

PMDS508L - Python Programming

Module 1: Algorithmic Problem Solving

Dr. B.S.R.V. Prasad
Department of Mathematics
School of Advanced Sciences
Vellore Institute of Technology
Vellore

Python



Data Science Techniques



srvprasad.bh@gmail.com (Personal)



srvprasad.bh@vit.ac.in (Official)



+91-8220417476

Those who can imagine anything,
can create the impossible.

Alan Turing

- ▶ **Computational thinking** is an effective learning method that is used to solve complicated problems in a smart way.
- ▶ Those problems could be related to computer science or to everyday life.
- ▶ This method can be applied by humans, smart machines, or both.
- ▶ **Computational thinking** refers to the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer.

Computational Thinking Definition



2

- ▶ **Computational Thinking (CT)** is a problem-solving process.
- ▶ In our daily life we implement Computational Thinking knowingly or unknowingly. Such as
 - ▶ How we cook if we are hungry?
 - ▶ How do you plan your finances?
 - ▶ How do we improve ourselves at gym?
- ▶ In all the above activities we follow a step-by-step approach based on the priority.
- ▶ This step-by-step process involves computational thinking.

Computational Thinking Characteristics



- ▶ Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- ▶ Logically organizing and analyzing data.
- ▶ Representing data through abstractions such as models and simulations.
- ▶ Automating solutions through algorithmic thinking (a series of ordered steps).
- ▶ Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources.
- ▶ Generalizing and transferring this problem solving process to a wide variety of problems.

Computational Thinking Essential Dimensions



The Computational Thinking skills enhance a number of dispositions or attitudes.

- ▶ Confidence in dealing with complexity.
- ▶ Persistence in working with difficult problems.
- ▶ Tolerance for ambiguity.
- ▶ The ability to deal with open ended problems.
- ▶ The ability to communicate and work with others to achieve a common goal or solution.

A Puzzle



Tree



Fish



Bear



Bird



Flower



Dolphin



Tiger



Rooster

Computational Thinking - Illustration

Guess Which Organism I am Thinking Of?

Question 1: Does the organism has legs?

Question 1: Does the organism has legs?

YES

Question 1: Does the organism has legs?

YES



Bear



Bird



Tiger



Rooster

Question 1: Does the organism has legs?

YES

NO



Bear



Bird



Tiger



Rooster

Question 1: Does the organism has legs?

YES



Bear



Bird



Tiger



Rooster

NO



Tree



Fish



Flower



Dolphin

Question 2: Does the organism has wings/fins?

Organisms with Legs



Bear



Bird



Tiger



Rooster

Question 2: Does the organism has wings/fins?

Organisms with Legs

NO



Bear



Bird



Tiger



Rooster

Question 2: Does the organism has wings/fins?

Organisms with Legs



Bear



Bird



Bear



Tiger



Rooster



Tiger

NO

Question 2: Does the organism has wings/fins?

Organisms with Legs

NO

YES



Bear



Bird



Bear



Tiger



Rooster



Tiger

Question 2: Does the organism has wings/fins?

Organisms with Legs



Bear



Bird



Bear



Bird



Tiger



Rooster



Tiger



Rooster

NO

YES

Question 3: Does the organism has stripes?

Organisms with Legs



Bear



Bird



Tiger



Rooster

NO Wings



Bear



Tiger

Question 3: Does the organism has stripes?

YES

Organisms with Legs

NO Wings



Bear



Bird



Bear



Tiger



Rooster



Tiger

Question 3: Does the organism has stripes?

Organisms with Legs



Bear



Bird



Tiger



Rooster

NO Wings



Bear



Tiger

YES



Tiger

Question 3: Does the organism has stripes?

Organisms with Legs



Bear



Bird

NO Wings



Bear



Tiger



Rooster



Tiger

YES



Tiger

NO

Question 3: Does the organism has stripes?

Organisms with Legs



Bear



Bird

NO Wings



Bear

YES



Tiger



Tiger



Rooster



Tiger

NO



Bear

Question 1: Does the organism has legs?

YES



Bear



Bird



Tiger



Rooster

NO



Tree



Fish



Flower



Dolphin

Question 2: Does the organism has wings/fins?

Organisms with no Legs



Tree



Fish



Flower



Dolphin

Question 2: Does the organism has wings/fins?

Organisms with no Legs

NO



Tree



Fish



Flower



Dolphin

Question 2: Does the organism has wings/fins?

Organisms with no Legs

NO



Tree



Fish



Tree



Flower



Dolphin



Flower

Question 2: Does the organism has wings/fins?

Organisms with no Legs

NO

YES



Tree



Fish



Tree



Flower



Dolphin



Flower

Question 2: Does the organism has wings/fins?

Organisms with no Legs



Tree



Fish



Flower



Dolphin

NO



Tree



Flower

YES



Fish



Dolphin

Question 3: Does the organism has stripes?

Organisms with no Legs



Tree



Fish



Flower



Dolphin

With Fins



Fish



Dolphin

Question 3: Does the organism has stripes?

YES

Organisms with no Legs



Tree



Fish



Flower



Dolphin

With Fins



Fish



Dolphin

Question 3: Does the organism has stripes?

Organisms with no Legs



Tree



Fish



Flower



Dolphin

With Fins



Fish



Dolphin

YES



Fish

Question 3: Does the organism has stripes?

Organisms with no Legs



Tree



Fish



Flower



Dolphin

With Fins



Fish



Dolphin

YES



Fish

NO

Question 3: Does the organism has stripes?

