UNIT 1

Introduction to Computational Thinking & Variables and Data Representation

(Complete Notes in Simple Hindi-English Language)

- Concept of Problem Solving & Computational Thinking
- Steps involved in Problem Solving
- Components of Computational Thinking
- Converting Problems into Flowchart & Algorithms
- Variables, Constants, Data Types
- Memory Representation of Variables

1. Problem Solving और Computational Thinking

1.1 Problem Solving क्या है? (What is Problem Solving?)

Problem Solving एक ऐसी process है जिसमें किसी problem (समस्या) को समझकर, उसका सही solution ढूंढना और उसे step-by-step तरीके से हल करना शामिल होता है।

- आसान भाषा में कहें तो, यह एक तरीका है जिससे किसी काम या समस्या को systematic (व्यवस्थित) तरीके से हल किया जाता है।
- Problem solving सिर्फ computer science में ही नहीं, बल्कि हमारी daily life में भी काम आती है — जैसे रास्ता ढूंढना, homework पूरा करना, या कोई गणित का सवाल हल करना।
- एक अच्छा problem solver पहले समस्या को अच्छी तरह समझता है, फिर उसे छोटे भागों में बांटता है और एक-एक करके हल करता है।
- Computer programming में, problem solving का इस्तेमाल किसी program या software को design करने के लिए किया जाता है।

1.2 Computational Thinking क्या है? (What is Computational Thinking?)

Computational Thinking एक ऐसी thought process (सोचने का तरीका) है, जिसकी मदद से हम किसी भी बड़ी और complex (जटिल) problem को इस तरह से analyze और solve करते हैं, जिसे एक computer भी समझ सके और execute कर सके।

- Computational Thinking में हम problem को logical steps में तोड़ते हैं ताकि उसका solution आसानी से एक algorithm या program में बदला जा सके।
- यह skill सिर्फ programmers के लिए नहीं, बल्कि हर व्यक्ति के लिए उपयोगी है जो व्यवस्थित तरीके से सोचना चाहता है।
- इसका उपयोग coding, data analysis, engineering, और यहाँ तक कि daily life की planning में भी किया जाता है।
- Computational thinking की मदद से complex problems को छोटे, समझने में आसान भागों में बदला जाता है।

Computational Thinking के मुख्य Features (Characteristics)

- Logical (तार्किक) — हर step एक logical order में होता है।
- Systematic (व्यवस्थित) — समस्या को एक क्रम (sequence) में हल किया जाता है।
- Reusable (पुनः प्रयोग योग्य) — एक बार बनाया गया solution दूसरी समान समस्याओं में भी इस्तेमाल हो सकता है।
- Efficient (कुशल) — कम समय और कम resources में समस्या हल करने पर ध्यान दिया जाता है।

1.3 Problem Solving में शामिल Steps (Steps involved in Problem Solving)

किसी भी problem को सही तरीके से हल करने के लिए हमें कुछ निश्चित steps को follow करना पड़ता है। ये steps हमें एक व्यवस्थित (organized) तरीके से solution तक पहुँचाते हैं।

चित्र 1.1 — Problem Solving के Steps

Step	विवरण (Description)
1. Analyze the Problem	सबसे पहले problem को ध्यान से पढ़ें और समझें कि वास्तव में समस्या क्या है।
2. Define the Problem	Problem के Input (क्या दिया गया है) और Output (क्या चाहिए) को स्पष्ट रूप से define करें।
3. Design the Algorithm	Problem को हल करने के लिए step-by-step तरीका (algorithm) तैयार करें।
4. Draw the Flowchart	Algorithm को visual (चित्रमय) रूप में flowchart की मदद से दिखाएं।
5. Write/Code the Program	Algorithm के आधार पर किसी programming language में code लिखें।
6. Test the Program	Program को अलग-अलग inputs के साथ चलाकर check करें कि सही output मिल रहा है या नहीं।
7. Debug	अगर program में कोई error (गलती) मिलती है, तो उसे ढूंढकर ठीक करें।
8. Document & Maintain	Program के बारे में जानकारी लिखें ताकि भविष्य में उसे समझना और सुधारना आसान हो।

