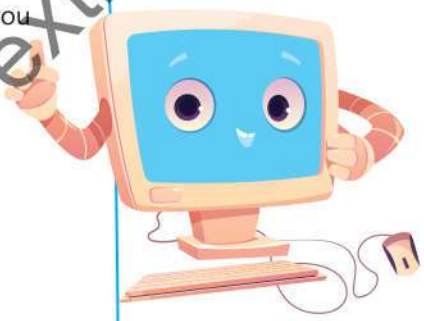
5 Computational Thinking

Knowledge:

Students will be able to:

- Define and infer simple and complex problems, and how to identify each.
- Create algorithms/solutions to simple and complex problems.
- Discuss the scope and limitations, and that some problems cannot be solved computationally (e.g. factoring very large numbers in a small amount of time, or Turing's Halting problem, how a computer can never reliably inspect someone's code and tell you whether it will halt or run forever).
- Discuss basics of writing pseudocode
- Discuss the concept of nesting.
- Discuss the concept of constants and variables.
- Distinguish scenario/problem where If, If then else, and If with multiple conditions can be applied.



Skills:

Students will be able to:

- Create algorithms/solutions to simple and complex problems.
- Apply the best possible solution to a problem from a pool of solutions.
- Describe that there are ways to characterize how well algorithms perform and that two algorithms can perform differently for the same task.
- Explain, with examples, some problems, which cannot be solved computationally.
- Represent algorithms using structured language, such as pseudocode
- Apply the concept of nesting up to level 2 in looping and conditions.
- Apply repeat and forever loops in Algorithm building.
- Identify problems using the IF statement with multiple conditions.
- Identify problem-solving techniques (sequence, loop, and conditions) applicable to a specific problem.

PROBLEM-SOLVING APPROACH

Problem-solving refers to the process of finding solutions to problems that you come across in everyday life. In other words, problem-solving states that you always have to try to find the necessary steps to be taken in a sequence to complete a task.

Let us understand this concept with the help of an example. To prepare a cup of tea or

coffee, there are some equipment and ingredients required, and then you need to follow the process involving the steps given below:

- Pour the ingredients into the utensil.
- Put it on the gas stove.
- Turn on the gas stove.
- Bring the solution to a boil until it is cooked.
- Pour it into a cup.

Similarly, to solve any problem on the computer, a step-by-step approach is followed. For example, if you want to get the sum of two numbers, you need to follow the steps given below.

- Input two numbers.
- Perform the addition of input numbers.
- Display the sum.

This shows that there is always a need for problem-solving processes to complete a task whether it is general or it is related to a computer system.

COMPUTATIONAL THINKING (CT)

Computational thinking is a set of problem-solving processes that includes several characteristics and perspectives. Computational thinking allows us to take a complex problem, understand what the problem is and develop possible solutions.

We can then present these solutions in a way that a computer, a human, or both, can understand. It is important to learn computational thinking also for the development of computer programs.

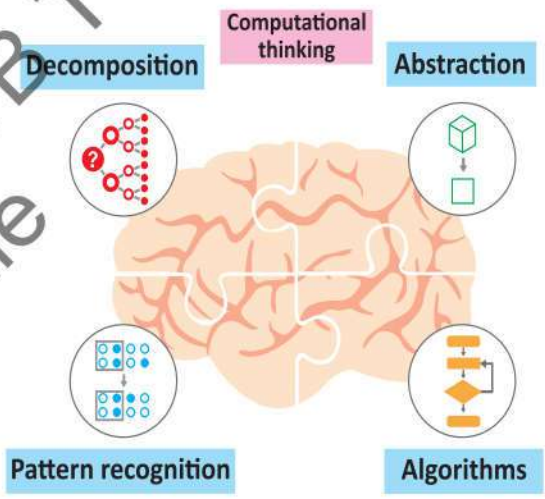
There are four fundamental steps of computational thinking. Let us understand them with the help of some integrated examples.

Computational Thinking Concept	Description	Integrated Example
Decomposition	Analysing and breaking down a problem into smaller parts to solve it in an organised manner and to make it easier to understand and manage.	A person can learn a new language and how to construct sentences using a new foreign language for example by dividing a sentence into parts like a subject, verb, object, etc.
Pattern Recognition	Observing patterns, trends, and regularities in data along with looking for similarities among and within the problem.	In Economics, to find cycle patterns in the rise and drop of the country's economy.

Abstraction	Identifying the general principles that generate these patterns as well as focusing on the important information only, ignoring irrelevant details.	A man operating a vehicle only knows that applying the brakes will make the car stop, but he is unaware of the inner workings of the car or how brakes and other driving controls are truly implemented.
Algorithm Design	Developing a step-by-step approach for solving a problem and similar problems.	The food you eat is the output of the sequential process of making it. The whole recipe is the step-by-step algorithm for making that food.

Thus, computational thinking involves the following steps:

- Taking the complex problem and breaking it down into a series of small, more manageable problems (decomposition).
- Each of these smaller problems is then looked at individually, considering how similar problems have been solved previously (pattern recognition).
- Focusing only on the important details, while ignoring irrelevant information (abstraction).
- Designing the simple steps or rules to solve each of the smaller problems (algorithms).
- Finally, these simple steps or rules are used to program a computer to help solve complex problems in the best way.



ALGORITHMS

An algorithm can be defined as a finite sequence of activities to be processed for getting, a task done or the desired output from a given input. It contains instructions that, when written in any computer language, become a computer program. Thus, an algorithm is important to develop a program.

Even for the same task, two algorithms can perform in different ways. This is because there are currently numerous solutions with various methods. It is comparable to adding water to a bucket. Either use a container to fill it gradually. The alternative is to utilise a pipe that can fill the water all at once. For the same problem, each solution performs differently.

Additionally, this can also help you find a solution in a pool of solutions. Examine all the variables, like cost, time complexity, etc. Considering how little labour will be required, employing a pipe may be the best approach in this case.