Organisms with no Legs



Tree



Fish



Flower



Dolphin

With Fins



Fish



Dolphin

YES



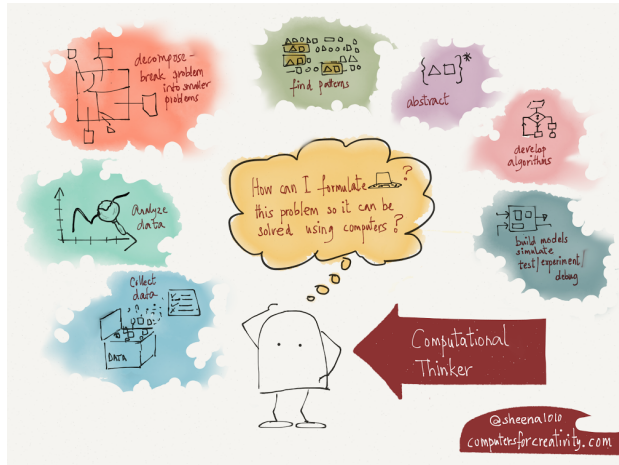
Fish

NO



Dolphin

Computational Thinking Components



Computational Thinking Components



Computational Thinking is usually broken down into the following four components:

- ▶ **Decomposition**
- ▶ **Pattern Recognition**
- ▶ **Abstraction**
- ▶ **Algorithm Design**

Computational Thinking Components



- ▶ **Decomposition:** Breaking down data, processes, or problems into smaller, manageable parts.

Example in Literature: Break down the analysis of a poem into analysis of meter, rhyme, imagery, structure, tone, diction and meaning.

- ▶ **Pattern Recognition:** Observing patterns, trends, similarities, and differences across data.

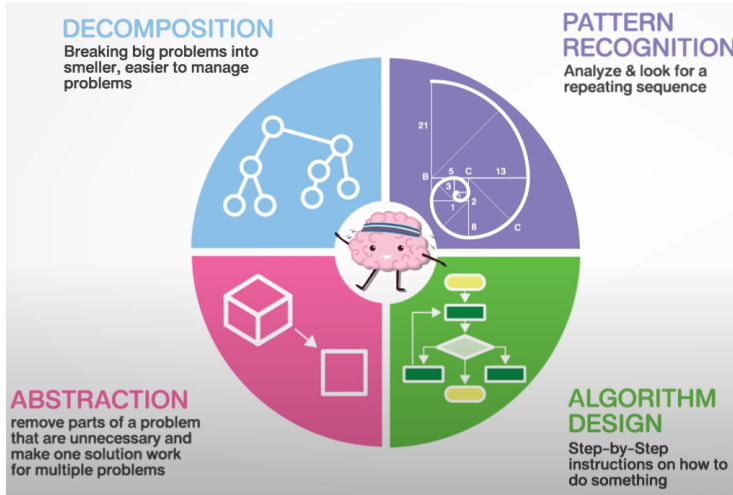
Example in Economics: Find cycle patterns in the rise and drop of the country's economy.

Computational Thinking Components



- ▶ **Abstraction:** Removing unnecessary information and identifying the general principles that generate these patterns.
Example in Mathematics: Figure out the rules for factoring 2nd-order polynomials.
Example in Chemistry: Determine the rules for chemical bonding and interactions.
- ▶ **Algorithm Design:** Developing the step-by-step instructions for solving this and similar problems.
Culinary Arts: Write a recipe for others to use.

Computational Thinking Components



Computational Thinking vs Computer Science



Computational Thinking and Computer Science are different.

- ▶ Computational thinking is a problem-solving process that can be integrated across subject areas.
- ▶ Computer science is an academic discipline that involves the study of computation and its application using computers, and uses computational thinking as its primary problem-solving process.

Problem Solving: A Systematic Approach



- ▶ Problems are inherent in our day-to-day life.
- ▶ When a problem arises, we are ready to start analysing it and coming with a solution.
- ▶ As we are talking about computerising the solution, we need not only any solution but, one solution that is specially formed so that a computer could carry it out.
- ▶ So a question arises to all of us – “Where on earth do we start?”.

Problem Solving: A Systematic Approach



- ▶ Real-world problems tend to be big, complex things.
- ▶ Examining any non-trivial problem reveals all manner of hidden details, complex nuances and various facets to consider.
- ▶ It is impossible to understand the computational thinking without understanding the problem-solving skills and techniques.
- ▶ A step-by-step procedure for problem-solving would be an obvious benefit.
- ▶ Unfortunately, problem-solving is partly a *creative* process.
- ▶ Problem solving cannot be totally systematised, but strategies, heuristics and good practices exist to help us during the our creative endeavours.

Problem Solving: A Systematic Approach



- ▶ Problem solving requires a systematic approach.
- ▶ An early example of systematic approach to general problem-solving was introduced by George Pölya, a Hungarian mathematician in a book called "*How to Solve It*" first published in 1973.
- ▶ Still, this book is in print after more than half-a-century and still abounds with relevant and helpful tips.
- ▶ The techniques described by Pölya guided many problem solvers down the years.
- ▶ Techniques to problem-solving inspired by the best traditions of mathematical and natural sciences.

