⭐ Sample Data Used in All Examples

```
var numbers = new List<int> { 5, 10, 15, 20, 25, 30 };


var students = new List<Student>
{
   new Student { Id = 1, Name = "Amit",  Marks = 80, Gender = "Male" },
   new Student { Id = 2, Name = "Sana",  Marks = 95, Gender = "Female" },
   new Student { Id = 3, Name = "John",  Marks = 60, Gender = "Male" },
   new Student { Id = 4, Name = "Riya",  Marks = 72, Gender = "Female" },
   new Student { Id = 5, Name = "Alex",  Marks = 88, Gender = "Male" }
};
```

🧩 1. Where() → Filtering

Filters records based on condition.

```
var result = students.Where(s => s.Marks > 80);
```

Output: Sana, Alex

🧩 2. Select() → Projection (Selecting specific fields)

```
var names = students.Select(s => s.Name);
```

Output: ["Amit", "Sana", "John", "Riya", "Alex"]

🧩 3. SelectMany() → Flattening nested collections

```
var classes = new List<List<int>> {
   new List<int> {1,2},
   new List<int> {3,4}
};
```

var result = classes.SelectMany(c => c);

Output: 1,2,3,4

🧩 4. OrderBy() → Ascending order

var ordered = students.OrderBy(s => s.Marks);

🧩 5. OrderByDescending() → Descending order

var orderedDesc = students.OrderByDescending(s => s.Marks);

🧩 6. ThenBy() → Secondary sorting

var sorted = students.OrderBy(s => s.Gender).ThenBy(s => s.Marks);

🧩 7. ThenByDescending() → Secondary sorting descending

var sorted = students.OrderBy(s => s.Gender)

.ThenByDescending(s => s.Marks);

🧩 8. GroupBy() → Grouping

var groups = students.GroupBy(s => s.Gender);

Groups:

- Male → Amit, John, Alex
- Female → Sana, Riya

🧩 9. ToList(), ToArray(), ToDictionary()

ToList()

var list = students.Where(s => s.Marks > 80).ToList();

ToDictionary()

var dict = students.ToDictionary(s => s.Id, s => s.Name);

🧩 **10. First() → Return first item**

var first = students.First();

🧩 **11. FirstOrDefault() → Return first or null**

var firstOrNull = students.FirstOrDefault(s => s.Marks > 100);

🧩 **12. Single() → Exactly one item must match**

var one = students.Single(s => s.Id == 1);

🧩 **13. SingleOrDefault() → One or none**

var oneOrNull = students.SingleOrDefault(s => s.Id == 10);

🧩 **14. Last(), LastOrDefault()**

var last = numbers.Last();

var lastOrNull = numbers.LastOrDefault(n => n > 100);

🧩 **15. Count()**

int count = students.Count(s => s.Marks > 80);

🧩 **16. Sum()**

int total = numbers.Sum();

🧩 17. Average()

```
double avg = students.Average(s => s.Marks);
```

🧩 18. Min() / Max()

```
int maxMarks = students.Max(s => s.Marks);

int minMarks = students.Min(s => s.Marks);
```

🧩 19. Any() → Check if any match exists

```
bool hasTopper = students.Any(s => s.Marks > 90);
```

🧩 20. All() → Check if all match condition

```
bool allPassed = students.All(s => s.Marks > 35);
```

🧩 21. Contains()

```
bool exists = numbers.Contains(15);
```

🧩 22. Distinct()

```
var distinct = numbers.Distinct();
```

🧩 23. Union() → Combine without duplicates

```
var result = numbers.Union(new List<int> {20, 40, 50});
```

🧩 24. Intersect() → Common values

```
var result = numbers.Intersect(new List<int> {15, 20, 100});
```

🧩 25. Except() → A minus B

```
var result = numbers.Except(new List<int> {10, 20});
```

🧩 26. Take() → First N items

```
var first3 = numbers.Take(3);
```

🧩 27. Skip() → Skip N items

```
var after3 = numbers.Skip(3);
```

🧩 28. TakeWhile()

```
var output = numbers.TakeWhile(n => n < 20);
```

Output: 5, 10, 15

🧩 29. SkipWhile()

```
var result = numbers.SkipWhile(n => n < 20);
```

Output: 20, 25, 30

🧩 30. Join() → Inner Join between sequences

```
var joinResult =
   students.Join(
      numbers,
      student => student.Id,
      number => number / 10,
      (student, number) => new { student.Name, number }
```

```
);
```

🧩 31. GroupJoin()

```
var groupJoin =
    students.GroupJoin(
        numbers,
        s => s.Id,
        n => n / 10,
        (s, nums) => new { s.Name, nums }
    );
```

🧩 32. Zip()

Combines two sequences element-wise.

```
var zipped = numbers.Zip(new[] { "A", "B", "C" },
                         (n, s) => n + "-" + s);
```

🧩 33. Reverse()

```
var reversed = numbers.Reverse();
```

🧩 34. OfType<T>() → Filter by type

```
var mixed = new List<object> {1, "hello", 2, "test"};
var ints = mixed.OfType<int>();
```

🧩 35. DefaultIfEmpty()

```
var empty = new List<int>();
var result = empty.DefaultIfEmpty(100);
```

Output: 100

🎯 Summary Table (Cheat Sheet)

Category	Operators
Filtering	Where, OfType
Projection	Select, SelectMany
Sorting	OrderBy, OrderByDescending, ThenBy, ThenByDescending
Grouping	GroupBy
Joining	Join, GroupJoin
Set Operations	Distinct, Union, Intersect, Except
Quantifiers	Any, All, Contains
Aggregation	Count, Sum, Max, Min, Average
Element	First, FirstOrDefault, Single, SingleOrDefault, Last
Partitioning	Take, Skip, TakeWhile, SkipWhile
Conversion	ToList, ToArray, ToDictionary
Others	Zip, Reverse, DefaultIfEmpty