Rules for Writing Algorithms

There is a set of rules that you should follow while writing an algorithm.

- It should be clear, exact, and well-defined.
- It should always begin with the word Start and end with the word Stop.
- Each step in an algorithm should be written in a separate line.
- Steps should be numbered as Step 1, Step 2, Step 3, etc.

Some Examples of Simple Algorithms:

Example 1:

Write an algorithm to go for a picnic with your classmates.

Solution:

Step 1 : Start
Step 2 : Decide the picnic venue, date, and time.
Step 3 : Decide the picnic activities.
Step 4 : Rent a vehicle to reach the venue and come back.
Step 5 : Go to the picnic venue on the decided date.
Step 6 : Do the activities planned for the picnic.
Step 7 : Come back to school in the rented vehicle.
Step 8 : Stop

Example 2:

Write an algorithm to make tea.

Answer:

Step 1 : Start
Step 2 : Boil water in a saucepan.
Step 3 : Add tea to boiling water.
Step 4 : Add sugar to boiling water.
Step 5 : Add milk to boiling water.
Step 6 : Boil this water with all the ingredients for 2 mins.
Step 7 : Filter the tea into a cup.
Step 8 : Stop

Example 3

Input the length of two different line segments and check whether they are equal or unequal. Display the message accordingly.

Algorithm

Step 1 : Start

Step 2 : Input the length of the two line segments as l1 and l2.

Step 3 : If l1 and l2 are equal, then print 'Line Segments are equal'.

Step 4 : If l1 and l2 are not equal, then print 'Line Segments are not equal'.

Step 5 : Stop

Do You Know?

If all the instructions in an algorithm are executed in the same order as they are listed, the flow of execution is called as Sequence. The program based on these algorithms will execute their instructions in the same sequence as they were typed.

Example 4:

Write an algorithm m to find if a number is odd or even.

Answer:

Step 1 : Start

Step 2 : Take a number to N.

Step 3 : Divide the number by 2 and store the remainder in R.

Step 4 : If R = 0 Then go to Step 6.

Step 5 : Print 'N is odd' go to step 7.

Step 6 : Print 'N is even'

Step 7 : Stop

Do You Know?

If some of the instructions in an algorithm are executed based on some condition, the flow of execution is called a Selection. The program based on these algorithms will execute their instructions based on the condition.

In the example above, the program will skip the step 5 is the condition R=0 is true. Otherwise, it will skip Step 6.

Some Examples of Complex Algorithms

Example 5:

Write an algorithm to convert a decimal number to a binary number.

Solution:

Step 1 : Start

Step 2 : Divide the number by 2 and store the answer(quotient) and remainder. separately.

Step 3 : Divide the answer of step 2 by 2 and store the answer(quotient) and remainder separately.

Step 4 : Repeat steps 2 and 3 until the number is greater than 0.

Step 5 : Write all the remainder in reverse order.

Step 6 : Stop

Do You Know?

If some of the instructions in an algorithm are executed repeated based on some condition, the flow of execution is called as Iteration or Loop.

The program based on these algorithms will execute their instructions based on the condition repeatedly until the condition remains true. When the condition is false, it will come out of the loop and begin executing next instructions.

In the example above, the program will repeat step 2 and 3 until the quotient is greater than 0.

Difference Between Simple and Complex Problems

Simple Problem

A simple problem is one with a certain source and a certain outcome that is simple to locate and resolve.

Example

Let's say you place food in an oven, but you do not remember to set the timer. When left unattended for too long, it burns. Burnt food is the result, and you may need to eat something else for supper.

Solution:

Establishing an oven timer for your meals is a simple option.

Complex Problem:

A complex issue has various root causes. While some of these factors could be clear, others might be difficult to find.

Example

Raising a Child:

- You do not know much about bringing your first child up.
- While having one child gives you experience, it does not guarantee that you will be successful with the next.
- It neither guarantees success nor is it necessary.
- Each child is different and needs to be treated as an individual.
- The result still is not guaranteed.

The difference between these two problems is that the simple problems can be solved easily while a complex problem's solution is not guaranteed. These types of examples cannot be solved computationally.

Multiple Solution of the Same Problem

In many of the cases, there are always more than one solution available. We must choose the best one. The best one is the solution that will solve our problem in minimum steps, using minimum computer resources.

For example, please read the following problem.

Write an algorithm to find whether a given positive number is prime or not.

Solution 1:

Step 1 : Start

Step 2 : Take a number n as an Input.

Step 3 : If the n is greater than 1, keep on dividing this number with natural number start from 2 and ending with n-1.

Step 4 : If any number from 2 to n-1 divides n and you get the remainder 0, print that the number is not a prime number.

Step 5 : If step 4 fails, print that the number is prime.

Step 6 : Stop

Solution 2:

Step 1 : Start

Step 2 : Take a number n as an Input.

Step 3 : If the n is greater than 1, keep on dividing this number with natural number start from 2 and ending with n/2.

Step 4 : If any number from 2 to n/2 divides n and you get the remainder 0, print that the number is not a prime number.

Step 5 : If step 4 fails, print that the number is prime.

Step 6 : Stop

Solution 3:

Step 1 : Start

Step 2 : Take a number n as an Input.

Step 3 : If the n is greater than 1, keep on dividing this number with natural number start from 2 and ending with √n.

Step 4 : If any number from 2 to √n divides n and you get the remainder 0, print that the number is not a prime number.

Step 5 : If step 4 fails, print that the number is prime.

Step 6 : Stop

Analysis of All Three Solutions

If you look at all three solutions, you will get that all are correct. All solutions find whether the given number is prime or not but still they are different. Now the question is what solution should be selected for problem solving?