Problem Solving: A Systematic Approach

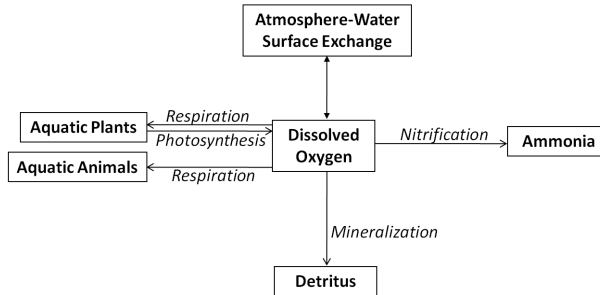


- ▶ There are four steps in problem-solving process as per Pölya
 1. **Understand the problem:** This is also known as problem representation. Pölya encourages solvers to draw diagrams and graphs if necessary.
 2. **Devise a plan:** In this step the solvers find a connection between the data and the unknown by trying to find similar problems that they have encountered before or finding auxiliary problems if a direct connection cannot be found. Ultimate goal of this step is to come up with a plan to the solution.
 3. **Carry out the plan:** In this step solvers implement their solution also check that each step is correct.
 4. **Evaluate the result and Look backward:** Finally, solvers examine their solution for correctness and understanding. If any thing is not according to the plan the go back and redo the process.

Problem Solving: A Systematic Approach



Example: Suppose we wish to model the Dissolved Oxygen in a Lake. Then we should have proper understanding of various processes that influence the Oxygen levels in water viz., Oxygen exchange between air and water, Oxygen generation and consumption by Phytoplankton, Oxygen consumption by Fish etc., Nitrification process and finally mineralisation process of dead animals. This can be represented as a pictorial way as below:



Problem Solving: Present Day Approach



- ▶ The present day computer scientists are employing the below four strategies that are the core of this modern style of problem solving:
 1. Problem definition
 2. Logical reasoning
 3. Decomposition
 4. Abstraction

Problem Solving using Boolean Logic



In the next few slides we delve into solving some logical problems using the Boolean logic techniques.

Tom was telling you what he ate yesterday afternoon. He tells you,

"I had either popcorn or raisins. Also, if I had cucumber sandwiches, then I had soda. But I didn't drink soda or tea."

Of course you know that Tom is the worlds worst liar, and everything he says is false. What did Tommy eat?

Hint: Justify your answer by writing all of Tom's statements using sentence variables (P for Popcorn; C for Cucumber; R for Raisins; S for Soda; and T for Tea;), taking their negations, and using these to deduce what Tommy actually ate.

Boolean Logic

What did Tommy eat?



Let P = Tom had popcorn,
 C = Tom had cucumber sandwich,
 R = Tom had Raisins,
 S = Tom drink Soda,
 T = Tom drink Tea.

Boolean Logic

What did Tommy eat?



Let P = Tom had popcorn,

C = Tom had cucumber sandwich,

R = Tom had Raisins,

S = Tom drink Soda,

T = Tom drink Tea.

We now have the following symbolic form of the statements made by Tommy:

$S1 : P \vee R,$

$S2 : C \rightarrow S,$

$S3 : \neg(S \vee T).$

Let P = Tom had popcorn,

C = Tom had cucumber sandwich,

R = Tom had Raisins,

S = Tom drink Soda,

T = Tom drink Tea.

We now have the following symbolic form of the statements made by Tommy:

$S1 : P \vee R$,

$S2 : C \rightarrow S$,

$S3 : \neg(S \vee T)$.

Since we know that Tom is a liar, we will take negations of above statements and we obtain

$\neg S1 : \neg(P \vee R)$, is true if $P \vee R$ is false. So, both P and R are false.

$\neg S2 : \neg(C \rightarrow S)$, is true if $C \rightarrow S$ is false. So, C is true and S is false.

$\neg S3 : \neg(\neg(S \vee T))$, is true if $S \vee T$ is true. Since S is false, we must have T is true.

Let P = Tom had popcorn,

C = Tom had cucumber sandwich,

R = Tom had Raisins,

S = Tom drink Soda,

T = Tom drink Tea.

We now have the following symbolic form of the statements made by Tommy:

$S1 : P \vee R$,

$S2 : C \rightarrow S$,

$S3 : \neg(S \vee T)$.

Since we know that Tom is a liar, we will take negations of above statements and we obtain

$\neg S1 : \neg(P \vee R)$, is true if $P \vee R$ is false. So, both P and R are false.

$\neg S2 : \neg(C \rightarrow S)$, is true if $C \rightarrow S$ is false. So, C is true and S is false.

$\neg S3 : \neg(\neg(S \vee T))$, is true if $S \vee T$ is true. Since S is false, we must have T is true.

From the above statements we conclude that **“Tom had Cucumber Sandwich and drink Tea”**.



Holmes owns two suits: one black and one tweed. He always wears either a tweed suit or sandals. Whenever he wears his tweed suit and a purple shirt, he chooses to not wear a tie. He never wears the tweed suit unless he is also wearing either a purple shirt or sandals. Whenever he wears sandals, he also wears a purple shirt. Yesterday, Holmes wore a bow tie. What else did he wear?

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

True statements:

$$W \vee S$$

$$(W \wedge P) \rightarrow \neg T$$

$$W \rightarrow (P \vee S)$$

$$S \rightarrow P$$

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T				
T	T	F				
T	F	T				
T	F	F				
F	T	T				
F	T	F				
F	F	T				
F	F	F				

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T			
T	T	F	T			
T	F	T	T			
T	F	F	T			
F	T	T	T			
F	T	F	T			
F	F	T	F			
F	F	F	F			

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

Since $\neg T$ is false, for this statement to be true, we need $W \wedge P$ to be false



True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T			
T	T	F	T			
T	F	T	T			
T	F	F	T			
F	T	T	T			
F	T	F	T			
F	F	T	F			
F	F	F	F			

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

Since $\neg T$ is false, for this statement to be true, we need $W \wedge P$ to be false



True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T	F		
T	T	F	T	T		
T	F	T	T	F		
T	F	F	T	T		
F	T	T	T	T		
F	T	F	T	T		
F	F	T	F	T		
F	F	F	F	T		

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

Since $\neg T$ is false, for this statement to be true, we need $W \wedge P$ to be false



True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T	F	T	
T	T	F	T	T	T	
T	F	T	T	F	T	
T	F	F	T	T	F	
F	T	T	T	T	T	
F	T	F	T	T	T	
F	F	T	F	T	T	
F	F	F	F	T	T	

Boolean Logic

Deduction Problem - What did Sherlock Holmes wear?



- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

Since $\neg T$ is false, for this statement to be true, we need $W \wedge P$ to be false



True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T	F	T	T
T	T	F	T	T	T	F
T	F	T	T	F	T	T
T	F	F	T	T	F	T
F	T	T	T	T	T	T
F	T	F	T	T	T	F
F	F	T	F	T	T	T
F	F	F	F	T	T	T

- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

Since $\neg T$ is false, for this statement to be true, we need $W \wedge P$ to be false



True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T	F	T	T
T	T	F	T	T	T	F
T	F	T	T	F	T	T
T	F	F	T	T	F	T
F	T	T	T	T	T	T
F	T	F	T	T	T	F
F	F	T	F	T	T	T
F	F	F	F	T	T	T

- Then if we cross out all rows where one of the final four columns contains an F, we are only left with one row, the one in bold!

- W = Holmes wears a tweed suit
- P = Holmes wears a purple shirt
- S = Holmes wears sandals
- T = Holmes wears a tie

**Conclusion: Holmes wears a Black Suit,
Bow Tie, Purple Shirt, Sandals**



True statements:

$W \vee S$

$(W \wedge P) \rightarrow \neg T$

$W \rightarrow (P \vee S)$

$S \rightarrow P$

W	S	P	$W \vee S$	$(W \wedge P) \rightarrow \neg T$	$W \rightarrow (P \vee S)$	$S \rightarrow P$
T	T	T	T	F	T	T
T	T	F	T	T	T	F
T	F	T	T	F	T	T
T	F	F	T	T	F	T
F	T	T	T	T	T	T
F	T	F	T	T	T	F
F	F	T	F	T	T	T
F	F	F	F	T	T	T

- Then if we cross out all rows where one of the final four columns contains an F, we are only left with one row, the one in bold!

Boolean Logic

Engineering Elevator Doors



As a corporate client, I wish to hire a software engineer to write software to control an elevator system for a small four-floor office building. The elevator has two doors that are known as the front and the rear door. The front doors are usable only on the first two floors, and the rear doors are usable only on the top two floors.

Boolean Logic

Engineering Elevator Doors



41

An interview session between the software engineer (SE), and me, might go something like the following dialogue:

SE: What should happen when the “open door” button is pressed?

ME: The door should open.

SE: The front door or the rear door or both?

ME: The front door should open if the elevator is on either floor 1 or 2, otherwise the rear door should open.

SE: What if the “open door” and “close door” buttons are pressed at the same time?

ME: Hmmm. I hadn’t thought about that. I guess nothing should happen.

SE: What if the elevator is moving between two floors and the “open door” button is pressed?

ME: Nothing should happen.

SE: What if emergency personnel have inserted a key to manually override the software control system. In that case, what should happen when the “open door” button is pressed?

ME: Hmmm. That’s a good question. I guess that both the front and rear doors should open regardless of which floor the elevator is on or even if the elevator is moving between floors.

Formulate a logical sequence of steps to carry out the elevator control system software following the above conversation.

Boolean Logic

Engineering Elevator Doors



The “open door” function for the elevator that has to be specified by Software Engineer has four steps.

- ▶ (Floor=1 OR Floor=2) AND (Not Moving) AND Button_Pushed IMPLIES Front_Door_Opens
- ▶ (Floor=3 OR Floor=4) AND (Not Moving) AND Button_Pushed IMPLIES Rear_Door_Opens
- ▶ (Emergency_Key_Inserted AND Button_Pushed) IMPLIES (Front_Door_Opens AND Rear_Door_Opens)
- ▶ In all OTHER CASES, the System Must Do NOTHING

Algorithmic Thinking

Definition



- ▶ An **Algorithm** is sequence of discrete actions that when followed, will result in achieving some goal or solving some problem.
- ▶ In other words, an **Algorithm** is a sequence of clearly defined steps that describe a process to follow a finite set of unambiguous instructions with clear start and end points.

Algorithmic Thinking

Definition



- ▶ An **Algorithm** is sequence of discrete actions that when followed, will result in achieving some goal or solving some problem.
- ▶ In other words, an **Algorithm** is a sequence of clearly defined steps that describe a process to follow a finite set of unambiguous instructions with clear start and end points.
- ▶ Logic and algorithms are not same.
- ▶ Algorithms build on logic because, as part of their work, algorithms make logical decisions and stich those decisions together.

Algorithmic Thinking Examples



- ▶ Everyone is accustomed to following algorithms in daily life whether we are aware of it or not.
- ▶ For example:
 - ▶ Players are following an algorithms when they execute a play on the filed.
 - ▶ Drivers are using an algorithm when they follow a set of instructions for getting from one city to another.
 - ▶ Musicians follow a set of rhythmic instructions (algorithm) to produce music which is enjoyable by everyone.
 - ▶ Mathematicians use a step-by-step well defined and well executed steps to solve any mathematical problem such as finding the ratio of two numbers etc.

