✅ **1. Difference between int.Parse and Convert.ToInt32 with null inputs**

- int.Parse(null) → **throws ArgumentNullException**
- Convert.ToInt32(null) → **returns 0** (handles null safely)

🔷 **Use Convert.ToInt32** if you expect the input might be null.

✅ **2. Why is TryParse recommended over Parse in user-facing apps?**

- int.Parse throws exceptions on invalid input (e.g., text or empty string)
- int.TryParse **returns false instead of crashing**, and gives a safe way to validate input

🔷 TryParse makes your app more **robust, user-friendly, and exception-free**.

✅ **3. Purpose of GetHashCode()**

- It returns an **integer hash code** used in hash-based collections like Dictionary, HashSet, or Hashtable.
- It **represents the object's content (ideally)** for quick lookups.

🔷 Two objects with the same data may have the same hash code, but it's **not guaranteed to be unique**.

✅ **4. What is reference equality in .NET?**

- It checks whether **two variables refer to the same object in memory**, not just the same value.
- Use ReferenceEquals(obj1, obj2) or object.ReferenceEquals(...) to test it.

🔷 Important for **comparing objects, especially mutable ones or when memory identity matters**.

✅ **5. Why is string immutable in C#?**

- Once created, a string **cannot be changed**.
- Benefits:
  - **Thread safety**
  - **Consistent behavior**
  - **Better memory management** (via string interning and reuse)

🔷 Any operation that seems to "modify" a string actually creates a **new string**.

✅ **6. How does StringBuilder solve string concatenation inefficiency?**

- StringBuilder allows **modifying the same memory** without creating new strings.
- Efficient for **repeated append/replace/insert operations**.

🔷 Ideal when you're doing **lots of string changes in loops or large data**.

✅ **7. Which string formatting method is most used and why?**

String interpolation ($"") is the most popular because:

- It's **cleaner, readable**, and **less error-prone**:

$"Sum is {a} + {b} = {a + b}"

- Easier than string.Format or messy concatenation (+)

🔷 Preferred in modern C# (especially from C# 6.0 onwards).

✅ **8. How is StringBuilder designed for frequent string modifications?**

- Internally uses a **mutable character buffer**
- Expands its memory **dynamically**, avoiding reallocation like string does
- Efficient for:
    - Appending
    - Replacing
    - Removing
    - Inserting

🔷 It's **optimized for performance** when you expect multiple string changes.