By carefully examining the solutions, you will find the following points:

- In first solution, the steps of calculations are starting from number 2 and ending on n-1 value.
- In second solution, the steps of calculations are starting from number 2 and ending on n/2.
- In third solution, the steps of calculations are starting from number 2 and ending on √n.
- Now it is evident that $\sqrt{n} < n/2 < n-1$.
- Hence, we can choose the best solution i.e., the third solution that is offering the least steps and maximum efficiency of the algorithm.

Do You Know?

The speed and efficiency of an algorithm is measured in terms of **Time and Complexity** and **SpaceComplexity** . It means that an algorithm will be efficient if it is taking less processing time and less storage space in the computer system.

That is why, two algorithms for the same problem will perform differently even on same computer system as both are different is using processor time and storage space in memory.

FLOWCHARTS

A flowchart is another problem-solving tool that represents an algorithm in pictorial form. It uses a set of symbols that represents various instructions and shows the sequence and interconnection of functions with the use of lines and arrows.

Symbols Used in a Flowchart

Different symbols used in drawing a flowchart are listed in the following table.

Symbol	Name	Description
Oval	Start/Stop box	represents the beginning and the end of a flowchart
Parallelogram	I/O box	represents the input and output instructions in a flowchart
Rectangle	Process box	represents the processing instructions in a flowchart
Diamond	Decision box	represents the instructions that involve a condition with two options–'Yes' and 'No', to choose from
Lines with Arrowhead	Flow lines	show the direction of flow of data and instructions in a flowchart
Circle	Connector	used to connect various sections of a flowchart

Rules for Drawing a Flowchart

Like an algorithm, certain rules must be followed while drawing a flowchart.

- There can be only one start and one stop symbol in a flowchart.
- Only one flow line can be used with the Start symbol.
- Only one flow line can come out from a process symbol.
- Only one flow line can enter a decision symbol and two flow lines can come out from it.
- The flow lines should not cross each other.
- The direction of the flow of information in a flowchart is either from top to bottom or from left to right.

Examples of Flowcharts

Example 1:

Draw a flowchart for printing "Hello World" once.

Solution:

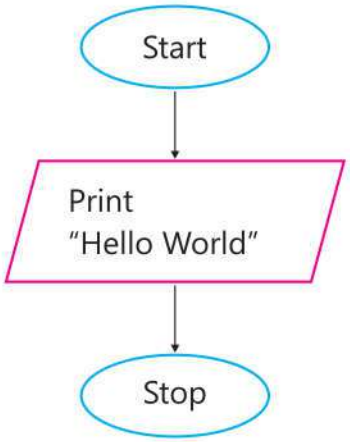
The algorithm for printing 'Hello World' once is as follows:

Step 1: Start

Step 2: Print 'Hello World'.

Step 3: Stop

Each rectangle represents a step in the sequence, and the arrows flow from one step to the next.



Things to Know

In algorithms and flow charts, some terms are used which are briefly defined here:

Declare: Declaring means to reserve memory location for a value,

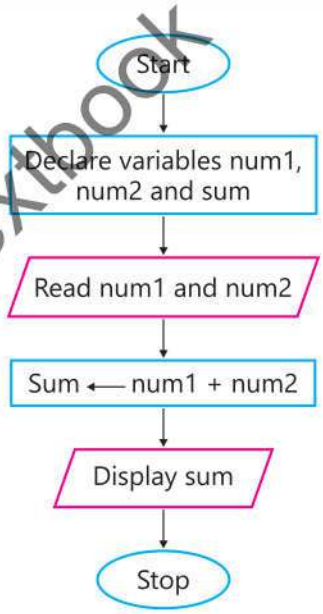
Variable: Variable is the named memory location to store a value temporarily during the execution of a program

Read: Read is the command used to store values in the variables.

% Symbol: It is used to show the remainder in a division process. It is called Mod operator e.g. 5 % 2 =1

Example 2:

Add two numbers entered by the user.



Solution:

The algorithm for the sum of two numbers is as follows:

Step 1: Start

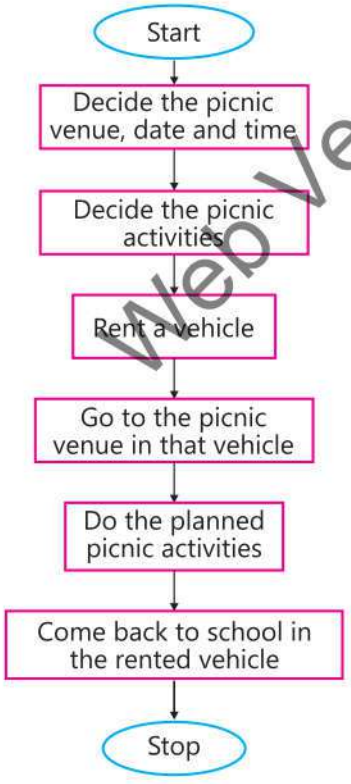
Step 2: Declare variables num1, num 2 and sum.

Step 3: Read the values num1 and num 2

Step 4: Add num1 and num 2 and assign the result to the sum.

Step 5: Display the sum.

Step 6: Stop



Example 3:

Write an algorithm to go for a picnic with your classmates:

Solution:

Step 1: Start

Step 2: Decide the picnic venue, date, and time.

Step 3: Decide the picnic activities.

Step 4: Rent a vehicle to reach the venue and come back.

Step 5: Go to the picnic venue on the decided date.

Step 6: Do the activities planned for the picnic.

Step 7: Come back to school in the rented vehicle.

Step 8: Stop

Example 4:

Write an algorithm to make tea.

Answer:

Step 1: Start

Step 2: Boil water in a saucepan.

Step 3: Add tea to boiling water.