1.4 Advantages of Computational Thinking

- Complex problems को आसान भागों में बांटकर हल करना आसान हो जाता है।

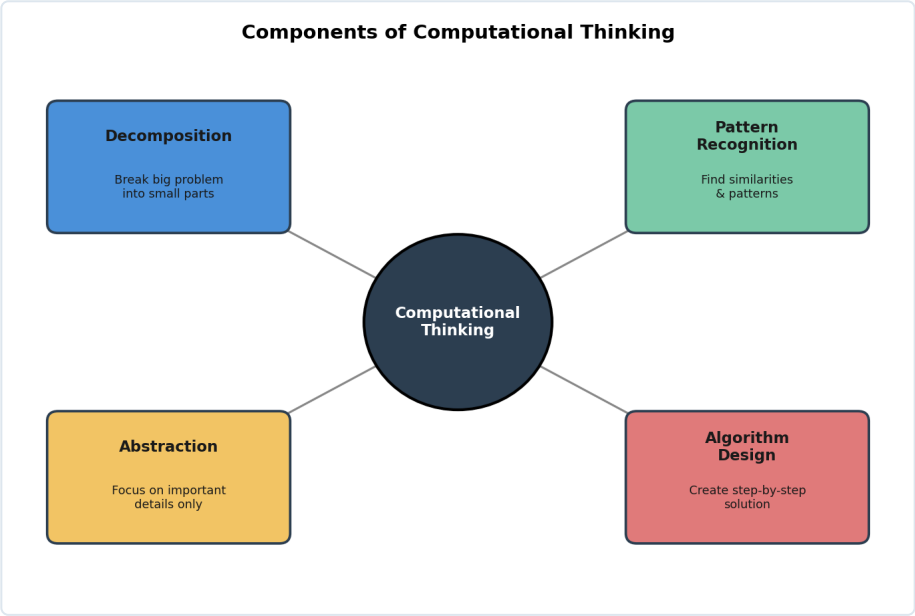
- Solution को दोबारा उपयोग (reuse) किया जा सकता है, जिससे समय की बचत होती है।
- Logical और systematic सोच विकसित होती है।
- Errors कम होते हैं क्योंकि हर step को ध्यान से plan किया जाता है।

1.5 Disadvantages / Challenges

- शुरुआत में इसे सीखना और लागू करना थोड़ा कठिन लग सकता है।
- बहुत बड़ी और complex problems के लिए ज्यादा समय और planning चाहिए होती है।
- गलत तरीके से decomposition (विभाजन) करने पर solution गलत भी हो सकता है।

2. Components of Computational Thinking

Computational Thinking चार मुख्य components (भागों) से मिलकर बनी होती है। इन चारों की मदद से किसी भी problem को आसानी से समझा और हल किया जा सकता है।



चित्र 2.1 — Computational Thinking के चार Components

2.1 Decomposition (विभाजन)

Decomposition का मतलब है किसी बड़ी और जटिल (complex) problem को छोटे-छोटे, आसान भागों (sub-problems) में तोड़ना।

- हर छोटे भाग को अलग-अलग समझना और हल करना आसान होता है।
- उदाहरण: एक पूरा project बनाने के बजाय, उसे planning, designing, coding, testing जैसे भागों में बांटा जाता है।
- Advantage: बड़ी समस्या कम डरावनी लगती है और काम को team के बीच बांटा जा सकता है।

2.2 Pattern Recognition (पैटर्न पहचान)

Pattern Recognition का मतलब है अलग-अलग problems या sub-problems के बीच समानताएं (similarities) और patterns ढूंढना।

- अगर हमें पता चल जाए कि कोई problem पहले भी solve हो चुकी है, तो हम उसी तरीके का इस्तेमाल कर सकते हैं।
- उदाहरण: कई programs में loop का इस्तेमाल बार-बार होने वाले कामों (repetitive tasks) के लिए किया जाता है।
- Advantage: समय की बचत होती है क्योंकि पुराने solutions दोबारा इस्तेमाल हो सकते हैं।

2.3 Abstraction (सार-संक्षेप)

Abstraction का मतलब है सिर्फ जरूरी (important) जानकारी पर ध्यान देना और बेकार की details को हटा देना।

- यह हमें problem के core (मुख्य) हिस्से पर focus करने में मदद करता है।

- उदाहरण: कार चलाते समय हमें सिर्फ स्टीयरिंग, ब्रेक और एक्सीलेटर के बारे में पता होना चाहिए, इंजन के अंदर की जटिल wiring के बारे में नहीं।
- Advantage: Solution को simple और समझने में आसान बनाता है।

