

when you have a problem \_ What should you do? First, you need to identify the problem and then apply a strategy for solving the problem.

Practicing different problem-solving strategies can help professionals develop efficient solutions to challenges they encounter at work and in their everyday lives.

Before finding a solution to the problem, the problem must first be clearly identified. After that, one of many problem solving strategies can be applied, hopefully resulting in a solution

A problem-solving strategy is a plan used to find a solution or overcome a challenge.

Each problem-solving strategy includes multiple steps to provide you with helpful guidelines on how to resolve a business problem or industry challenge. Effective problem-solving requires you to identify the problem, select the right process to approach it and follow a plan tailored to the specific issue you are trying to solve.

Problems themselves can be classified into two different categories known as ill-defined and well-defined problems (Schacter, 2009).

Ill-defined problems represent issues that do not have clear goals, solution paths, or expected solutions

well-defined problems have specific goals, clearly defined solutions, and clear expected solutions.

also in many cases require abstract thinking and creativity in order to find novel solutions. Various methods of studying problem solving exist including introspection, simulation, computer modelling, and experimentation.

By mastering several problem-solving strategies, you can more effectively select the right plan of action when faced with challenges in the future. This can help you solve problems faster and develop stronger critical thinking skills.

## 1. Trial and Error

A trial-and-error approach to problem-solving involves trying a number of different solutions and ruling out those that do not work. This approach can be a good option if you have a very limited number of options available.

When using trial and error, one would continue to try different solutions until the problem is solved. Although trial and error is not typically one of the most time-efficient strategies, it is a commonly used one.

## 2. Algorithm and Heuristic

A common type of strategy is an algorithm. An algorithm is a problem-solving formula that provides you with step-by-step instructions used to achieve a desired outcome (Kahneman, 2011).

#### Heuristic

A heuristic is another type of problem solving strategy. While an algorithm must be followed exactly to produce a correct result, a heuristic is a general problem-solving framework (Tversky & Kahneman, 1974).

n. Different types of heuristics are used in different types of situations, but the impulse to use a heuristic occurs when one of five conditions is met (Pratkanis, 1989):

- When one is faced with too much information
- When the time to make a decision is limited
- When the decision to be made is unimportant
- When there is access to very little information to use in making the decision
- When an appropriate heuristic happens to come to mind in the same moment.

#### 3. Means-Ends Analysis

This strategy involves choosing and analysing an action at a series of smaller steps to move closer to the goal. One example of means-end analysis can be found by using the Tower of Hanoi paradigm. This paradigm can be modelled as a word problem.

#### Other Problem-solving Strategies

##### i. Use past experience

Take the time to consider if you have encountered a similar situation to your current problem in the past. This can help draw connections between different events. Ask yourself how you approached the previous situation and adapt those solutions to the problem currently being solved.

##### ii. Bring in a facilitator

If one is trying to solve a complex problem with a group of other people, bringing in a facilitator can help increase efficiency and mediate collaboration. Having an impartial third party can help a group stay on task, document the process and have a more meaningful conversation. Consider inviting a facilitator to your next group meeting to help generate better solutions.

##### iii. Develop a decision matrix for evaluation

If multiple solutions are developed for a problem, one may need to determine which one is the best. A decision matrix can be an excellent tool to help you approach this task because it allows you to rank potential solutions.

Some factors you can analyse when ranking each potential solution are:

Timeliness  
Risk  
Manageability  
Expense  
Practicality  
Effectiveness

iv. Ask your peers for help

Getting opinions from peers can expose new perspectives and unique solutions. Friends, families or colleagues may have different experiences, ideas and skills that may contribute to finding the best solution to a problem.

v. Step away from the problem

Finally, if the problem being worked on does not need an immediate solution, consider stepping away from it for a short period of time. You can do this literally by taking a walk to help clear your mind or figuratively by setting the problem aside for a few days until you are ready to approach it again.

Allowing yourself time to rest, exercise and take care of your own well-being can make solving the problem easier when you come back to it because you may feel energised and focused.

problem-solving is not a flawless process.

There are a number of different obstacles that can interfere with the ability to solve a problem quickly and efficiently. These include functional fixedness, irrelevant information, and assumptions.

Problem-solving strategies which may include multiple steps in order to proffer solution to business problem or industrial challenges.

Effective problem-solving requires you to identify the problem, select the right process to approach it and follow a plan tailored to the specific issue you are trying to solve

Understanding the strategies of proffering solutions to problem through trial and error, algorithm, heuristic and means-ends analysis.

