Problem Solving Techniques

PALLAVI VAIDYA

Agenda

- Why Problem solving matters?
- Problem solving process
- Python problems

“Programming isn’t about typing — it’s about thinking.”

MASTERING PROBLEM-SOLVING GIVES YOU THE ABILITY TO TACKLE **ANY PROGRAMMING CHALLENGE**, REGARDLESS OF THE LANGUAGE OR TOOL.

Why Problem-Solving Matters

- Builds logical and analytical thinking
- Improves debugging and optimization skills
- Helps write efficient, maintainable code
- Prepares you for coding interviews and real-world challenges

The Problem-Solving Process

1. Understand the Problem
2. Plan the Solution
3. Write Pseudocode
4. Implement the Code
5. Test and Debug
6. Optimize the Solution

Step 1: Understand the Problem

- Read the question carefully
- Identify inputs, outputs, and constraints
- Clarify any ambiguities

Step 2: Plan the Solution

- Break the problem into smaller steps
- Think of multiple approaches
- Choose the most efficient one (time/space complexity)

Step 3: Write Pseudocode

Example:

```
1. Start
2. Get two numbers
3. Add them
4. Display the sum
```

Example Problem

Find the sum of all even numbers between 1 and n.

Pseudocode:

```
SET sum = 0
FOR i FROM 1 TO n
    IF i is even THEN sum = sum + i
PRINT sum
```

Python Implementation

```
n = int(input('Enter n: '))

total = 0

for i in range(1, n+1):

    if i % 2 == 0:

        total += i

print('Sum of even numbers:', total)
```

Key Problem-Solving Techniques

- Decomposition – Break problem into smaller parts
- Pattern Recognition – Identify recurring structures
- Abstraction – Focus on essential details
- Algorithm Design – Create step-by-step logical plan
- Testing & Debugging – Ensure correctness

Types of Problem

Arithmetic and Mathematical Problems

String Manipulation Problems

List, Tuple, and Collection Problems

Conditional and Logical Problems

Looping and Iteration Problems

Function and Recursion Problems

File Handling Problems

Object-Oriented Programming (OOP) Problems

Algorithm and Data Structure Problems

Error Handling and Exception Problems

Arithmetic and Mathematical Problems

Involve calculations, formulae, or numerical patterns.

Concepts: operators, loops, functions, math module.

Example:

Sum of first n natural numbers

Factorial of a number

Prime number check

String and Text Processing

Work with characters, words, or sentences.

Concepts: string methods, slicing, formatting, regex.

Example:

Count vowels in a string

Reverse a string

Palindrome check

Data Structure Problems

Use lists, tuples, sets, dictionaries for storage and manipulation.

Concepts: indexing, iteration, insertion, deletion, sorting.

Example:

Find the largest element in a list

Merge two dictionaries

Remove duplicates from a list

Logical and Conditional Problems

Involve decision-making with if-else statements and boolean logic.

Concepts: comparison operators, logical operators, nested conditions.

Example:

Grade calculator

Leap year checker

Voting eligibility

Looping and Iteration Problems

Use for and while loops to repeat actions.

Concepts: counters, accumulators, nested loops.

Example:

Multiplication table

Fibonacci sequence

Pattern printing

Algorithmic Problems

Require designing a step-by-step approach to solve efficiently.

Concepts: sorting, searching, recursion, optimization.

Example:

Binary search

Bubble sort

Tower of Hanoi (recursive problem)

File Handling and I/O Problems

Read/write data from/to files.

Concepts: open(), read(), write(), context managers.

Example:

Count words in a file

Append new data to a file

CSV processing

Object-Oriented Programming (OOP) Problems

Use classes, objects, inheritance, and methods.

Concepts: encapsulation, abstraction, polymorphism.

Example:

Bank account class with deposit/withdraw methods

Employee class with salary calculation

Exception Handling Problems

Focus on handling runtime errors gracefully.

Concepts: try-except, finally, raising exceptions.

Example:

Division by zero handling

File not found error handling