2.4 Algorithm Design (एल्गोरिथम डिज़ाइन)

Algorithm Design का मतलब है समस्या को हल करने के लिए एक step-by-step तरीका (instructions का क्रम) तैयार करना।

- यह पूरी problem solving process का अंतिम (final) चरण है, जहाँ बाकी सभी components मिलकर एक ठोस solution बनाते हैं।
- Algorithm को बाद में program में बदला जाता है ताकि computer उसे execute कर सके।
- Advantage: एक स्पष्ट (clear) और सही क्रम में समाधान मिलता है, जिसे कोई भी follow कर सकता है।

Component	काम (Function)
Decomposition	बड़ी problem को छोटे भागों में तोड़ना
Pattern Recognition	समान patterns और similarities ढूंढना
Abstraction	सिर्फ जरूरी जानकारी पर ध्यान देना
Algorithm Design	step-by-step solution तैयार करना

3. Problems को Flowchart और Algorithm में बदलना

3.1 Algorithm क्या है? (What is an Algorithm?)

Algorithm किसी problem को हल करने के लिए दिए गए step-by-step instructions (निर्देशों) का एक निश्चित क्रम (sequence) होता है, जिसे follow करके हम एक तय समय में सही solution प्राप्त कर सकते हैं।

Algorithm की Characteristics (Features)

Feature	विवरण
Input	Algorithm में zero या उससे ज्यादा input होने चाहिए।
Output	कम से कम एक output जरूर मिलना चाहिए।
Finiteness	Algorithm एक निश्चित संख्या के steps के बाद खत्म (terminate) होना चाहिए।
Definiteness	हर step स्पष्ट (clear) और बिना किसी confusion के होना चाहिए।
Effectiveness	हर step को हाथ से (manually) भी करना संभव होना चाहिए।

उदाहरण: दो संख्याओं का योग निकालने का Algorithm

Step 1: Start
Step 2: दो संख्याओं A और B को Input लें
Step 3: Sum = A + B की calculation करें
Step 4: Sum को Print/Display करें
Step 5: Stop

Algorithm के Advantages

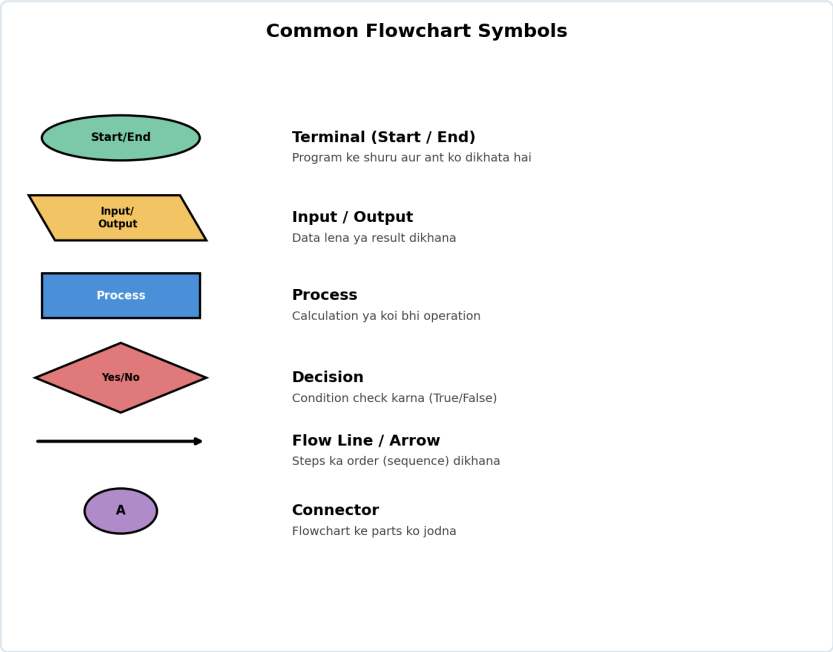
- समस्या को समझना और हल करना आसान हो जाता है।
- यह किसी भी programming language में code करने से पहले एक साफ blueprint (खाका) देता है।
- Errors ढूंढना और ठीक करना आसान हो जाता है।
- इसे बार-बार अलग-अलग जगह इस्तेमाल (reuse) किया जा सकता है।