Applying Tower of Hanoi to solve strategy which involves choosing and analysing an action at a series of smaller steps to move closer to the goal

Problem Solving is the sequential process of analysing information related to a given situation and generating appropriate response options.

Steps to problem solving

## 1. Understand the Problem

It sounds strange, but the first step to solving any problem is to make sure that one understands the problem about to be solved. One needs to know:

What input data/information is available?

What does the data/information represent?

In what format is the data/information?

What is missing in the data provided?

Does the person solving the problem have everything needed?

What output information needs to be produced?

In what format should the result be: text, picture, graph?

What are the other requirements needed for computation?

## 2. Formulating a Model

The next step is to understand the processing part of the problem. Many problems break down into smaller problems that require some kind of simple mathematical computations in order to process the data.

## 3. Develop an Algorithm

Having understood the problem and formulated a model, it is time to come up with a precise plan of what the computer is expected to do.

Algorithm is a precise sequence of instructions for solving a problem.

## 4. Writing the Program

Now that we have a precise set of steps for solving the problem, most of the hard work has been done. The next step is to transform the algorithm from step 3 into a set of instructions that can be understood by the computer.

semi-colon

characters (;) is what it understands to be the end of an instruction. Leaving one of them off will cause the program to generate what is known as a compile-time error.

Compiling is the process of converting a program into instructions that can be understood by the computer.

The longer a program is, the more the likelihood of having multiple compile-time errors. One needs to fix all such compile-time errors before continuing on to the next step.

## 5. Test the Program

Once a program is written and compiles, the next task is to make sure that it solves the problem that it was intended to solve and that the solutions are correct.

Bugs are errors with a program that cause it to stop working or produce incorrect or undesirable results.

Debugging is the process of finding and fixing errors in program code

## 6. Evaluating the Solution

Once the program produces a result that seems correct, the original problem needs to be reconsidered to make sure that the answer is formatted into a proper solution to the problem.

In this unit you learnt that the various stages involve in the problem solving processing: The stages are sequential and are seven in number:

Computer as a model of computation

Understanding the Problem

Formulating a Model

Developing an Algorithm

Writing the Program

Testing the Program

Evaluating the Solution

To be able to solve a problem using computational approaches, the problem itself needs to have certain characteristics:

The problem needs to be clearly defined — this means that one should be able to identify the current situation, the end goal, the possible means of reaching the end goal, and the potential obstacles

The problem needs to be computable — one should consider what type of calculations are required, and if these are feasible within a reasonable time frame and processing capacity

The data requirements of the problem need to be examined, such as what types of data the problem involves, and the storage capacity required to keep this data

One should be able to determine if the problem can be approached using decomposition and abstraction, as these methods are key for tackling complex problems.

Computational approaches available for solving a problem

### 1. Brute-force Approach

This strategy is characterised by a lack of sophistication in terms of their approach to the solution. It typically takes the most direct or obvious route, without attempting to minimise the number of operations required to compute the solution.

### 2. Divide-and-conquer Approach

In the divide and conquer strategy, a problem is solved recursively by applying three steps at each level of the recursion: Divide, conquer, and combine.

#### Divide

Divide is the first step of the divide and conquer strategy. In this step the problem is divided into smaller sub-problems until it is small enough to be solved.

## Combine

In this step, the solution of the sub-problems is combined to solve the whole problem. The output returned from solving the base case will be the input of larger sub-problems.

## 3. Dynamic Programming Approach

Dynamic programming approach is similar to divide-and-conquer in that both solve problems by breaking it down into several sub-problems that can be solved recursively. The difference between the two is that in the dynamic programming approach, the results obtained from solving smaller sub-problems are reused in the calculation of larger sub-problems.

## 4. Greedy Algorithm Approach

In a greedy algorithm, at each decision point the choice that has the smallest immediate (i.e., local) cost is selected, without attempting to look ahead to determine if this choice is part of our optimal solution to the problem as a whole (i.e., a global solution).

### Characteristics of the Greedy Algorithm

The important characteristics of a Greedy algorithm are:

1. There is an ordered list of resources, with costs or value attributions. These quantify constraints on a system.
2. Take the maximum quantity of resources in the time a constraint applies.
3. For example, in an activity scheduling problem, the resource costs are in hours, and the activities need to be performed in serial order.

## 5. Randomized Approach

This approach is dependent not only on the input data, but also on the values provided by a random number generator.

## The Concept of Abstraction

This is the creation of well-defined interfaces to hide the inner workings of computer programs from users. It may also be defined as the process of identifying the general characteristics needed to solve a problem while filtering out unnecessary information.

