1-What is the purpose of the finally block?

- The **finally** block in C# contains code that **always executes** after a try/catch, regardless of whether an exception was thrown or caught.
  Its main purpose is **cleanup** — e.g., closing files, releasing resources — to ensure these actions happen no matter what.

2-How does int.TryParse() improve program robustness compared to int.Parse()?

- int.TryParse() is safer than int.Parse() because it avoids exceptions on invalid input, returning **true/false** instead, which lets you handle errors gracefully and keep the program running.

3-What exception occurs when trying to access Value on a null Nullable?

- Accessing .Value on a null Nullable<T> throws an **InvalidOperationException** saying *"Nullable object must have a value".*

4-Why is it necessary to check array bounds before accessing elements?

- It's necessary to check array bounds before accessing elements to prevent **IndexOutOfRangeException**, which occurs if you try to access an index less than 0 or greater than or equal to the array's length.

5-How is the GetLength(dimension) method used in multi-dimensional arrays?

int[,] matrix = new int[3, 4]; // 3 rows, 4 columns

Console.WriteLine(matrix.GetLength(0)); // Output: 3 (rows)

Console.WriteLine(matrix.GetLength(1)); // Output: 4 (columns)

6-How does the memory allocation differ between jagged arrays and rectangular arrays?

- **Rectangular arrays** → One continuous memory block, fixed row size, faster access, better cache use.
  Jagged arrays → Array of references to separate arrays, variable row sizes, more flexible but slightly slower.

7-What is the purpose of nullable reference types in C#?

- Nullable reference types make nullability explicit in C#, helping the compiler warn you about possible null reference errors and reducing null-related bugs.

8-What is the performance impact of boxing and unboxing in C#?

- Boxing and unboxing are slow compared to direct value-type use because they involve extra memory allocation (boxing) and type conversion (unboxing), which add CPU and garbage collection overhead.

9-Why must out parameters be initialized inside the method?

- out parameters must be initialized inside the method to ensure the caller always receives a definite, valid value.

10-Why must optional parameters always appear at the end of a method's parameter list?

- Optional parameters must be last so that all required arguments can be supplied in order without ambiguity, ensuring the compiler knows which values match which parameters.

11-How does the null propagation operator prevent NullReferenceException?

- The null propagation operator (?.) stops evaluation if the left side is null, returning null instead of accessing members and causing a NullReferenceException.

12-When is a switch expression preferred over a traditional if statement?

- A switch expression is preferred when you have multiple fixed conditions based on a single value and want more concise, readable, and expression-based code.

13-What are the limitations of the params keyword in method definitions?

- params can only be used for one parameter, it must be the last in the method’s parameter list, and it must be a single-dimensional array type.