Algorithm के Disadvantages

- बहुत बड़े और complex algorithms को समझना कठिन हो सकता है।
- Algorithm बनाने में समय लगता है।
- यह सिर्फ logic दिखाता है, actual program का output नहीं दिखाता।

3.2 Flowchart क्या है? (What is a Flowchart?)

Flowchart किसी algorithm या process को अलग-अलग symbols (जैसे oval, rectangle, diamond आदि) की मदद से visual (चित्रमय) रूप में दिखाने का तरीका है। यह किसी program के logic को समझने में बहुत मदद करता है।



चित्र 3.1 — Flowchart के Common Symbols

Flowchart के Advantages

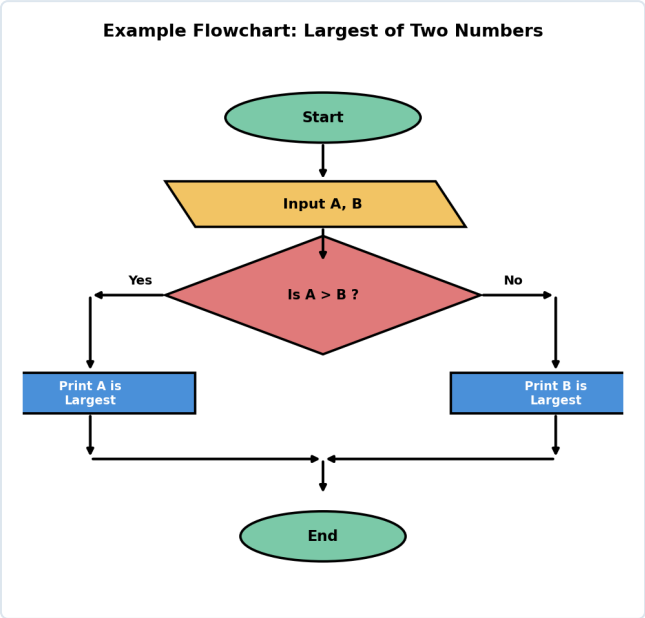
- Program के logic को समझना आसान हो जाता है, क्योंकि यह visual रूप में होता है।
- किसी भी व्यक्ति को (चाहे वो programmer हो या नहीं) समझाना आसान होता है।
- Errors को ढूंढना और debug करना आसान हो जाता है।

Flowchart के Disadvantages

- बड़े और complex programs के लिए flowchart बनाना समय लेने वाला (time-consuming) हो सकता है।
- अगर program में changes होते रहें, तो flowchart को बार-बार update करना पड़ता है।

3.3 उदाहरण: दो संख्याओं में से बड़ी संख्या ढूंढने का Algorithm और Flowchart

Step 1: Start
Step 2: दो संख्याओं A और B को Input लें
Step 3: अगर A > B है, तो A को "Largest" Print करें
Step 4: अन्यथा (Otherwise), B को "Largest" Print करें
Step 5: Stop



चित्र 3.2 — Largest of Two Numbers का Flowchart

4. Variables (चर)

Variable एक ऐसा नाम (name) होता है, जो computer की memory में किसी value (मान) को store करने के लिए इस्तेमाल किया जाता है। इस value को program के चलते समय बदला (change) जा सकता है, इसलिए इसे 'variable' कहा जाता है।

- हर variable का एक नाम, एक data type और एक value होती है।
- उदाहरण: `int age = 18;` यहाँ 'age' एक variable है जिसमें value 18 store है।
- Variable की value program के दौरान कभी भी बदली जा सकती है, जबकि constant की value fix रहती है।

4.1 Variable के नाम रखने के Rules (Naming Rules)

- Variable का नाम letter (a-z, A-Z) या underscore (_) से शुरू होना चाहिए, number से नहीं।
- Variable के नाम में space (खाली जगह) नहीं होनी चाहिए।
- Special characters (जैसे @, #, %) का इस्तेमाल नहीं किया जा सकता।
- Variable का नाम meaningful (अर्थपूर्ण) होना चाहिए, जैसे 'age', 'totalMarks' आदि।
- Programming language के reserved keywords (जैसे int, for, if) को variable नाम के तौर पर इस्तेमाल नहीं किया जा सकता।