Abstraction is widely used to simplify things that may be very complex. We use abstractions all the time, almost without thinking. For example, if you learn to drive, you will be taught that putting your foot on the accelerator will speed up the car and putting your foot on the brake pedal will slow it down. You will not be taught anything about how the acceleration or braking systems actually work.

### Importance of Abstraction

In computer science, abstraction is used to manage the complexity of a lot of what is designed and created. Computer hardware is seen as components or black boxes.

### How to Abstract

In computing, when we decompose problems, we then look for patterns among and within the smaller problems that make up the complex problem. Abstraction allows us to create a general idea of what the problem is and how to solve it. We remove all specific detail, and any patterns that will not help us solve the problem.

## Types of Abstraction

### 1. Representational Abstraction

Abstraction appears in many forms within computing, both in terms of techniques used to approach problem-solving, and in the computational tools employed to develop solutions.

### 2. Abstraction by Generalisation

When you group things in terms of a set of common characteristics, you are generalising. This is a fundamental technique used in object-oriented programming (although it is not exclusive to OOP) when you identify that some objects are 'kinds of' more generic objects.

### 3. Procedural abstraction

This type represents a computational method. One of the skills that you will develop as a computer scientist is the ability to design a well-abstracted procedure that is generalised as far as possible.

### 4. Functional Abstraction

In functional abstraction, the implementation detail of the computational method is hidden. You can think of a function as a black box. The function will receive an input (or set of inputs), process the input(s), and return the output. How the transformation is achieved is hidden from the user?

### 5. Data Abstraction

Some data types, such as unsigned integers, are conceptually simple; others are more complex. Data abstraction is a technique that allows you to separate the way that a compound data object is used, from the details of how it is constructed.

Abstraction is one of the four cornerstones of computer science.

It is important in the study of

computing and problem solving and involves identification of critical aspects of the problem environment and the required system.

An algorithm is an effective step-by-step procedure for solving a problem in a finite number of steps. In other words, it is a finite set of well-defined instructions or step-by-step description of the procedure written in human readable language for solving a given problem. An algorithm itself is division of a problem into small steps which are ordered in sequence and easily understandable. Algorithms are very important to the way computers process information, because a computer program is basically an algorithm that tells

computer what specific tasks to perform in what specific order to accomplish a specific task. The same problem can be solved with different methods.

### Reasons for Algorithm

A programmer writes a program to instruct the computer to do certain tasks as desired. The computer then follows the steps written in the program code.

### Steps Involved in Algorithm Development

An algorithm can be defined as a complete, unambiguous, finite number of logical steps for solving a specific problem

Step1. Identification of input: For an algorithm, there are quantities to be supplied called input and these are fed externally. The input is to be identified first for any specified problem.

Step2: Identification of output: From an algorithm, at least one quantity is produced, called for any specified problem.

Step3: Identify the processing operations: All the calculations to be performed in order to lead to output from the input are to be identified in an orderly manner.

Step4: Processing Definiteness: The instructions composing the algorithm must be clear and there should not be any ambiguity in them.

Step5: Processing Finiteness: If we go through the algorithm, then for all cases, the algorithm should terminate after a finite number of steps.

Step6: Possessing Effectiveness: The instructions in the algorithm must be sufficiently basic and in practice they can be carried out easily.

### Characteristics of Algorithm

An algorithm must possess following characteristics:

1. Precision — the steps are precisely stated or defined.
2. Uniqueness — results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
3. Finiteness — the algorithm always stops after a finite number of steps.
4. Input — the algorithm receives some input.
5. Output — the algorithm produces some output.

### Representation of Algorithms

Using their algorithmic thinking skills, software designers or programmers analyse the problem and identify the logical steps that need to be followed to reach a solution.

There are two common methods of representing an algorithm —flowchart and pseudocode.

### Measuring Efficiency of Algorithms

It may be possible to solve the problem in more than one way, resulting in more than one algorithm. The choice of various algorithms depends on the factors like reliability, accuracy

and easy to modify.

Algorithm efficiency is used to describe properties of an algorithm relating to how much of various types of resources it consumes.

The runtime complexity (execution time) of an algorithm is the amount of time that it takes to complete once it has begun.

The space complexity of an algorithm is the amount of storage space that it requires while running from start to completion.

