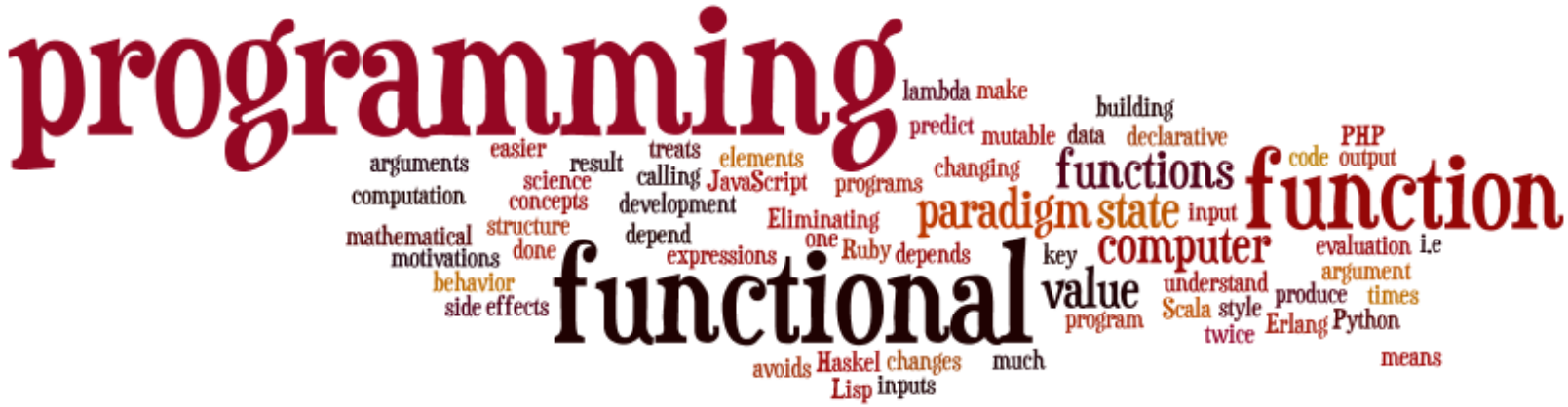
Windows Programming with C#
LINQ

Lương Trần Hy Hiến
hienlth@hcmup.edu.vn
085.4774.690

Agenda

- Functional Programming – Lambda expression
- LINQ

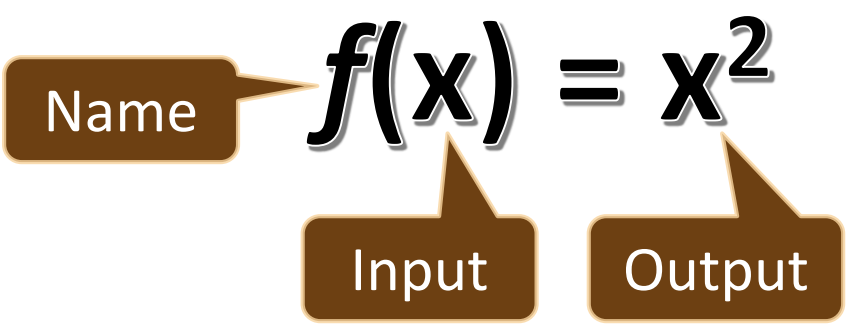


Functional Programming

Paradigms, Concepts

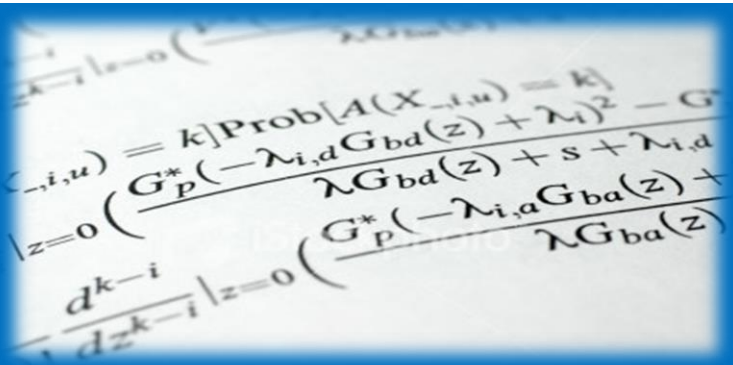
What is Function?

❑Mathematic function



x	$f(x)$
3	9
1	1
0	0
4	16
-4	16

A function is a special relationship where each input has a single output.