4.2 Types of Variables

Type	विवरण
Local Variable	यह किसी function या block के अंदर define होता है और सिर्फ उसी block में इस्तेमाल किया जा सकता है।
Global Variable	यह पूरे program में कहीं भी define किया जा सकता है और पूरे program में इस्तेमाल किया जा सकता है।

5. Constants (स्थिरांक)

Constant एक ऐसी value होती है, जो program के चलते समय (execution के दौरान) कभी नहीं बदलती। इसे एक बार define करने के बाद उसकी value fixed रहती है।

- उदाहरण: गणित में `PI = 3.14` हमेशा एक ही रहता है, इसे बदला नहीं जा सकता।
- Constants का इस्तेमाल तब किया जाता है जब हमें पता हो कि कोई value कभी नहीं बदलेगी।

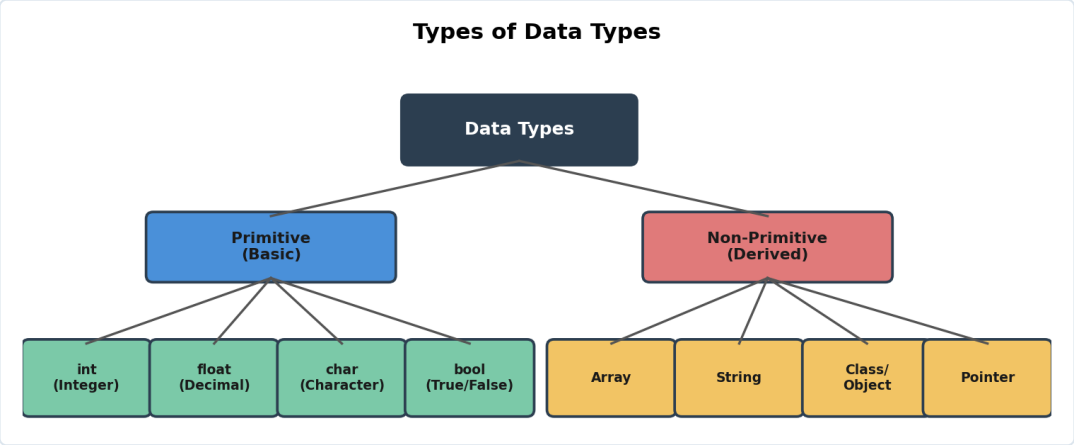
5.1 Constants के Types

Type	उदाहरण (Example)
Numeric Constant	10, 3.14, -25

Type	उदाहरण (Example)
Character Constant	'A', 'z', '5'
String Constant	"Hello World", "India"
Boolean Constant	True, False

6. Data Types (डाटा प्रकार)

Data Type यह बताता है कि किसी variable में किस प्रकार की value (जैसे number, text, decimal आदि) store की जा सकती है। हर programming language में कुछ built-in data types होते हैं।



चित्र 6.1 — Data Types के प्रकार

6.1 Primitive (Basic) Data Types

Data Type	विवरण और उदाहरण
int (Integer)	बिना दशमलव (decimal) की पूर्ण संख्याएं (whole numbers) store करता है। उदाहरण: 10, -5, 200
float (Decimal)	दशमलव (decimal point) वाली संख्याएं store करता है। उदाहरण: 3.14, -0.5, 99.99
char (Character)	एक single character store करता है। उदाहरण: 'A', 'z', '9'
boolean	सिर्फ दो values store करता है: True या False

6.2 Non-Primitive (Derived) Data Types

Data Type	विवरण
Array	एक ही data type की कई values को एक साथ store करने के लिए इस्तेमाल होता है।
String	characters का एक समूह (sequence) होता है, जैसे नाम या वाक्य।
Class/Object	कई अलग-अलग data types और functions को एक साथ जोड़कर बनाया गया एक structure।
Pointer	किसी variable के memory address को store करता है।

6.3 Data Types के Features

- हर data type memory में अलग-अलग जगह (size) लेता है।
- Data type यह decide करता है कि variable पर कौन-कौन से operations किए जा सकते हैं।
- सही data type का चुनाव program को fast और efficient बनाता है।

6.4 सामान्य Data Types का Memory Size (उदाहरण के तौर पर)

