Building Logic In Python

By
Harshit Kumar Singh
harshitsingh097@gmail.com

CODE_U_BABY

WhatsApp Channel

Chart Beast Portal

Telegram Channel

Challenge: Not Knowing the Formula

Scenario:

Suppose you encounter a problem asking for the **sum of the interior angles of a polygon**, but you're unaware of the formula.

Step-by-Step Solution:

1️⃣ Understand the Problem:
The sum of the interior angles of a polygon depends on the number of sides n.

Formula:

$$\text{Sum of Interior Angles} = (n-2) \times 180$$

2️⃣ Break it Down:

- Each polygon can be divided into triangles.
- A triangle has a sum of 180°.
- For an n-sided polygon, you can divide it into n−2 triangles.

3️⃣ Implement in Python:
Even if you don't know the formula initially, deriving it logically allows you to write the solution in Python.

Python Code:

```
def sum_of_interior_angles(sides):
    if sides < 3:  # A polygon must have at least 3 sides
        return "Invalid polygon"
    return (sides - 2) * 180
```

```
# Examples
sides_1 = 4  # Quadrilateral
sides_2 = 6  # Hexagon
sides_3 = 2  # Invalid case
```

```
print(f"Sum of interior angles (4 sides): {sum_of_interior_angles(sides_1)}°")
print(f"Sum of interior angles (6 sides): {sum_of_interior_angles(sides_2)}°")
print(f"Sum of interior angles (2 sides): {sum_of_interior_angles(sides_3)}")
```

Output:

Sum of interior angles (4 sides): 360°

Sum of interior angles (6 sides): 720°

Sum of interior angles (2 sides): Invalid polygon

Tips to Overcome Lack of Knowledge:

1. **Research and Learn**: If you encounter an unfamiliar problem, research the formula or theory. Google, documentation, or books can help.
2. **Start with Simpler Examples**: For polygons, test with triangles and quadrilaterals to identify patterns.
3. **Build a Logical Derivation**: Even if you don't know the formula, break the problem into smaller, understandable steps to derive it.
4. **Keep Notes**: Maintain a personal cheat sheet for formulas and concepts to refer to later

Challenge:
The Thought Process or Approach in Solving Problems
“ The Dice Problem”

Problem Statement:
You are given a standard six-sided dice, where the sum of opposite faces is always 7. If one face of the dice is shown, you need to determine the number on the opposite face.

Thought Process and Approach

To solve this problem effectively, one needs to apply both **logical reasoning** and structured programming skills. Let's break it down:

Step 1: Understand the Rules of the Problem

- For a standard six-sided dice:
    - Opposite faces always sum up to **7**.
    - The pairs of opposite faces are:
        - 1↔6----------1↔6
        - 2↔5----------2↔5
        - 3↔4----------3↔4

Step 2: Design an Efficient Solution

Approaches to solve this problem:

Approach 1: Direct Calculation

- If one face x is given, the opposite face is 7−x.
- This approach is simple and efficient, as it works in O(1).

Python Code:

```
def opposite_face(face):
        if face < 1 or face > 6:
                return "Invalid face of a dice"
        return 7 – face

# Example
given_face = 3
print(f"The opposite face of {given_face} is opposite_face(given_face)}.")
```

Output:

The opposite face of 3 is 4.

Approach 2: Use a Dictionary for Predefined Pairs

- Store the pairs of opposite faces in a dictionary.
- This approach allows for extension to non-standard dice with different rules.

Python Code:

```
def opposite_face_with_dict(face):
    opposite_faces = {1: 6, 2: 5, 3: 4, 4: 3, 5: 2, 6: 1}
    return opposite_faces.get(face, "Invalid face of a dice")

# Example
given_face = 2
print(f"The opposite face of {given_face} is
{opposite_face_with_dict(given_face)}.")
```

Output:

The opposite face of 2 is 5.

Think Like a Software Developer

1. **Start with Simplicity**:
    - For a well-defined rule like 7−x, a direct calculation is the most efficient approach.
2. **Consider Extensibility**:
    - If the dice has different rules (e.g., custom sides or labels), using a dictionary or mapping is more flexible.
3. **Validate Inputs**:
    - Handle edge cases, like invalid inputs (e.g., x>6x > 6x>6 or x<1x < 1x<1).
4. **Test Your Solution**:
    - Test with all possible valid inputs (1–6) and some invalid inputs (e.g., 0, 7).

Competitive Programming Perspective

For competitive programming, developing the right thought process is critical:

1. **Understand the Constraints**:
    - Analyze the problem to identify the simplest rules or patterns.
    - Optimize for speed and space if constraints are tight.
2. **Break Down the Problem**:
    - Divide the problem into logical steps (e.g., identifying the rule, writing the logic, handling exceptions).
3. **Choose Efficient Data Structures**:
    - Use dictionaries, lists, or arrays only if they add significant value. Avoid overcomplicating simple problems.
4. **Write Clean, Readable Code**:
    - Competitive programming often values concise solutions, but clarity is equally important during interviews.
5. **Practice Regularly**:
    - Problems like this help build foundational skills in pattern recognition and efficient coding.

Mindset as a Developer

- **Curiosity**: Always ask "why" and "how" a rule or formula works. For example, why do opposite faces of dice sum to 7? This understanding helps solve variants of problems.
- **Iterative Improvement**: Start with a brute-force approach if needed and refine it to a more optimal solution.
- **Adaptability**: Learn to generalize solutions (e.g., handling custom dice in this case).

THANK YOU

By
Harshit Kumar Singh
harshitsingh097@gmail.com

CODE_U_BABY

WhatsApp Channel

Chart Beast Portal

Telegram Channel