LINQ

✅ **Why LINQ works without creating any object?**

Because:

👉 **LINQ is not a class. LINQ is a set of *extension methods***

👉 When you write using System.Linq;, you add methods like OrderBy, Where, Select to all types that implement **IEnumerable<T>**.

This is why you can call:

languages.OrderBy(x => x)

even though languages is just:

List<string>

You *never* create a "LINQ object".

✅ **So how does the query actually work?**

This LINQ query:

```
var sortedList = from lang in languages
             orderby lang
             select lang;
```

Is silently transformed by the compiler into:

```
var sortedList = languages.OrderBy(lang => lang);
```

And that works because:

✓ languages implements IEnumerable<string>

✓ OrderBy() is an extension method defined in System.Linq.Enumerable

So when you add:

using System.Linq;

You unlock over **100 extension methods**, including:

- Where
- Select
- OrderBy
- ThenBy
- GroupBy
- Count
- etc.

🧠 **Why no object creation?**

Because LINQ uses **deferred execution**, meaning:

- It **creates no collection** until you start iterating.
- It's not executed until the foreach loop.

So sortedList is actually an *iterator*, not a list.

You saw that earlier:

Output:

```
Type of sortedList: System.Linq.OrderedEnumerable`2[System.String,System.String]
```

This is just a wrapper that knows **how to sort items when you iterate**.

🟦 **Short summary**

Concept	Meaning
using System.Linq	Adds extension methods to IEnumerable
LINQ methods	Are static extension methods inside Enumerable class
No object created	Because LINQ uses deferred execution
LINQ query syntax	Compiles into extension method calls

❌ No, you cannot place select before where in LINQ query syntax

This will NOT compile:

```
var result = from s in students
          select s
          where s.Grade == userGrade;
```

Because **query syntax follows a strict grammar**, similar to SQL:

✅ Correct order in LINQ query syntax

from → where → orderby → select

So the valid pattern is:

```
from item in source
where condition
select item;
```

🧠 Why the order matters?

Because:

- from declares the *range variable* → s
- where filters the range variable
- select must be the **final projection**

select tells LINQ:

➡️ "what should the result contain?"

If you place select earlier, the where would have no defined source.

✔ If you want an intuitive, "method-like" syntax

You can use **method syntax** (which feels more natural to many developers):

```
var result = students
          .Select(s => s)
          .Where(s => s.Grade == userGrade);
```

Or usually just:

```
var result = students.Where(s => s.Grade == userGrade);
```

LINQ method syntax allows any order you want, because it's just chaining functions.

🟦 Summary

Query syntax	Method syntax
Strict order required	Flexible chaining
from → where → select	Select().Where().OrderBy()
More readable	More powerful