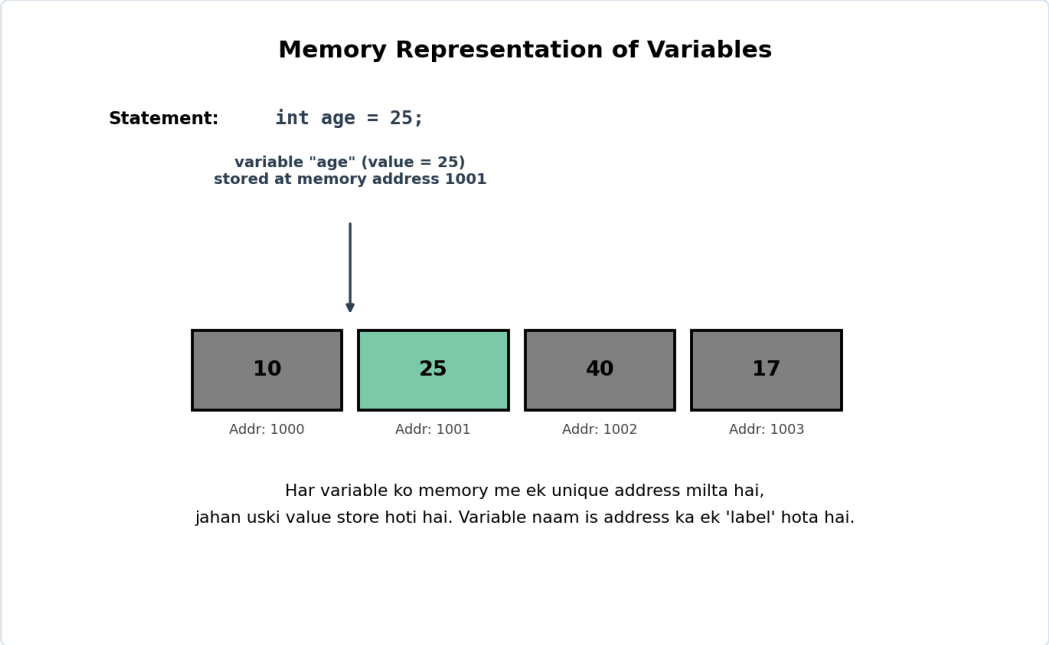
Data Type	Memory Size (लगभग)
int	4 bytes
float	4 bytes
char	1 byte
boolean	1 byte
double	8 bytes

Note: यह size अलग-अलग programming language और system के अनुसार थोड़ा अलग हो सकता है।

7. Memory Representation of Variables

जब भी हम किसी program में कोई variable बनाते (declare करते) हैं, तो computer उस variable के लिए memory (RAM) में एक जगह (space) allot करता है। इस जगह का एक unique address होता है, जहाँ variable की value store होती है।

- हर variable को memory में एक address मिलता है, जैसे 1000, 1001 आदि।
- Variable का नाम असल में उस memory address का एक आसान (human-readable) 'label' या 'नाम' होता है।
- जब हम variable का इस्तेमाल करते हैं, तो computer उस नाम की मदद से सही memory address ढूंढकर value को access करता है।
- अलग-अलग data types अलग-अलग मात्रा में memory (bytes) लेते हैं — जैसे int को 4 bytes और char को 1 byte चाहिए होता है।



चित्र 7.1 — Variable की Memory में Representation

7.1 Memory Representation क्यों जरूरी है?

- इससे हमें समझ आता है कि program के अंदर data कैसे store और access होता है।
- यह concept आगे चलकर pointers, arrays और references जैसे advanced topics समझने में मदद करता है।
- सही memory management से program तेज़ (fast) और efficient बनता है।

याद रखने वाली बातें (Key Points to Remember):

- Problem Solving एक व्यवस्थित (systematic) process है जो analyze से लेकर maintain करने तक चलती है।
- Computational Thinking के चार components हैं: Decomposition, Pattern Recognition, Abstraction, और Algorithm Design।
- Algorithm एक step-by-step solution है, जबकि Flowchart उसी solution को visual (चित्रमय) रूप में दिखाता है।

- Variable की value बदल सकती है, जबकि Constant की value हमेशा fixed रहती है।
- Data Type यह तय करता है कि variable में किस तरह की value store होगी और वह memory में कितनी जगह लेगी।