✅ 2. What is an enum data type, when is it used?

🔷 Definition:

An enum (short for enumeration) is a user-defined value type that consists of a set of named constants (like labels for numbers).

```
enum Days { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday }
```

🔷 When is it used?

- To represent a fixed set of related values (like states, options, categories).
- To make code easier to read and less error-prone (no magic numbers or strings).
- To group constants meaningfully.

🔷 Example:

```
enum Status { Pending, Approved, Rejected }
```

```
Status orderStatus = Status.Approved;
```

🔷 Three Common Built-in Enums:

1. ConsoleColor – used to set foreground/background text color in the console.

```
Console.ForegroundColor = ConsoleColor.Green;
```

2. DayOfWeek – returns the day of the week (used in DateTime).

```
DayOfWeek today = DateTime.Now.DayOfWeek;
```

3. StringComparison – used when comparing strings with different rules (case-sensitive, culture-aware, etc.).

```
string.Equals("hello", "HELLO", StringComparison.OrdinalIgnoreCase);
```

✅ 3. When to use string vs StringBuilder

Scenario	Use string	Use StringBuilder
Small or single modifications	✅ Better	❌ Not needed
Readable and simple operations	✅ Cleaner (e.g., interpolation)	❌ Overkill
Many changes in a loop	❌ Inefficient (creates many strings)	✅ Best choice (high performance)
Performance-sensitive scenarios	❌ Causes memory churn	✅ Optimized for performance
Example	"Hello " + name + "!"	sb.Append("Hello ").Append(name).Append("!")

🔷 Summary:

- Use string for simple or occasional concatenation.
- Use StringBuilder when building or editing strings multiple times, especially inside loops.

5. What is a user-defined constructor and its role in initialization?

🔷 **Definition:**

A **user-defined constructor** is a special method in a class that is **explicitly written by the programmer** to initialize objects with custom values.

```
class Person
{
  public string Name;
  public int Age;

  // User-defined constructor
  public Person(string name, int age)
  {
    Name = name;
    Age = age;
  }
}
```

🔷 **Role in Initialization:**

- Allows **setting values at the time of object creation**.
- Can **enforce required values** (no uninitialized fields).
- Can define **multiple constructors (overloading)** for flexible object creation.

🔷 **Example Usage:**

```
Person p = new Person("Ahmed", 22);
Console.WriteLine(p.Name); // Outputs: Ahmed
```

🔷 Without a user-defined constructor, you'd have to assign values manually after creation, or rely on default values.

✅ 6. Compare between Array and Linked List

Feature	Array	Linked List
Memory Allocation	Contiguous memory block	Nodes stored in scattered memory
Size	Fixed-size (unless using List<T>)	Dynamic size (can grow/shrink easily)
Access (Indexing)	Fast (O(1)) via index	Slow (O(n)) — must traverse from head
Insertion/Deletion	Costly (need shifting)	Efficient (O(1) at head/tail if pointer known)
Cache Friendliness	High (due to contiguous memory)	Low (nodes may not be in order in memory)
Implementation	Built-in (int[], string[])	Requires creating custom Node and List classes
Use Case	Use when size is known, fast access needed	Use when frequent insert/delete is needed

---

🔷 **Summary:**

- **Use arrays** when:
    - You need **fast access** by index.
    - Size is **known and fixed** or mostly stable.
- **Use linked lists** when:
    - Frequent **insertions/deletions**.
    - You don't know the size in advance.