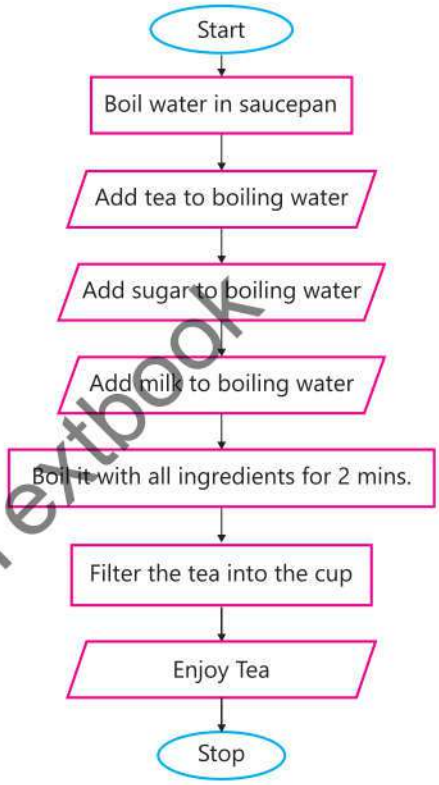
Step 4: Add sugar to boiling water.

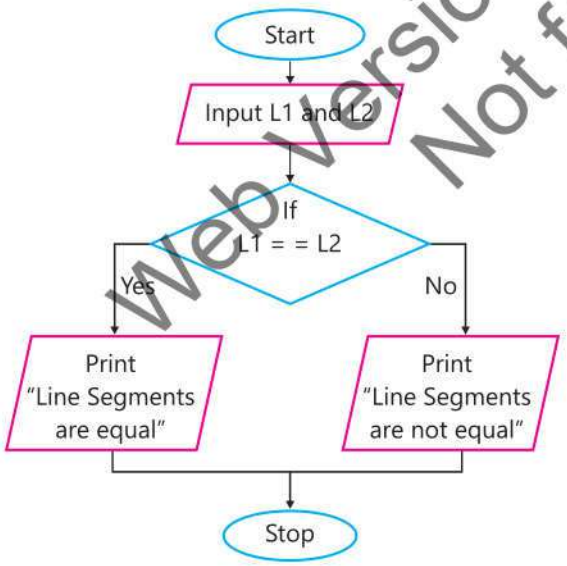
Step 5: Add milk to boiling water.

Step 6: Boil this water with all the ingredients for 2 mins.

Step 7: Filter the tea into a cup.

Step 8: Stop





Example 5

Input the length of two different line segments and check whether they are equal or unequal. Display the message accordingly.

Algorithm

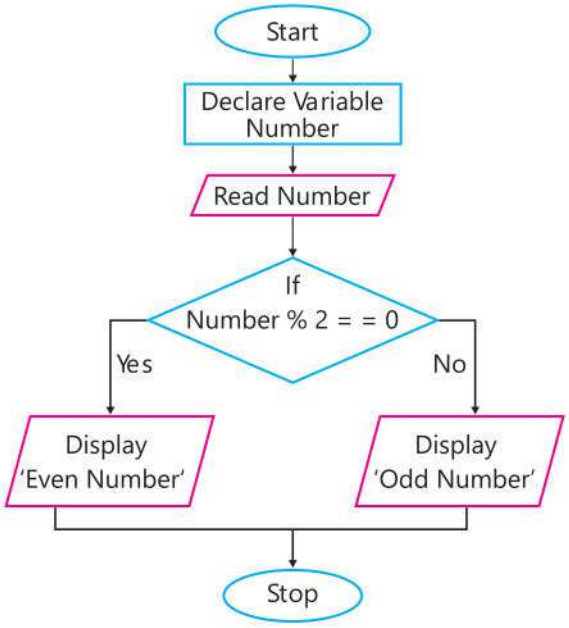
Step 1 : Start

Step 2 : Input the length of the two line segments as L1 and L2.

Step 3 : If L1 and L2 are equal, then print 'Line Segments are equal'.

Step 4 : If L1 and L2 are not equal, then print 'Line Segments are not equal.

Step 5 : Stop



Example 6:

Draw a flowchart to find whether a number is even or odd.

Solution:

Algorithm to find whether a number is even or odd:

Step 1: Start.
Step 2: Declare the variable Number
Step 3: Read the value Number
Step 4: if Number % 2 == 0 then
Print 'Even Number'
Step 5: Otherwise
Print 'Odd Number'
Step 6: Stop

Advantages of Flowcharts

- Expressing an algorithm in a flow chart allows us to see an algorithm in action.
- It forces us to think very carefully about sequencing and selection. Which arrow goes to what node? Are there any missing arrows? Those are the kinds of valuable questions that can come up while translating an algorithm into a flow chart.

Do You Know?

Although using flowcharts is fairly simple, making changes to them is a problem. Most likely, you'll have to redo the entire flowchart.

COMPUTATION THINKING in PRACTICE

Before computers can be used to solve a problem, the problem itself and how it could be resolved must be understood. Computational thinking helps with these tasks. It enables the programmer to work out exactly what to instruct the computer to do. Once it is clear, the solution can be converted into the programming language.

Look at the given table to find the advantages and disadvantages of computational thinking.

Terms	Advantages	Disadvantages
Thinking abstractly	Allows us to make predictions	Predicting markets, users, trends, and technical factors could be challenging. Too many factors that are changeable may mean the scenario is too complex to model accurately.
Thinking ahead	Caching can speed up a process	Caching can be complicated to implement. Caching requires the correct data to be fetched for the next task.
Thinking procedurally/ Decomposition	Can be done with flowcharts, etc.	It may not be entirely possible with an event-driven approach rather than a procedural approach to problem solving.
Thinking concurrently	Concurrency speeds up the solution.	It may be difficult to program

Hence, we can say that it is not possible to solve all the problems computationally.

Some Problems Cannot be Solved Computationally

Some problems cannot be solved computationally. Some of these examples include:

Factoring Large Numbers in Small Time

The factoring problem asks you to identify the prime factors of an odd integer. It is an example of a computational challenge. The factorization problem is said to not be solved computationally. The factoring effort increases exponentially as the size of the input integer increases.

