

Logic to Find the Second Largest Number:

1. Check the list length:

- Make sure the list has at least two distinct numbers.

2. Initialize two variables:

- One for the **largest** number.
- One for the **second largest** number.

3. Go through each number in the list:

- If the number is **greater than the largest**, update:
 - `second largest = largest`
 - `largest = current number`
- Else if the number is **not equal to the largest** and **greater than the second largest**, update:
 - `second largest = current number`

4. At the end, the second largest variable will contain the result.

2. A function needs to convert an integer to its binary representation without using Python's built-in `bin()` function.

Write logic to convert a given integer to its binary representation.

Step-by-step Logic:

1. Check if the number is 0:

- If yes, return `"0"` (since the binary of 0 is 0).

2. Initialize an empty string or list to store binary digits.

3. Repeat the following steps while the number is greater than 0:

- Divide the number by 2.
- Store the **remainder** (either 0 or 1).

- Update the number as the **quotient** from the division.
 - 4. **The binary digits will be in reverse order**, so reverse them at the end.
 - 5. **Return the reversed sequence** as the binary representation.
-

✓ Example:

To convert 10 to binary:

- $10 \div 2 = 5$, remainder = 0
- $5 \div 2 = 2$, remainder = 1
- $2 \div 2 = 1$, remainder = 0
- $1 \div 2 = 0$, remainder = 1

Now, reverse the remainders: **1 0 1 0** → **1010**

Binary of 10 is **1010**

3.A function needs to merge two sorted lists into a single sorted list efficiently. Write logic to merge two sorted lists into one sorted list.

Logic to Merge Two Sorted Lists:

1. Initialize two pointers, one for each list (let's call them **i** and **j**, both start at 0).
2. Create an empty list to hold the merged result.
3. While both pointers are within the bounds of their respective lists:
 - Compare the current elements of both lists.
 - Append the smaller element to the result list.

- Increment the pointer of the list from which the element was taken.
4. After one list is exhausted, append all remaining elements from the other list to the result list.
 5. The result list will be a single merged sorted list.
4. A function needs to find the first non-repeating character in a string for text processing. Write logic to find the first non-repeating character in a given string.


Step-by-step Logic:

1. Initialize a frequency counter:
 - Go through each character in the string.
 - Count how many times each character appears (e.g., using a dictionary or map).
2. Scan the string again from the beginning:
 - For each character, check its count from the frequency map.
 - The first character with a count of 1 is the first non-repeating character.
3. If no such character is found, return a message like "No non-repeating character".

Example:

For the string "aabbcd~~eff~~":

- Frequencies:

- a: 2
 - b: 2
 - c: 1 
 - d: 1
 - e: 1
 - f: 2
- First non-repeating character is: **c**

5. A program needs to identify common elements between two lists for data filtering. Write logic to find the common elements between two lists.

Step-by-step Logic:

1. Create a data structure (like a set) to store the elements of the first list.
 - Sets are efficient for membership checks ($O(1)$ time).
2. Initialize an empty list (or set) to store the common elements.
3. Go through each element in the second list:
 - Check if the element exists in the set created from the first list.
 - If it does, add it to the result list (or set).
4. (Optional): If you want only unique common elements, use a set for the result.

 **Example:**

List A: [1, 2, 3, 4, 5]

List B: [4, 5, 6, 7]

Common elements: [4, 5]

6. A function is required to reverse a given number.

Write logic to reverse a given number.

Step-by-step Logic:

1. Initialize a variable to store the reversed number (e.g., `rev = 0`).
2. Repeat the following steps until the number becomes 0:
 - Extract the last digit using modulo: `digit = number % 10`.
 - Add the digit to the reversed number: `rev = rev * 10 + digit`.
 - Remove the last digit from the original number: `number = number // 10`.
3. At the end, the `rev` variable holds the reversed number.

✓ Example:

For number 1234:

- Step 1: `digit = 4` → `rev = 0 * 10 + 4 = 4`
- Step 2: `digit = 3` → `rev = 4 * 10 + 3 = 43`
- Step 3: `digit = 2` → `rev = 43 * 10 + 2 = 432`
- Step 4: `digit = 1` → `rev = 432 * 10 + 1 = 4321`

✓ Reversed number = 4321

7. A program needs to count the number of words in a given sentence. Write logic to count the number of words in a given sentence.

Step-by-step Logic:

1. Start with a sentence (string) input.
 2. Trim extra spaces at the beginning and end (to avoid counting empty words).
 3. Split the sentence using spaces as separators:
 - Words in a sentence are usually separated by one or more spaces.
 - Splitting the sentence based on spaces gives you a list of words.
 4. Count the number of elements in the list – that gives the total word count.
-

✓ Example:

Sentence: " Hello world, this is Python! "

- After trimming: "Hello world, this is Python!"
- Splitting gives: ["Hello", "world,", "this", "is", "Python!"]
- Word count = 5

8. A function needs to compute the factorial of a number using iteration instead of recursion. Write logic to find the factorial of a given number using iteration.

Step-by-step Iterative Logic:

1. Check if the number is 0 or 1:
 - The factorial of 0 and 1 is 1.
 2. Initialize a result variable as 1 (e.g., `result = 1`).
 3. Use a loop from 2 to the given number (inclusive):
 - Multiply the result with the current number in the loop.
 - Update the result each time.
 4. After the loop ends, the result variable will contain the factorial.
-

✓ Example:

For `n = 5`:

- `result = 1`
- `result = 1 × 2 = 2`
- `result = 2 × 3 = 6`
- `result = 6 × 4 = 24`
- `result = 24 × 5 = 120`

✓ So, `5! = 120`

9. A program is required to convert all strings in a list to uppercase. Write logic to convert all strings in a list to uppercase.

Step-by-step Logic:

1. Start with a list containing strings.
 2. Initialize an empty list to store the uppercase results.
 3. Loop through each string in the original list:
 - For each string, convert it to uppercase using a string operation.
 - Add the uppercase string to the new list.
 4. After the loop, the new list will contain all strings in uppercase.
-

Example:

Original list: ["apple", "banana", "Cherry"]

After conversion: ["APPLE", "BANANA", "CHERRY"]

10. A function is needed to compute the greatest common divisor (GCD) of two numbers using the Euclidean algorithm.

Write logic to calculate the GCD of two numbers using the Euclidean algorithm.

Euclidean Algorithm Logic:

1. Start with two positive integers, say **a** and **b**.
 2. Repeat the following steps until **b** becomes 0:
 - Replace **a** with **b**.
 - Replace **b** with **a % b** (the remainder when **a** is divided by **b**).
 3. When **b** becomes 0, the value of **a** is the GCD.
-

✓ Example:

Find GCD of 48 and 18:

- Step 1: $a = 48, b = 18$
- Step 2: $a = 18, b = 48 \% 18 = 12$
- Step 3: $a = 12, b = 18 \% 12 = 6$
- Step 4: $a = 6, b = 12 \% 6 = 0$
- Now $b = 0$, so GCD is 6