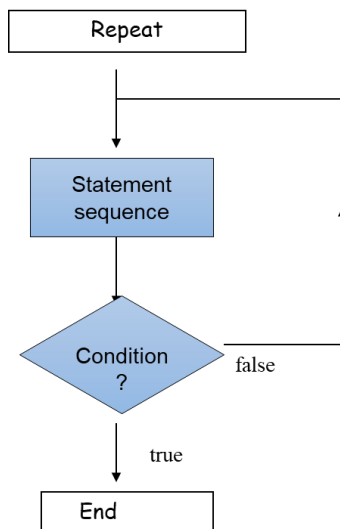


```

BEGIN
    FOR no=1 to 100
        DISPLAY no
        no=no+1
    END

```

REPEAT ... UNTIL...



Pseudocode

REPEAT

Sequence

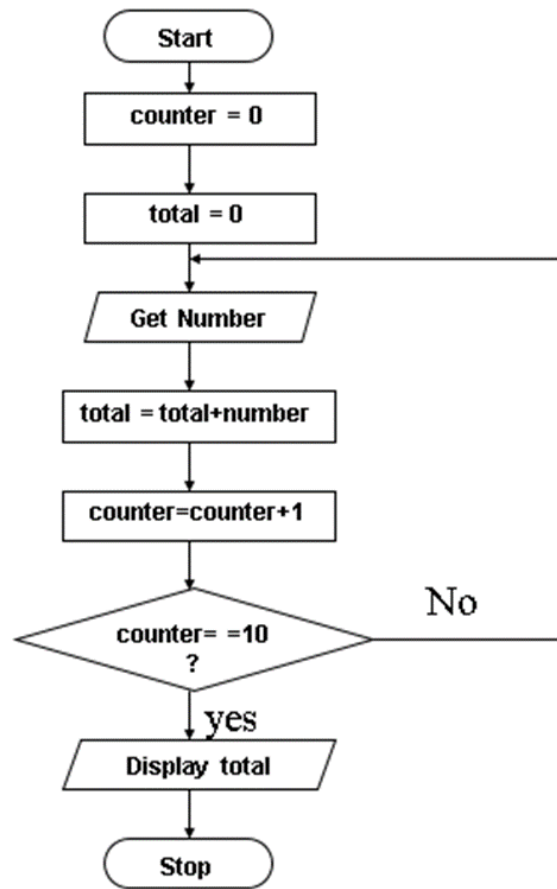
UNTILL condition

Find the sum of 10 numbers

```

BEGIN
  counter = 0
  total = 0
  REPEAT
    INPUT number
    total = total + number
    counter = counter + 1
  UNTILL counter = 10
  DISPLAY total
END

```



Hand Traces

- Also known as **Dry Run /Trace Table/ Desk Check...**
- You have just completed a piece of code but when you run it, it does not behave as expected.
- One way to check and troubleshoot your code is to perform a dry run using a **trace table**.
- Trace tables are used by programmers to track the values of variables as they change throughout the program. This is useful when a program is not producing the desired result.
- The purpose of a Dry run is to see what the code actually does, not what it is supposed to do.

```
turns = 0
```

```
X = 3
```

```
while turns < 22
```

```
    X = X * 3
```

```
    turns = turns + 3
```

```
endwhile
```

```
print (X)
```

```
print (turns)
```

Turns	X	Output
0	—	—
0	3	✓
0	9	✓
3	9	✓
3	27	✓
6	27	✓
6	81	✓

9 81 200+...
24

Algorithm

1

number = 3

2

PRINT number

3

FOR i from 1 to 3:

4

number = number + 5

5

PRINT number

6

PRINT " ? "

Trace Table

Line	number	i	OUTPUT
1	3		
2			3
3		1	
4	8		
5			8
3		2	
4	13		
5			13
3		3	

Exercise

Draw flowcharts and write pseudocodes to solve following problems.

1. Inputs 5 numbers and outputs the sum and average of them.
2. Input the length and width of a quadrilateral and state whether it is a square
3. Input the length and width of a quadrilateral and state whether it is a square or a rectangle.
4. Enter marks of 4 subjects and find the average. If the average is less than 50 then display “fail” else display “pass”.
5. Add two numbers entered by user.
6. Find the largest among three different numbers entered by user.
7. Check whether a number entered by user is prime or not.
8. Find the factorial of a number entered by user.

References –

- Grade 11 O/L Text Book
- A/L ICT Teachers Guide
- http://learnline.cdu.edu.au/units/applicationconcepts/programming/powerpoint/methodsreview_files/textonly/index.html
- https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving_Programming_Data_Representation_and_Practical_Exercise/Problem_Solving/Top-down_design_and_Step-wise_refinement