The Halting Problem

The halting problem is a decision problem. Halting means terminating or stopping our program after a certain execution. We are unable to create a generic algorithm that can accurately predict whether a certain program will ever stop or not. Additionally, there is no generic algorithm that can tell us whether a certain program will complete its execution and stop. The best approach is to run the application and check to see if it stops or halts.

Keep in mind that if our program does not halt, it will keep the CPU busy forever. In programming, we say it is an infinite loop. Our programs should never get into infinite loops otherwise, the system will not work properly.

PSEUDOCODE

A collection of instructions to solve a problem simply described in plain English is called Pseudocode. Pseudocode can be viewed as a collection of well-organised ideas for solving issues. It is an informal way to describe problem-solving instructions. It also serves as a

communication option that can help you explain your ideas to other people.

Although there are no set guidelines for writing pseudocode, it can be useful to use common terms like input, output, if-then-else, while-do, for, etc. Coders often use pseudocode as an intermediate step in programming in between the initial planning stage and the stage of writing actual executable code.

Writing a Basic Structured Pseudocode

It can be easy to convert pseudocode into actual code once you program. We must remember the purpose of our pseudocode and explain what each line of the program should do. This will keep us focused while creating the pseudocode document.

Following are some basics of writing pseudocode:

- Put the tasks in the proper order, then start writing.
- Start with a statement that states the primary objective.
- Use the proper sentence case for each type of word.
- Apply the proper naming conventions. The name must be clear and concise
- It should be easy enough for a layman to understand.
- Write everything that will occur in the real code without abstraction.

It can be easy to revert to writing in code once you plan your program. We must remember the purpose of our pseudocode and explain what each line of the program should do. This will keep us grounded while creating the pseudocode document.

A Pseudocode to Print the Sum of two Integer Numbers

Step 1: Use plain-text editors. Here we have opened Notepad.

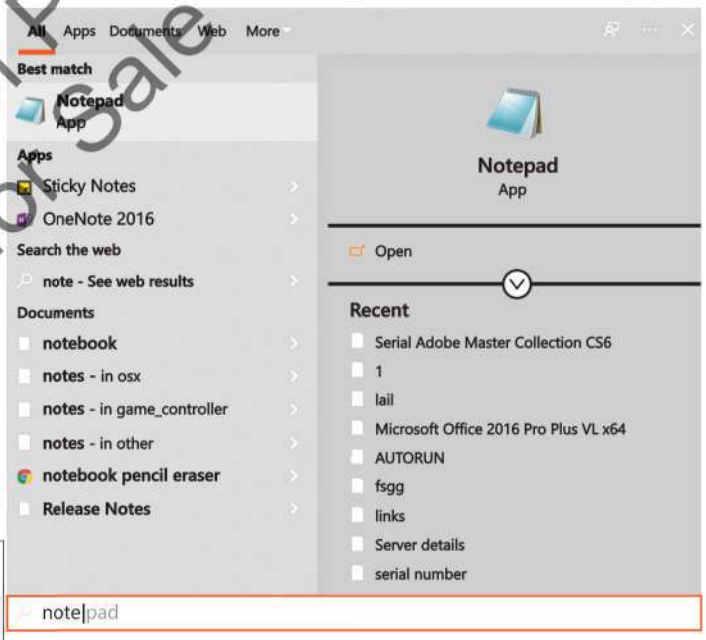
Step 2: Write down the purpose of the process.

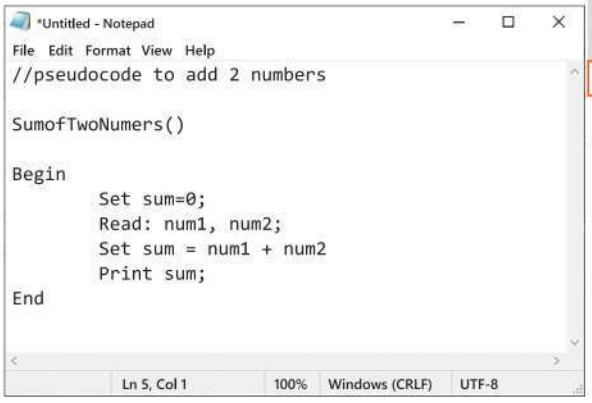
Step 3: Write the opening command.

Step 4: Start the program.

Step 5: Set the variable 'sum'

Step 6: Read the first number 'num1' and the second number 'num2'.





Step 7: Sum 'num1' and 'num2' and save the result in the variable 'sum'.
sum -> num1 + num2

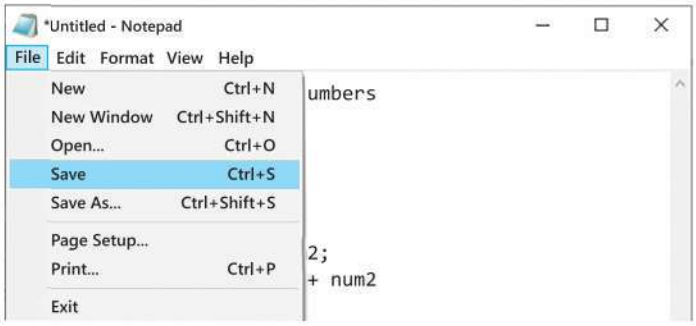
Step 8: Print the variable 'sum'..

Step 9: End the program.

As per this method's example, your final pseudocode document should look something like this:

Step 10: Press Ctrl + S to save your document. Enter a name and click Save to do so.

You have created your pseudocode well.



Conditions in Problem Solving

Conditions are the key factor in problem solving. Conditions tell us to take the right path based on some test. These tests may be simple tests like checking the value of a statement or selecting the right key for your home door lock from a bunch of keys. These conditions decide which tasks will be performed if the condition is true. For example, if the right key for the door lock is chosen, the lock will be opened.