## Advantages and Disadvantages of Algorithm

### Advantages

Designing an algorithm has following advantages:

1. **Effective Communication:** Since algorithm is written in English like language, it is simple to understand step-by-step solution of the problems.
2. **Easy Debugging:** Well-designed algorithm makes debugging easy so that we can identify logical error in the program.
3. **Easy and Efficient Coding:** An algorithm acts as a blueprint of a program and helps during program development.
4. **Independent of Programming Language:** An algorithm is independent of programming languages and can be easily coded using any high level language.

### Disadvantages

An algorithm has following disadvantages:

1. Developing algorithm for complex problems would be time consuming and difficult to understand.
2. Understanding complex logic through algorithms can be very difficult.

A flowchart is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

## Areas for using Flowcharts

Sales & Marketing

Manufacturing

Business Operations

Software Engineering or Programming

## Differences between Algorithm and Flowchart

### algorithm

1. A method of representing the step-by-step logical procedure for solving a problem
2. It contains step-by-step English descriptions, each step representing a particular operation leading to solution of problem
3. These are particularly useful for small Flowchart

### flowchart

1. Flowchart is diagrammatic representation of an algorithm. It is constructed using different types of boxes and symbols.
2. The flowchart employs a series of blocks and arrows, each of which represents a particular step in an algorithm
3. These are useful for detailed

## Advantages of Flowcharts

1. The flowchart shows the logic of a problem displayed in pictorial fashion which facilitates easier checking of an algorithm.
2. The Flowchart is good means of communication to other users. It is also a compact means of recording an algorithm solution to a problem.
3. The flowchart allows the problem solver to break the problem into parts. These parts can be connected to make master chart.
4. The flowchart is a permanent record of the solution which can be consulted at a later time.

Pseudo code is a term which is often used in programming and algorithm based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm.

## Reasons for using Pseudocode

1. Better readability.
2. Ease up code construction.
3. A good middle point between flowchart and code.
4. Act as a start point for documentation.
5. Easier bug detection and fixing.

The main constructs of pseudocode

The core of pseudocode is the ability to represent 6 programming constructs (always written in uppercase): SEQUENCE, CASE, WHILE, REPEAT-UNTIL, FOR, and IF-THEN-ELSE. These constructs — also called keywords — are used to describe the control flow of the algorithm.

1. SEQUENCE represents linear tasks sequentially performed one after the other.
2. WHILE a loop with a condition at its beginning.
3. REPEAT-UNTIL a loop with a condition at the bottom.
4. FOR another way of looping.
5. IF-THEN-ELSE a conditional statement changing the flow of the algorithm.
6. CASE the generalization form of IF-THEN-ELSE.

#### Rules for writing pseudocode

there are some simple rules that help make pseudocode more universally understood.

1. Always capitalize the initial word (often one of the main 6 constructs).
2. Have only one statement per line.
3. Indent to show hierarchy, improve readability, and show nested constructs.
4. Always end multiline sections using any of the END keywords (ENDIF, ENDWHILE, etc.).
5. Keep your statements programming language independent.
6. Use the naming domain of the problem, not that of the implementation. E.g., Append the last name to the first name instead of name = first+ last.
7. Keep it simple, concise, and readable.

#### Advantages of Pseudocode

1. Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.
2. Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.
3. The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

Recursion is a powerful implementation technique in which a function calls itself (either directly or indirectly) on a smaller problem of the same type in order to simplify the problem to a solvable state.

There are many different kinds of recursion, such as linear, tail, binary, nested, and mutual.

Every recursive function must have at least two cases: the recursive case and the base case.

The base case is a small problem that we know how to solve and is the case that causes the recursion to end.

The recursive case is the more general case of the problem we're trying to solve.

## Types of Recursion

1. Tail Recursion
2. Linear Recursive
3. Binary Recursive
4. Exponential recursion
5. Nested Recursion
6. Mutual Recursion

Control Structures can be considered as the building blocks of computer programs. They are commands that enable a program to take decisions, following one path or another.

The basic Control Structures in programming languages are:

Selection (Conditionals): which are used to execute one or more statements if a condition is met.

Iteration (Loops): which purpose is to repeat a statement a certain number of times or while a condition is fulfilled.

Selection is at the very core of programming. The idea behind them is that they allow the control flow of the code that is executed based on different conditions in the program

## The If Statement

If statements execute one or more statements when a condition is met. If the testing of that condition is TRUE, the statement gets executed. But if it is FALSE

## The If-Else Statement

This Control Structure allows a program to follow alternative paths of execution, whether a condition is met or not.

## While Loops

In While Loops a condition is first evaluated, and if the result of testing that condition is TRUE, one or more statements are repeatedly executed until that condition becomes FALSE.