Algorithmic Thinking

Cookie Recipe

Chocolate Chip¹ Cookie Recipe

Ingredients

- 1 cup melted butter
- 2 cups brown sugar
- 2 eggs
- 3 cups flour
- 1 teaspoon baking powder
- 1 teaspoon baking soda
- 2 cups chocolate chips



Directions

1. Preheat the oven to 375 degrees F.
2. Line a cookie sheet with parchment paper
3. In a bowl, stir together the butter, brown sugar and eggs.
4. In a separate bowl, combine the flour, baking powder and baking soda. Gradually combine with the sugar mixture.
5. Add the chocolate chips
6. Fill the cookie sheet with one-spoonful drops of the cookie dough.
7. Bake dough for 9 minutes
8. Cool for five minutes before removing from cookie sheet.

Algorithmic Thinking

Properties of Algorithms



- ▶ One of the most important requirement is that each of the individual actions referred to in the algorithm be meaningful.
 - ▶ i.e., programmers must know the precise meaning of each action they write.
- ▶ Computer programmers refer to the meaning of action as the *semantics* of an action.
- ▶ The phrase *semantics* refers to the meaning of the actions that occur in an algorithm.

Algorithmic Thinking

Properties of Algorithms



- ▶ Another requirement is that actions of an algorithm must have only one possible interpretation.
 - ▶ i.e., there is no misunderstanding about the semantics of the action.
- ▶ We say that an action must be *unambiguous*; meaning that the action is not subject to conflicting interpretations.

Algorithmic Thinking

Properties of Algorithms



- ▶ An algorithm must also ensure that the order in which the actions occur be well defined.
- ▶ Finally, requirement is that the number of actions that are described in algorithm must be finite rather than infinite.
 - ▶ No goal can be reached if we are required to follow a never-ending sequence of actions.
 - ▶ In computer terminology this requirement can be expressed as every algorithm must halt in order to be useful.

- ▶ Summarising the above we have the following:
 - ▶ Algorithm is a collection of individual steps
 - ▶ Each step must be precisely defined and has one and only one meaning i.e., unambiguous.
 - ▶ Algorithms are sequential i.e., the steps must be carried out in the order specified.

Components of Algorithm



An algorithm consists of

- ▶ Statements
- ▶ State
- ▶ Control Flow/Repetition
- ▶ Functions/Modularisation

Components of Algorithm Statements



- ▶ In programming, a **statement** is a single line of code that performs a specific action.
- ▶ It can be an assignment, a function call, a loop, or a conditional operation.
 - ▶ For example, an assignment statement like `x = 10` assigns the value 10 to the variable x.

Components of Algorithm State



- ▶ **State** refers to the condition or status of a program or system at a particular moment.
- ▶ It represents the values stored in variables, memory, and other data structures.
 - ▶ For instance, if you have a counter variable count, its state could be the current value of count.

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. *Celsius* \leftarrow 33.5

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. *Celsius* \leftarrow 33.5 The state is now [Celsius = 33.5]

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. *Celsius* \leftarrow 33.5 The state is now [*Celsius* = 33.5]
2. *Fahrenheit* \leftarrow *Celsius* * 9

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. *Celsius* \leftarrow 33.5 The state is now [*Celsius* = 33.5]
2. *Fahrenheit* \leftarrow *Celsius* * 9 The state is now [*Celsius* = 33.5 and *Fahrenheit* = 301.5]

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. *Celsius* \leftarrow 33.5 The state is now [*Celsius* = 33.5]
2. *Fahrenheit* \leftarrow *Celsius* * 9 The state is now [*Celsius* = 33.5 and *Fahrenheit* = 301.5]
3. *Fahrenheit* \leftarrow *Fahrenheit*/5

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. $\text{Celsius} \leftarrow 33.5$ The state is now [Celsius = 33.5]
2. $\text{Fahrenheit} \leftarrow \text{Celsius} * 9$ The state is now [Celsius = 33.5 and Fahrenheit = 301.5]
3. $\text{Fahrenheit} \leftarrow \text{Fahrenheit}/5$ The state is now [Celsius = 33.5 and Fahrenheit = 60.3]

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. *Celsius* \leftarrow 33.5 The state is now [*Celsius* = 33.5]
2. *Fahrenheit* \leftarrow *Celsius* * 9 The state is now [*Celsius* = 33.5 and *Fahrenheit* = 301.5]
3. *Fahrenheit* \leftarrow *Fahrenheit*/5 The state is now [*Celsius* = 33.5 and *Fahrenheit* = 60.3]
4. *Fahrenheit* \leftarrow *Fahrenheit* + 32

Components of Algorithm State



Below we present an algorithm for converting temperature from Celsius to Fahrenheit.

$$\text{temperature in Fahrenheit} = \text{temperature in Celsius} \times \frac{9}{5} + 32$$

Convert from Celsius to Fahrenheit Algorithm

1. $\text{Celsius} \leftarrow 33.5$ The state is now [Celsius = 33.5]
2. $\text{Fahrenheit} \leftarrow \text{Celsius} * 9$ The state is now [Celsius = 33.5 and Fahrenheit = 301.5]
3. $\text{Fahrenheit} \leftarrow \text{Fahrenheit}/5$ The state is now [Celsius = 33.5 and Fahrenheit = 60.3]
4. $\text{Fahrenheit} \leftarrow \text{Fahrenheit} + 32$
The state is now [Celsius = 33.5 and Fahrenheit = 92.3]

Components of Algorithm

Control Flow



- ▶ **Control flow** is a computational term that refers to the specific order in which individual actions of a computer program are executed.
- ▶ Normally actions are performed sequentially in the order that they have been written.
- ▶ Control flow statements can change this ordering and control the specific ordering of the actions performed by a program.
- ▶ There are two types of control flows:
 - ▶ Selection statement
 - ▶ Repetition