Lambda Expressions

Lambda Expressions

- A lambda expression is an unnamed function with parameters and a body
- Lambda syntax

```
(parameters) => {body}
```

  - Use the lambda operator `=>`
    - Read as "goes to"
  - Parameters can be enclosed in parentheses ( )
  - The body holds the expression or statement and can be enclosed in braces { }

Lambda Expressions (2)

- Implicit lambda expression

```
msg => Console.WriteLine(msg);
```

- Explicit lambda expression

```
(String msg) => { Console.WriteLine(msg); }
```

- Zero parameters

```
() => { Console.WriteLine("hi"); }
```

- More parameters

```
(int x, int y) => { return x + y; }
```

Problem: Sort Even Numbers

- Read Integers from console/Textbox
- Print even numbers
- Print sorted even numbers
- Use 2 Lambda Expresions

4, 2, 1, 3, 5, 7, 1, 4, 2, 12

2, 2, 4, 4, 12

Solution: Sort Odd Numbers

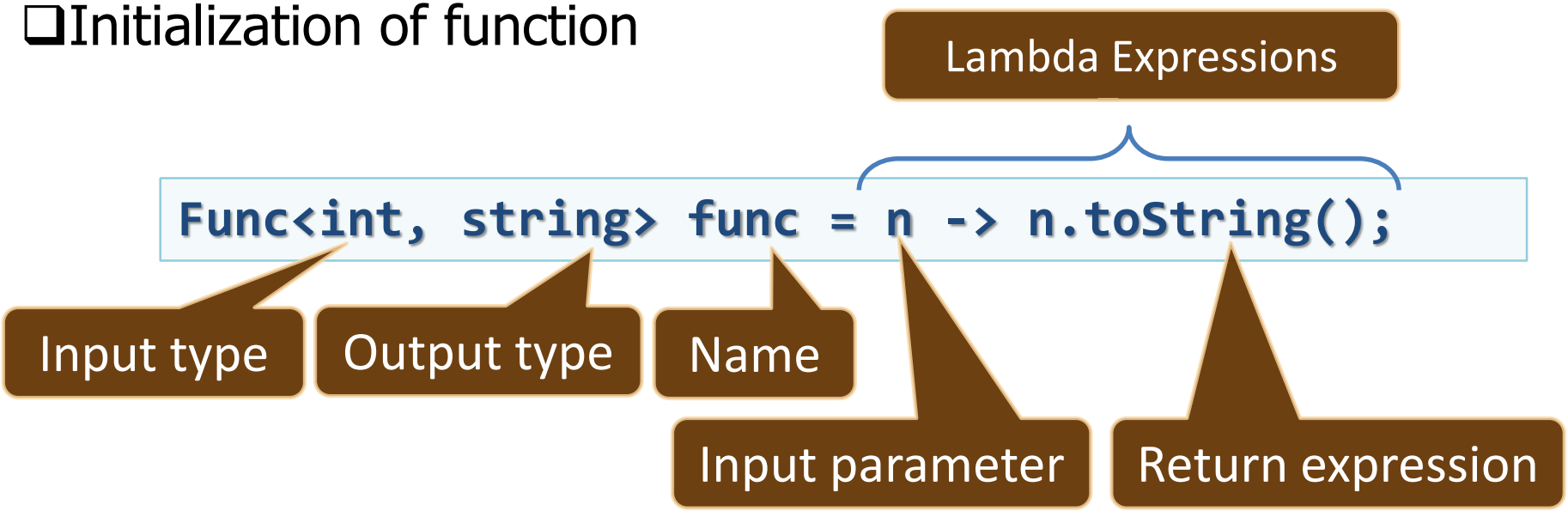
```
Console.ReadLine()/txtInput.Text
       .Split(new string[] { “, ” },
          StringSplitOptions.RemoveEmptyEntries)
       .Select(n => int.Parse(n))
       .Where(n => n % 2 != 0)
       .OrderBy(n => n)
       .ToList()
       .ForEach(n => Console.WriteLine(n));
```

Lambda Expressions

Exercises in class

C# Func

- Initialization of function



- Input and output type can be different type
- Input and output type must be from type which we declare them

Action<T>

- In .NET `Action<T>` is a void method:

```
private void Print(string message)
{
    Console.WriteLine(message);
}
```

- Instead of writing the method we can do:

```
Action<string> print = message => Console.WriteLine(message);
```

- Then we use it like that:

```
print("pesho");
print(5.ToString());
```

Problem: Sum Numbers

- Read numbers from console/Textbox
- Use your own Function for parse
- Print Count of numbers
- Print Sum


```
4, 2, 1, 3, 5, 7, 1, 4, 2, 12
```

```
10
41
```

Solution: Sum Numbers

```
var numbers = Console.ReadLine();//4, 2, 1, 3, 5, 7, 1, 4, 2, 12

Func<string, int > parser = n => int.Parse(n);

Console.WriteLine(numbers.Split(new string[] {", "},
                  StringSplitOptions.RemoveEmptyEntries)
                        .Select(parser)
                        .Sum());
```

Problem: Count Uppercase Words

- Read text from console/Textbox
- Count how many words start with Uppercase
- Print count and words
- Use Predicate

```
The following example shows how to use Predicate
```

```
The
Predicate
```

Solution: Count Uppercase Words

```
var words = Console.ReadLine()
            .Split(new string[] {" "},
                StringSplitOptions.RemoveEmptyEntries);

Func<string, bool> checker = n => n[0] == n.ToUpper()[0];

words.Where(checker)
     .ToList()
     .ForEach(n => Console.WriteLine(n));
```