The way you can differentiate between the problems where the 'if', 'if then else, and if with multiple conditions can be applied depends on the problem.

if condition:

In the 'if' condition, a statement or a block of statements are only executed if a specific condition is true. Otherwise, neither the statement nor the block of statements is executed. For example, consider that if it is cold outside; we would take tea.

'if then else' condition:

In the 'if then else' condition, a statement or a block of statements under the 'if' are only executed if a specific condition is true; otherwise, it goes for the 'else' statement or block of statements and executes it. So 'else' represents otherwise.

Consider this example: if it is cold outside; we would take tea, 'else' we would enjoy ice cream.

Here we will take tea if the condition of cold outside gets true. Otherwise, we will enjoy ice cream. Both actions can not be done at the same time.

Iterations in Problem Solving

Sometimes we want to repeat an action again and again, it is called Iteration. It is also referred to as loops. Loops allow you to repeat an action or set of actions until a particular condition is satisfied. Consider the example where you are asked to collect the balls in a hall and put them in a box. You will go to the ball, collect it, and put it in the box. Then you will go to the next ball, collect it, and put it in the box. You will repeatedly perform the same task until all balls are collected and put in the box.

These types of situations are called iterations or loops. In algorithms, we use Repeat or Repeat

Forever to represent these loops. Repeat tells to repeat the instructions until a condition is met. Repeat Forever loops are the loops that execute an instruction or a set of instructions infinitely. For example, the security algorithms keep on working to secure your system.

Example:

Write an algorithm that prints even numbers from 2 to 100.

Step 1: Start

Step 2: Declare the value 2 to Num

Step 3: Print the value Num

Step 4: Add 2 to Num

Step 5: If the value of Num <=100, Repeat Steps 2 and 3.

Step 6: Stop

This algorithm will keep on printing the value of Num i.e. 2,4,6... until the value is 100. Then it will stop.

Concept of Nesting

Nesting is the placement of one object within another object. In problem solving, different elements may be placed within other elements. For example, you may place conditions within other conditions, or loops within other loops.

Problem solving is made powerful yet easy because of nesting. The number of steps required is decreased.

Nested Conditions:

Conditions may be placed within other conditions to solve a problem. These conditions will have many 'if' condition blocks that are checked line by line until a condition is true. If it is not, then it executes the 'else' block at the end.

Example:

Step 1: Start

Step 2: Declare three values a, b and c

Step 3: Read the values a, b and c

Step 4: Check if a is greater than b

Step 5: If true, then check if a is greater than c

1. If true, then print "a" as the greatest number
2. If false, then print "c" as the greatest number

Step 6: If false, then check if b is greater than c

1. If true, print "b" as the greatest number
2. If false, print "c" as the greatest number

Step 7: Stop

Nested Iterations or Loops:

Just like the nested conditions, you can also nest the loops. Nesting can be done to any level. You can use as many loops within other loops as you want but increasing the level of nesting will result in increasing complexity.

Example:

An example of the way nested iteration or loops can be:

```
repeat (condition)
{
        repeat (condition)
        {
                statement(s)
        }
}
```

Glossary

step-by-step	to progress slowly from one stage to the next	computational	using computers
characteristic	a typical quality or feature of a person, place, or thing	pictorial	illustrated in pictures
perspective	a particular outlook regarding something	concurrent	happening or done simultaneously
regularity	something regular	halt	to stop
finite	to have a limit	informal	unofficial or not formal

LET'S HAVE A LOOK

- Problem-solving refers to the process of finding solutions to problems that you come across in everyday life.
- Computational thinking is a set of problem-solving processes that allows the user to take a complex problem, understand what the problem is and develop possible solutions.
- There are four fundamental steps of computational thinking–decomposition, pattern recognition, abstraction, and algorithm design.
- An algorithm can be defined as a sequence of activities to be processed for getting a task done or the desired output from a given input.
- A flowchart is another problem-solving tool that represents an algorithm in pictorial form.
- In the halting problem, we are unable to create a generic algorithm that can accurately predict whether a certain program will ever stop or not.
- Pseudocode can be viewed as a collection of well-organised ideas for solving issues. It is an informal way to describe problem-solving instructions.

Exercise

A. Multiple Choice Questions: Tick the correct answer.

1. Developing a step-by-step approach for solving a problem is:

   a. Decomposition **b.** Abstraction

   c. Algorithm Design **d.** Pattern Recognition

2. _____ allows us to take a complex problem, understand what the problem is and develop possible solutions.

   a. Computational thinking **b.** Excel **c.** Formulas **d.** None of these

3. Observing patterns, trends, and regularities in data along with looking for similarities among and within the problem are:

   a. Decomposition **b.** Abstraction

   c. Algorithm Design **d.** Pattern Recognition

4. Focusing only on the important details, while ignoring irrelevant information is:

   a. Decomposition **b.** Abstraction

   c. Algorithm Design **d.** Pattern Recognition

5. Sometimes we want to repeat an action again and again, which is called _____.

   a . iteration **b.** solution **c.** copying **d.** deletion

6. There can be only one start and ___ stop symbol in a flowchart.

   a. one **b.** two **c.** three **d.** four

7. The Start/Stop box is represented by _____.

   a. an oval **b.** a parallelogram **c.** a rectangle **d.** a diamond

8. The decision box is represented by _____.

   a. an oval **b.** a parallelogram **c.** a rectangle **d.** a diamond

9. What is the full form of CT?

   a. Computer Technology **b.** Computational Thinking

   c. Computer Tomography **d.** None of these

10. _____ is the placement of one object within another object.

a. Halting b. Flowchart c. Nesting d. None of these

B. Write "T" for true or "F" for false statements.

1. An algorithm always begins with the word Start. ☐
2. A diamond shows the direction of the flow of data and instructions in a flowchart. ☐
3. Computational thinking does not allow the development of solutions to a problem. ☐
4. Each step in an algorithm should be written in a separate line. ☐
5. The flow lines can cross each other. ☐
6. Like an algorithm, certain rules must be followed while drawing a flowchart. ☐
7. Concurrency slows down the solution. ☐

C. Answer the following questions:

1. Explain computational thinking. Why do you think it is important to learn?
2. Differentiate between decomposition and abstraction.
3. What are the rules to write an algorithm?
4. What do you think is the process to boil water? Write an algorithm and draw a flowchart for boiling the water.
5. Explain the factoring problems.
6. What is a pseudocode?
7. How are algorithms used in real life?
8. Are pseudocodes better than flowcharts? If yes then explain why.