## Decomposition

Decomposition is when we break a problem down into smaller parts to make it easier to tackle. Decomposition is a useful problem-solving strategy. It can help you write a complex computer program, plan a holiday or make a model plane.

Coupling is the measure of the degree of interdependence between the modules.

A good software will have low coupling while cohesion is a measure of the degree to which the elements of the module are functionally related.

## Modularisation

Modularisation is the process of separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

## Motivations for Modularisation

The followings are some of the major motivations

- Ease of Debugging
- Reusable Code
- Readability
- Reliability

## Basic Properties of Modularity

- Modular Decomposability
- Modular Composability
- Modular Understandability
- Modular Continuity
- Modular Protection

## Advantages of modularisation in Programming

- Manageability
- Team Programming
- Quality

## Program Testing

Fundamentally, program testing is a process to check if the program or the entire system is working same as it was supposed to do, and not working as it was not supposed to do.

## Types of Program Tests

### 1. Unit Test

Unit tests are low-level tests that focus on testing a specific part of the system. They are cheap to write and fast to run.

## 2. Integration Tests

Every complex application has internal and external components working together to do something interesting.

## 3. End-to-End Tests

End-to-end tests check to see if the system meets our defined business requirements. A common test is to trace a path through the system in the same manner a user would experience.

## 4. Functional Tests

Functional tests focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action.

## 5. Smoke Tests

Smoke tests are basic tests that check basic functionality of the application. They are meant to be quick to execute, and their goal is to give one the assurance that the major features of the system are working as expected.

## Properties of Tests

Fast \_Tests give us confidence that our code is working as intended.

Deterministic \_Tests should be deterministic.

Automated \_We can confirm our program works by running it.

## Need for Program Testing

The requirement of rigorous testing and their associated documentation during the software development life cycle arises because of the following reasons:

1. To identify defects
2. To reduce flaws in the component or system
3. Increase the overall quality of the system

The following shows the significance of testing for a reliable and easy to use software product:

The testing is important since it discovers defects/bugs before the delivery to the client, which guarantees the quality of the software.

It makes the software more reliable and easy to use.

Thoroughly tested software ensures reliable and high-performance software operation.

## Benefits of Program Testing

### Modify Code with Confidence

Identify Bugs Early  
Improve System Design

## Debugging

Debugging is a computer programming process for finding and resolving errors in software or a website, often referred to as "bugs."

### Importance of Debugging

Debugging is important because it allows software engineers and developers to fix errors in a program before releasing it to the public.

### Types of Errors to Debug

Syntax errors are grammatical interruptions in a line of code. For example, an extra bracket or period might cause a syntax error to occur.

Logic errors are issues in the code's algorithms. They can occur when a program's code produces an unexpected output or causes the program to stop working.

Run-time errors occur when a person uses the program and they're detected by the computer executing it. They can still appear after you finish an initial debugging process because a computer might interpret the program's code in an unexpected way.

Interface errors involve a disconnect in an API, which means one or both coding languages in an API cause this error to occur.

### Common Debugging Strategies

Consider the following list of eight debugging strategies in common use:

1. Run a Debugging Feature
2. Use the Scientific Method
3. Debug after Adding New Code
4. Incorporate the Backtracking Method
5. Perform the Binary Search Method
6. Classify Different Bug Types
7. Involve Static Code Analysis
8. Try Remote Debugging

### Difference between Testing and Debugging

#### Testing

Testing gets started with known conditions with expected results.

It performed based on the testing type which we need to perform unit testing, integration testing, system testing, user acceptance testing, stress, load, performance testing etc.

Testing is the process which can be planned, designed and executed.

It is the process to identify the failure of implemented code.

It is a demonstration of error or apparent correctness.

Testing is the display of errors

Design knowledge is not required for testing the system under test. Any person with or without test case can do testing.

Testing can be outsourced to outside team as well.

Most of the test cases in testing can be automated.

Testing is the process to identify the bugs in the system under test.

debugging

This is manual step by step unstructured and unreliable process to find and removes a specific bug from the system.

It performed based on the type of bug.

Debugging is the process which cannot be so forced.

It is the process to give the absolution to code failure.

It is always treated as a deductive process.

It is a deductive process.

Detailed design knowledge is definitely required to perform debugging.

Debugging cannot be outsourced to outside team. It must be done by inside development team.

Automation in the debugging cannot possible.

Debugging is the process to identify the root cause of the bugs.

GoodLuck Guys