TASK 1 LINQ

Link of Article

https://www.linkedin.com/pulse/deferred-execution-vs-eager-linq-what-business-leaders-mohamed-afify-abvkf/

self study:

1 Yielding

- Meaning in C#: Using the keyword yield return.
- Purpose: Instead of returning a full collection at once, it returns items *one by one* when the consumer asks for them.
- Relation to Deferred Execution: With yield, the sequence is not pre-built — each element is generated on demand.

Example:

```
static IEnumerable<int> GetNumbers()
{
    Console.WriteLine("Start generating...");
    for (int i = 1; i <= 5; i++)
    {
        Console.WriteLine($"Yielding {i}");
        yield return i;
    }
}

foreach (var num in GetNumbers())
{
    Console.WriteLine($"Consumed {num}");
}
```

Output shows that items are generated only when consumed, not upfront.

2 Lazy Loading

- **Meaning:** Delaying the loading of an object/resource until it's actually needed.
- **Common usage:** In ORMs like Entity Framework.
    - Example: Customer.Orders are not loaded from the database until you actually access Orders.
- **Relation to Deferred Execution:** Similar idea: *don't fetch or compute until someone requests it*.

Example:

```
public class Customer
{
    private List<Order> _orders;
    public List<Order> Orders
    {
        get
        {
            if (_orders == null)
            {
                Console.WriteLine("Loading orders from DB...");
                _orders = LoadOrdersFromDatabase();
            }
            return _orders;
        }
    }
}
```

Here, the orders are only loaded when the property is accessed for the first time.

Difference between them

- **Yielding:** Language feature in C# for iterators (delays item generation).
- **Lazy Loading:** Design pattern/architecture concept (delays resource or data fetching).

built-in functions in collections like List<T>

You don't need to implement them manually, because they are part of the class by default.

For example, List<T> has ready-to-use methods such as:

- Add()
- Remove()
- Insert()
- Clear()
- Contains()
- TrimExcess()

Example with a normal array (no self functions):

```
int[] arr = new int[5];

// If you want to add a new element, you must write manual logic

// like resizing the array and copying elements.
```

Example with a List (self functions):

```
List<int> list = new List<int>();

list.Add(10);     // built-in → self

list.Remove(10);  // built-in → self
```

dynamic:

- Type is resolved at **runtime** (self-handled at runtime).
- The variable can change its type during execution.

```
dynamic data = "Ali";  // runtime: string

data = 123;          // runtime: int

data = true;         // runtime: bool
```

Why the comment says *"Web >> Self"*?

- dynamic is **commonly used in Web programming** (e.g., APIs, JSON parsing, reflection) because often you don't know the type until **runtime**.
- The note "Self" means:
  the variable decides its type by itself at runtime (not by the compiler at compile-time).