D. Answer the questions comprehensively:

1. Write two pros and cons of computational thinking.
2. What do you mean by the halting problem? How can we prove that the halting problem is undecidable?

D. Define the following terms:

1. Computational thinking
2. Decomposition
3. Abstraction
4. Flowchart

E. Application-based questions:

1. Ali and his friends are playing tic-tac-toe. What will be his main strategy to win against his opponents?
2. Bilal left his house at the crack of dawn. He looks at his watch to see what time it is. He takes the bus to get to school if it is before 7 a.m.; otherwise, he takes the subway. Create a flowchart for this scenario.

Learning Activities

1. Represent algorithms using pseudocodes.

• Activity 1:

Provide a set of instructions or pseudocode. Using graph paper, follow the instructions and draw the output on a grid. Start from a point on the top left corner, initially 46 facing East and the distance between two points on the graph paper is 50 steps

The list of instructions/pseudocode is:

- Move forward by 100 steps
- Turn right by 90 degrees
- Move forward by 100 steps
- Turn right by 90 degrees
- Move forward by 100 steps
- Turn right by 90 degrees
- Move forward by 100 steps
- Turn right by 90 degrees

- If the activity has been done correctly the students are facing in the same direction as originally and should have drawn a square.

2. Activities that describe that there are ways to characterise how well algorithms perform and that two algorithms can perform differently for the same task.

• Activity 3:

Solve the Sudoku puzzle. The rules of the game are very easy. For the 4 x 4 game, the numbers 1 to 4 must be filled in each row, column, and smaller 2 x 2 boxes. Each number in a row, column, or box must be used exactly once! A 4 x 4 Sudoku puzzle looks as follows.

	1		
		2	
	3		
		1	

The main idea is to consider all the valid possible solutions and then eliminate the options that do not satisfy the rules of the game

• Activity 4:

Ali, Mariam, Neha, and Usman were the only four participants in a cake-baking competition; they placed 1st, 2nd, 3rd, and 4th in some order. Also,

1. Mariam was not first, and Usman was not last.
2. Neha was placed next to neither Mariam nor Usman;
3. Ali scored better than Mariam.

Based on these clues, who placed 1st, 2nd, 3rd, and 4th?

3.Apply the best possible solution to a problem from a pool of solutions.

• Activity 5:

A man finds himself on a riverbank with a wolf, a goat, and cabbage. He needs to transport all three to the other side of the river in his boat.

However, the boat has room for only the man himself and one other item (either the wolf, the goat, or the cabbage). In his absence, the wolf would eat the goat, and the goat would eat the cabbage.

Show how the man can get all these 'passengers' to the other side. ("Puzzle | Farmer, Goat, Wolf and Cabbage - GeeksforGeeks", 2022)

• Activity 6:

Once upon a time, there was a city that had no roads. Getting around the city was particularly difficult after rainstorms because the ground became very muddy -- cars got stuck in the mud and people got their boots dirty.

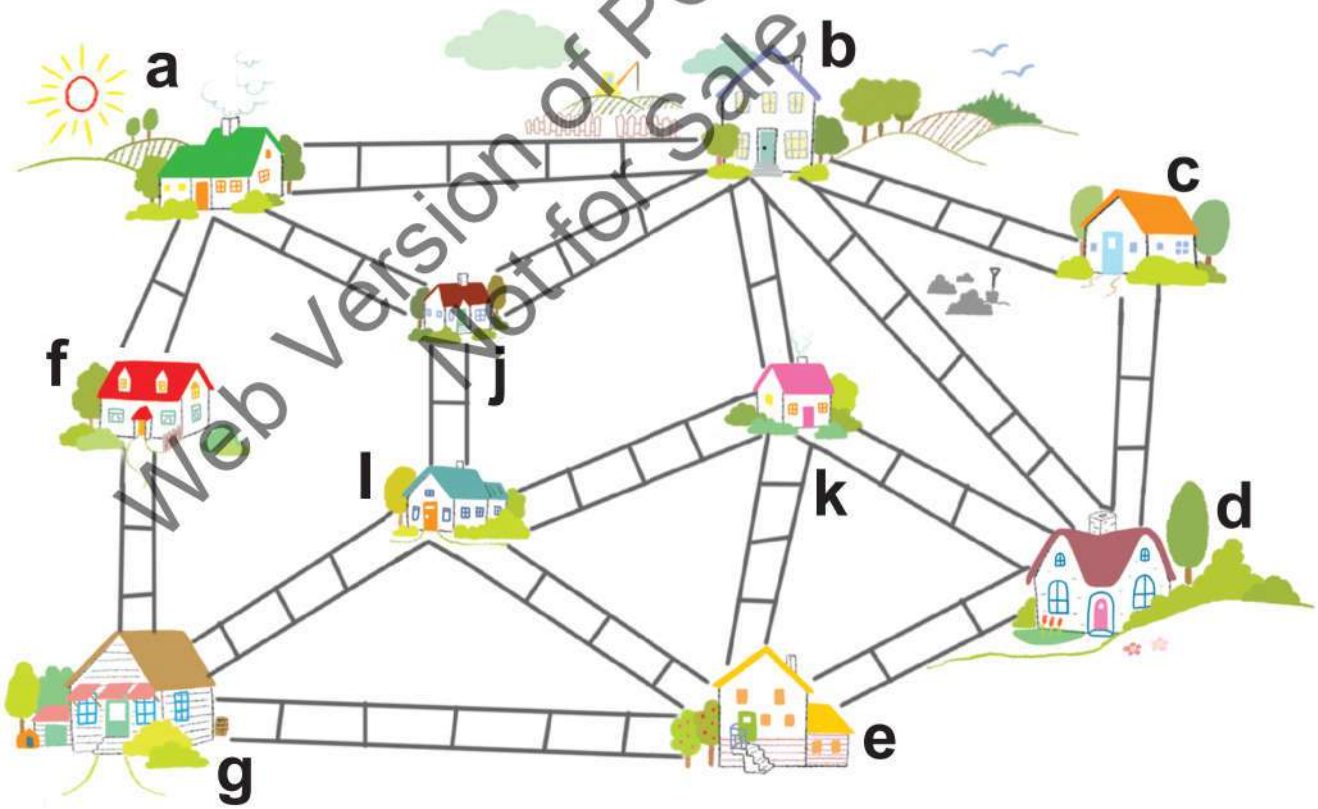
The mayor of the city decided that some of the streets must be paved but did not want to spend more money than necessary because the city also wanted to build a road.