Components of Algorithm

Control Flow - Selection



- ▶ A *selection statement* is a control flow that allows a program to make choices regarding whether certain actions should be performed.
- ▶ Selection statement again classified into three types:
 - ▶ *One-way selection* - allows a programmer to either perform an action or skip the action
 - ▶ *Two-way selection* - allows the computer to choose one of exactly two actions
 - ▶ *Multiway selection* - allows the computer to choose one of several alternatives

Components of Algorithm

Control Flow - Repetition



- ▶ We often required to repeat a sequence of actions in order to achieve some greater outcome.
- ▶ A *loop* is a control structure that repeatedly executes a sequence of actions.
- ▶ A *for loop* repeats a block of code a specified number of times.
- ▶ A *while loop* repeats a block of code as long as a condition is true.
- ▶ A *do-while loop* executes a block of code once and then repeats it while a condition holds.

Components of Algorithm

Control Flow - Repetition



All well-written loops must have three well-defined elements.

1. **Initialisation** — Every variable that occurs in the loop must hold the correct value prior to entering the loop.
2. **Condition** — The condition that determines when to repeat the loop must be precise.
3. **Progress** — The actions that are repeatedly executed must in some way make progress that allows the loop to terminate.

Components of Algorithm Functions



- ▶ **Functions** are reusable blocks of code that perform specific tasks.
- ▶ They encapsulate logic, accept input (parameters), and produce output (return value).
 - ▶ For example, the factorial occurrence in $\binom{n}{r}$.

- ▶ Algorithms are usually presented in the form called pseudo-code.
- ▶ A pseudo-code is an informal high-level textual description of the operating principle of a computer program or the algorithm.
- ▶ It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading.
- ▶ Good pseudo-code abstracts the algorithm, makes good use of mathematical notation and is easy to read.
- ▶ Bad pseudo-code gives too many details or is too implementation specific.

Algorithm: Maximum in a set of numbers

Data: A set $A = \{a_1, a_2, \dots, a_n\}$ of integers

Result: An index i such that $a_i = \max\{a_1, a_2, \dots, a_n\}$

begin

$index \leftarrow 1;$

for $i \leftarrow 2$ **to** n **do**

if $a_i > a_{index}$ **then**

$index \leftarrow i;$

end

end

output: $index, a_{index};$

end

Algorithm: Maximum in a set of numbers

Data: A set $A = \{a_1, a_2, \dots, a_n\}$ of integers

Result: An index i such that $a_i = \max\{a_1, a_2, \dots, a_n\}$

begin

$index \leftarrow 1;$

$i \leftarrow 1;$

while $i \leq n$ **do**

if $a_i > a_{index}$ **then**

$index \leftarrow i;$

end







$i \leftarrow i + 1;$

end

output: $index, a_{index};$

end

- ▶ A graphical tool that *diagrammatically* depicts the steps and structure of an algorithm or program
- ▶ The most commonly used symbols of flow chart are listed below

Symbol	Name/Meaning	Symbol	Meaning
	<u>Process</u> – Any type of internal operation: data transformation, data movement, logic operation, etc.		<u>Connector</u> – connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow
	<u>Input/Output</u> – input or output of data		<u>Terminal</u> – indicates start or end of the program or algorithm
	<u>Decision</u> – evaluates a condition or statement and branches depending on whether the evaluation is true or false		<u>Flow lines</u> – arrows that indicate the direction of the progression of the program

General Rules for Flowcharts



- ▶ All symbols of the flowcharts are connected by flow lines (note *arrows*, not by lines).
- ▶ Flow lines enter the top of the symbol and exit out the bottom, except for the Decision symbol, which can have flow lines existing from the bottom or the sides.
- ▶ Flowcharts are drawn so flow generally goes from top to bottom.
- ▶ The beginning and the end of the flowcharts is indicated using Terminal symbol.

Algorithmic Thinking

Control Flow - One-way Selection



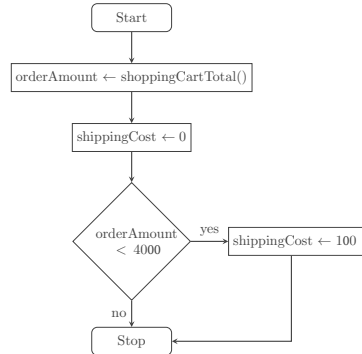
- ▶ As said earlier *One-way selection* statement allows a programmer to either perform an action or skip the action.
- ▶ As an example consider the case where an online store offers free shipping if the shopping cart total exceeds Rs. 4000 but charges a shipping charge of Rs. 100 if the cart total is below Rs. 4000.

Algorithmic Thinking

Control Flow - One-way Selection



- ▶ As said earlier *One-way selection* statement allows a programmer to either perform an action or skip the action.
- ▶ As an example consider the case where an online store offers free shipping if the shopping cart total exceeds Rs. 4000 but charges a shipping charge of Rs. 100 if the cart total is below Rs. 4000.
- ▶ At first we describe the algorithm as a flowchart as follows:



Algorithmic Thinking

Control Flow - One-way Selection



- ▶ As said earlier *One-way selection* statement allows a programmer to either perform an action or skip the action.
- ▶ As an example consider the case where an online store offers free shipping if the shopping cart total exceeds Rs. 4000 but charges a shipping charge of Rs. 100 if the cart total is below Rs. 4000.
- ▶ We now describe the algorithm using a pseudo code

