Sum of N Numbers

Problem Description

You are given an integer **N**, followed by **N integers**. Your task is to compute and print the **sum of these N numbers**.

The input guarantees that all numbers are valid integers and fit within standard integer limits.

Example Explanation

Input

```
5
1 2 3 4 5
```

Explanation

- N = 5, meaning there are 5 numbers to read
- The numbers are: 1, 2, 3, 4, and 5

Adding them together:

```
1 + 2 + 3 + 4 + 5 = 15
```

Output

```
15
```

Constraints & Key Observations

- `1 <= N <= 100`
- The number of elements is small
- A single pass through the numbers is sufficient
- No sorting or advanced data structures are required
- Integer overflow is not a concern under given constraints

These constraints indicate that a **simple linear approach** is optimal.

Intuition

To find the total sum, we only need to **keep adding numbers as we read them**.

There is no dependency between elements, and each number contributes independently to the final result.

This makes the problem ideal for a **single-pass accumulation strategy**.

Approaches

Approach 1: Brute Force

Explanation We read all N numbers and add them one by one using a loop.

Each number is added to a running total until all elements are processed.

Why It Works Addition is associative, and every element must be included exactly once in the sum. Since we explicitly add every number, the result is always correct.

Why It Fails This approach does **not fail** under the given constraints. However, if N were extremely large (e.g., or more), input reading itself could become a bottleneck.

Code

```
n = int(input().strip())
numbers = list(map(int, input().split()))

total = 0
for num in numbers:
    total += num

print(total)
```

Time Complexity

- **O(N)** — each number is processed once

Space Complexity

- **O(N)** — storing the list of numbers

Approach 2: Optimized

Explanation Instead of manually iterating, we use Python's built-in `sum()` function, which is optimized and concise.

Why It Works `sum()` internally iterates over the elements and accumulates their values efficiently.

The logic remains the same, but the implementation is cleaner and less error-prone.

Code

```
n = int(input().strip())
numbers = list(map(int, input().split()))

print(sum(numbers))
```

Time Complexity

- **O(N)** — `sum()` processes each element once

Space Complexity

- **O(N)** — input list storage

Edge Cases & Common Pitfalls

- Forgetting to read **N** before reading the numbers
- Input numbers spread across multiple lines (always read carefully)
- Printing extra spaces or text
- Assuming numbers fit in smaller data types (safe here, but dangerous in general)

When Not to Use This Approach

- When numbers arrive as a **continuous stream** and cannot be stored
- When working in **memory-constrained environments**
- When overflow-prone integer types are involved (not applicable here)

‘‘‘