The mayor, therefore, specified two conditions:

1. Enough streets must be paved so that everyone can travel from their house to anyone else's house only along paved roads.

2. The paving should cost as little as possible. Use as few stones as possible. On the map of the city, the number of paving stones between each house represents the cost of paving that route. Find the best route that connects all the houses but uses as few counters 47 (paving stones) as possible. What kind of strategies did you use to solve the problem?

Enough streets must be paved so that everyone can travel from their house to anyone else's house only along paved roads. The paving should cost as little as possible. Use as few stones as possible. ("The Muddy City Problem", 2022)



In Class Activity

1. Write the pseudocode of any 3 different algorithms.
2. Design an algorithm of 3 different problems (easy, difficult, and complex) and translate them into pseudocodes.
3. Identify whether the problem can be solved or not? If not write the reasons, if yes write the reasons.
4. You have nine cards of the following colours. We need to arrange these cards into three rows and three columns, for example, blue, orange, teal, blue, green, blue, gold, teal, green,
   We also want the following rules to be satisfied:
   - The two green cards are on the left.
   - The two teal cards are at the bottom.
   - The three blue cards are at the top.
   - The orange card is on the right.

   Based on these rules, arrange the nine cards in the grid.
5. A farmer is on his way back from the market, with him he has a fox, a chicken, and some grain. When he reaches a river crossing he must use a small boat only big enough for him and one other item.

   Unfortunately, if the fox is left alone with the chicken it will eat it, as will the chicken eat the grain? Explain how the farmer can cross the river. ("Crossing a river in a boat with some grain, 45 a chicken and a fox.", 2022)
6. In the eighteenth century the city we now know as Kaliningrad was called Königsberg and it was part of Prussia. Like many other great cities, Königsberg was divided by a river, called the Pregel. It contained two islands and seven bridges linking the various land masses.

   A famous puzzle at the time was to find a walk through the city that crossed every bridge exactly once — the path wasn't allowed to cross any bridge more than once, and it wasn't allowed to leave any bridge out. Apply graph theory to determine the solution to this problem. ("The Bridges of Königsberg", 2022)
7. Students should be able to write at least 2 different solutions to the same problem and identify which one is the best algorithm to solve the problem and why.

   (This activity can be a group/ class activity where students can identify their solutions and write the algorithm and pseudocode and then decide which solution is best and why?)

Bibliography

- Denning, Peter J., and Matti Tedre. Computational Thinking. MIT Press, 2019.
- Beecher, Karl. Computational Thinking a Beginner's Guide to Problem-Solving and Programming. BCS : British Computer Society, the Chartered Institute for IT, 2018.
- Convert the Following Algorithm to Flowchart. Step 1 ... - Brainly.in. **https://brainly.in/question/22461693.**
- 'Expressing an Algorithm | AP CSP (Article).' Khan Academy, Khan Academy, **https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/building-algorithms/a/expressing-an-algorithm** .
- Lloyd, Jack. 'How to Write Pseudocode: 15 Steps (with Pictures).' WikiHow, WikiHow, 7 May 2022, **https://www.wikihow.com/Write-Pseudocode**.
- 'The World's Most Engaging Learning Platform.' Quizizz, **https://quizizz.com/admin/quiz/6058b1e1ac823e001f26beed/computational-thinking** .
- 'What Is Computational Thinking? - Introduction to Computational Thinking - KS3 Computer Science Revision - BBC Bitesize.' BBC News, BBC, **https://www.bbc.co.uk/bitesize/guides/zp92mp3/revision/1** .
- Wu, Lei, et al. 'Realistic Drawing & Painting with Ai-Supported Geometrical and Computational Method (Fun - Joy $$ \Mathbf{\Mathit{f}}\Mathbf{Un}\Hbox{-} \Mathbf{\Mathit{j}}\Mathbf{Oy} $$).' SpringerLink, Springer International Publishing, 1 Jan. 1970, **https://link.springer.com/chapter/10.1007/978-3-030-70296-0_58.**
- Learning Tools, Flashcards, and Textbook Solutions | Quizlet. **https://quizlet.com/**.
- KnowledgeBoat. 'Write Algorithms and Draw Flowcharts for the Following Accept The.' KnowledgeBoat, 24 Jan. 2021, **https://www.knowledgeboat.com/question/write-algorithms-and-draw-flowcharts-for-the-following-accept-the--19597626725502950**.
- Class 6 Chapter 7: Programming Basics - Exploringit.net. **http://exploringit.net/book2/class6/Chapter7.pdf** .

Answers

A: Tick the correct option.

1	c	2	a	3	d	4	b	5	a
6	a	7	a	8	d	9	b	10	c

B: True and False

1	T	2	F	3	F	4	T	5	T
6	T	7	F						