One-way Selection

```
if CONDITION then  
    ACTIONS  
endif
```

Algorithmic Thinking

Control Flow - One-way Selection



- ▶ As said earlier *One-way selection* statement allows a programmer to either perform an action or skip the action.
- ▶ As an example consider the case where an online store offers free shipping if the shopping cart total exceeds Rs. 4000 but charges a shipping charge of Rs. 100 if the cart total is below Rs. 4000.
- ▶ We now describe the algorithm using a pseudo code

One-way Selection

```
if CONDITION then  
    ACTIONS  
endif
```

Algorithm: Algorithm for shipping charges

begin

```
    orderAmount  $\leftarrow$  shoppingCartTotal();  
    shippingCost  $\leftarrow$  0;  
    if orderAmount < 4000 then  
        | shippingCost  $\leftarrow$  100;  
    end  
    output: shippingCost;
```

end

Algorithmic Thinking

Control Flow - Two-way Selection



- ▶ Although the algorithm presented above computes the shipping cost for any order amount, it can be observed that two bindings are required in the algorithm.
- ▶ A two-way selection, the computer must choose exactly one of the two paths to follow.
- ▶ This is natural way of expressing the computation of shipping cost since there are exactly two possible shipping costs and exactly one of them must be chosen.

Algorithmic Thinking

Control Flow - Two-way Selection



Two-way Selection

```
if CONDITION then
    IF-TRUE-ACTIONS
else
    IF-FALSE-ACTIONS
endif
```

Algorithm: Algorithm for shipping charges

```
begin
    orderAmount  $\leftarrow$  shoppingCartTotal();
    if orderAmount < 4000 then
        |   shippingCost  $\leftarrow$  100;
    else
        |   shippingCost  $\leftarrow$  0;
    end
    output: shippingCost;
end
```

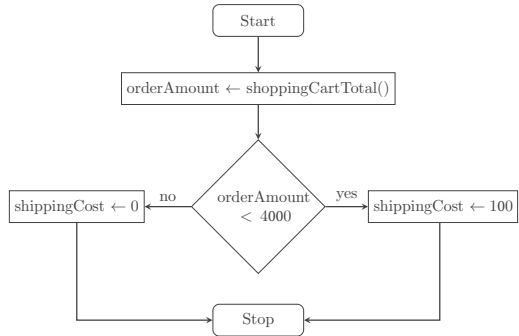
Algorithmic Thinking

Control Flow - Two-way Selection



Two-way Selection

```
if CONDITION then  
    IF-TRUE-ACTIONS  
else  
    IF-FALSE-ACTIONS  
endif
```



Algorithmic Thinking

Control Flow - Multi-way Selection



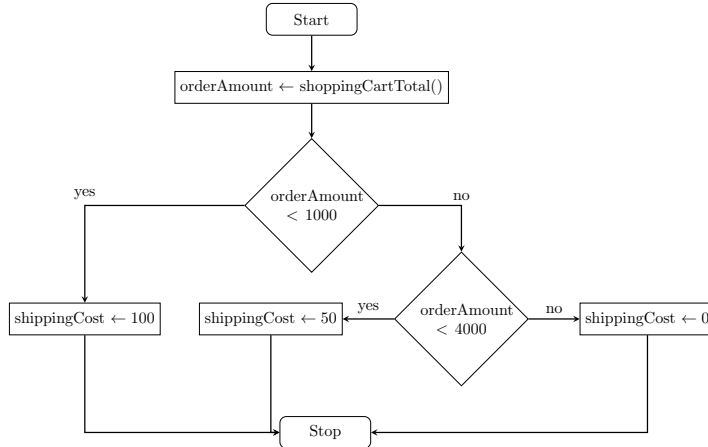
- ▶ We often need more flexibility than selecting one alternative between only two possible actions.
- ▶ For example consider the same problem of shipping cost but now we incorporate three-level shipping policy as shown below:

Shipping Cost Policy	
Order Amount	Shipping Cost
Rs. 0 to Rs. 999.99	Rs. 100
Rs. 1000 to Rs. 3999.99	Rs. 50
Rs. 4000 and above	Rs. 0

- ▶ This can be represented in the following flow chart

Algorithmic Thinking

Control Flow - Multi-way Selection



Algorithmic Thinking

Control Flow - Multi-way Selection



Algorithm: Algorithm for shipping charges

begin

$orderAmount \leftarrow shoppingCartTotal()$;

if $orderAmount < 1000$ **then**

$shippingCost \leftarrow 100$;

else if $orderAmount \geq 1000$ **and** $orderAmount < 4000$ **then**

$shippingCost \leftarrow 50$;

else

$shippingCost \leftarrow 0$;

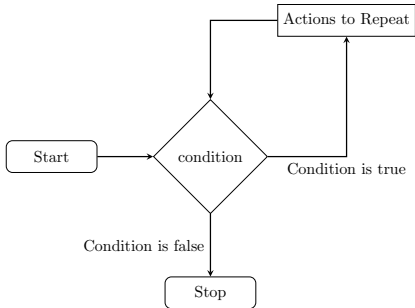
end

output: $shippingCost$;

end

Algorithmic Thinking

Control Flow - Repetition



While Loop

```
while CONDITION do  
    ACTIONS  
endwhile
```

Algorithmic Thinking

Control Flow - Repetition



Algorithm: Maximum in a set of numbers

Data: A set $A = \{a_1, a_2, \dots, a_n\}$ of integers

Result: An index i such that $a_i = \max\{a_1, a_2, \dots, a_n\}$

begin

$index \leftarrow 1;$

$i \leftarrow 1;$

while $i \leq n$ **do**

if $a_i > a_{index}$ **then**

$index \leftarrow i;$

end

$i \leftarrow i + 1;$

end

output: $index, a_{index};$

end

Algorithmic Thinking

Modularisation/Function



- ▶ *Modularisation/Function* is a vital element of programming that allows us to define new computable actions by assigning a name to some computable process.
- ▶ Algorithms can be modularised by breaking them into independent subprocess.
- ▶ A *module/function* is a named subprocess.

```
module CONDITION is  
    ACTIONS  
endmodule
```

Algorithmic Thinking

Modularisation/Function



Algorithm: Minimum in a Set Module

```
module setMin()  
    index  $\leftarrow$  1;  
    i  $\leftarrow$  1;  
    while  $i \leq n$  do  
        if  $a_i < a_{index}$  then  
            index  $\leftarrow$  i;  
        end  
        i  $\leftarrow$  i + 1;  
    end  
end
```

Algorithmic Thinking

Modularisation/Function



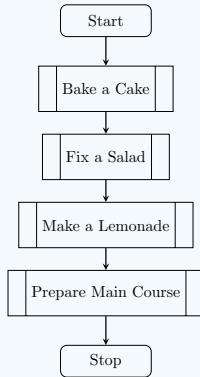
- ▶ Modules are vital when writing large programs that consist of many individual parts.
- ▶ For example consider a chef who is preparing dinner for a small private party.
- ▶ The entire process of preparing dinner can be thought of series of modules that involves
 - ▶ baking a cake
 - ▶ fixing a salad
 - ▶ making lemonade
 - ▶ preparing main course
- ▶ Each of these modules can be understood as an individual, programmer-defined computable action, and these actions can be invoked by simply referring to them by a name.

Algorithmic Thinking

Modularisation/Function



Dinner Preparation Module



```
module makeDinner()  
    bakeCake();  
    fixSalad();  
    makeLemonade();  
    prepareMainCourse();  
end
```

Algorithmic Thinking

Modularisation/Function



- ▶ Modularity is an extremely powerful technique for designing complex algorithms.
- ▶ Modules allow us to isolate smaller parts of a large process such that those smaller parts are much easier to understand and manage.
- ▶ These smaller parts are self-contained, they can be easily inserted into other complex algorithms without needing to customize them or rewrite them.
- ▶ Although we must understand precisely what a submodule does, we do not need to fully understand the details of how a submodule actually works.

Algorithmic Thinking

Modularisation/Function



- ▶ Well-designed modules should meet several criteria.
- ▶ These criteria ensures
 - ▶ that modules can be used in a wide variety of larger processes
 - ▶ module is self-contained, so that any errors caused by a module have limited effect on the whole system

Algorithmic Thinking

Modularisation/Function



The criteria that modules should follow are:

1. **Understandability** — Every module is self-contained, which implies that it can be fully understood without any knowledge of actions that take place outside of the module itself.
2. **Encapsulation** — Every module affects only the data that it contains. Any errors that arise from a module are also contained within the module.
3. **Composition** — Every module can be incorporated into a larger modules without special treatment.

Algorithmic Thinking

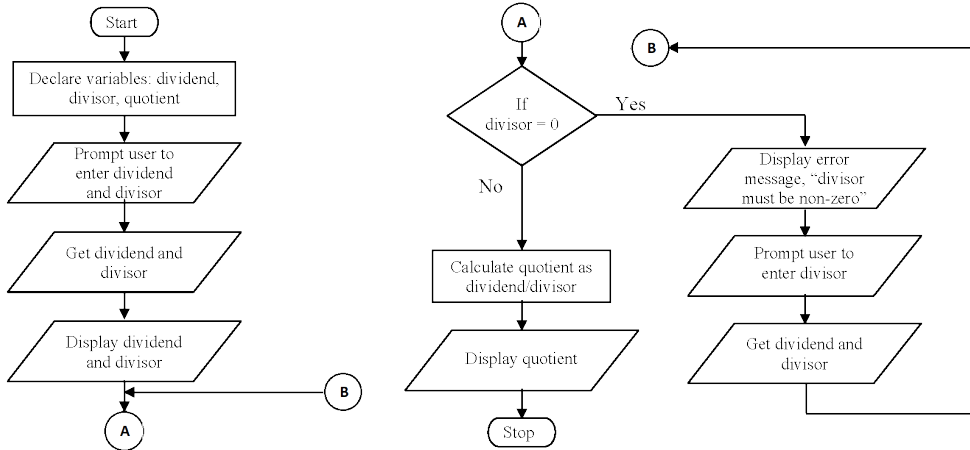
Modularisation/Function



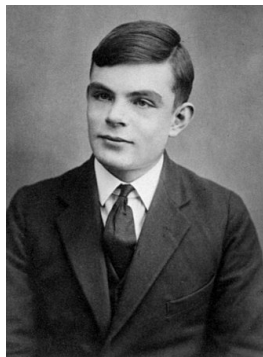
- ▶ Modules should typically be flexible enough to be used in a variety of conditions.
- ▶ Modules are made flexible by allowing users to feed input values into the code.
- ▶ The module can then respond differently depending on the input values provided by the user.
- ▶ The number of inputs that a module accepts must be defined when a module is written.
- ▶ These inputs are known as *formal parameters* — a variable where the initial value is bound to the values provided as input to the module.
- ▶ The initial values provided by the module user are known as *arguments* or *actual parameter*.

Flowchart Example

Finding quotient when one number is divided by another



- ▶ **Alan Mathison Turing** (23 June 1912 – 7 June 1954) was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist.
- ▶ Turing was highly influential in the development of *theoretical computer science*, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer.
- ▶ He is known as the father of modern computing and artificial intelligence.



Turing at the Age 16.

https://en.wikipedia.org/wiki/Alan_Turing

Back