Problem: Add VAT

- Read from console some item's prices
- Add VAT of **20%** to all of them
- Use UnaryOperator


```
1.38, 2.56, 4.4
```


```
Prices with
VAT:
1,66
3,07
5,28
```

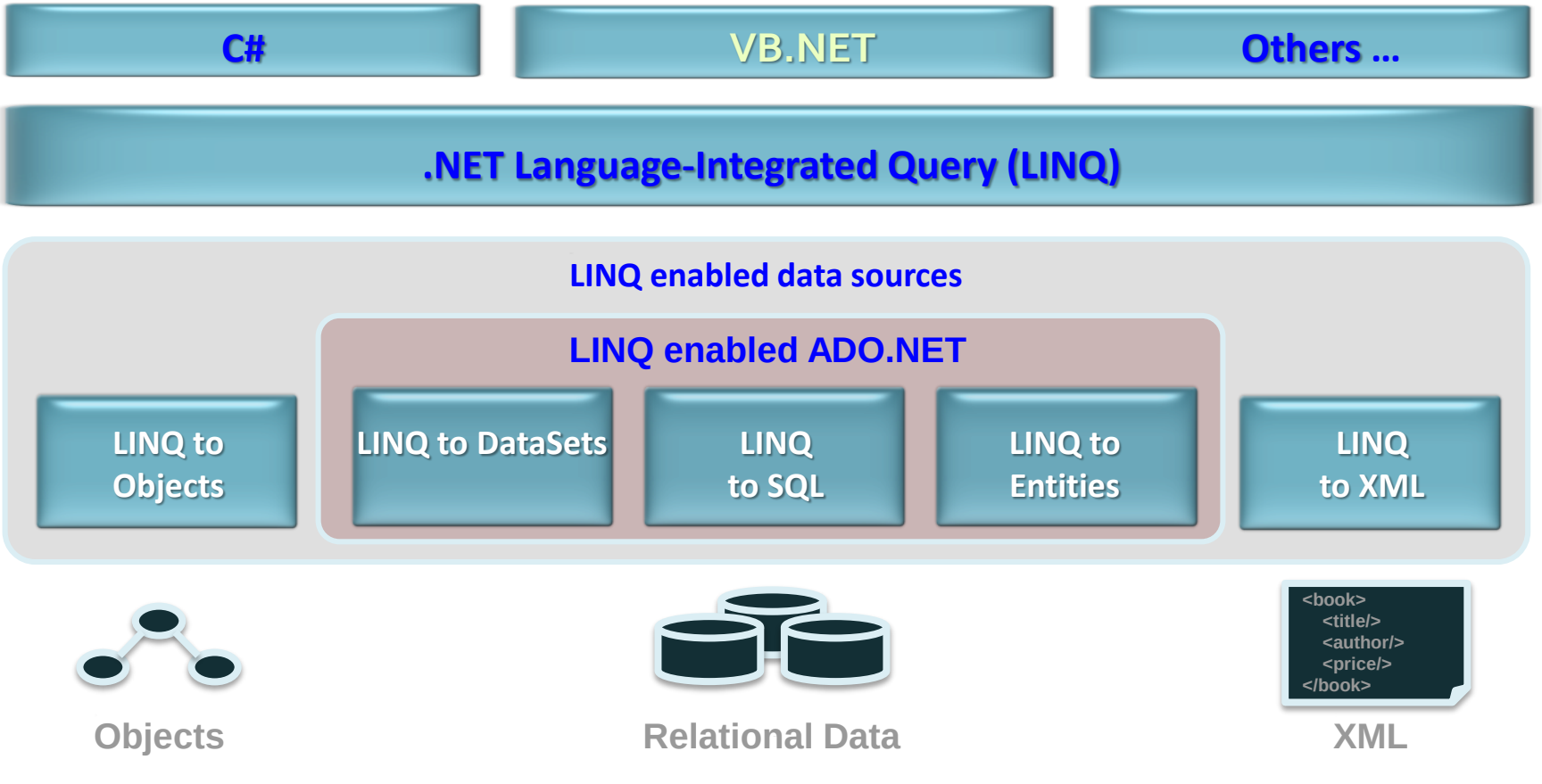
Solution: Add VAT

```
Console.ReadLine()
       .Split(new string[] { ", " },
           StringSplitOptions.RemoveEmptyEntries)
       .Select(double.Parse)
       .Select(n => n * 1.2)
       .ToList()
       .ForEach(n => Console.WriteLine($"{n:n2}"));
```

Action<T> and Func<T>

Exercises in class

LINQ to *


C#
VB.NET
Others ...
.NET Language-Integrated Query (LINQ)
LINQ enabled data sources
LINQ enabled ADO.NET
LINQ to Objects
LINQ to DataSets
LINQ to SQL
LINQ to Entities
LINQ to XML
<book>
<title/>
<author/>
<price/>
</book>
Objects
Relational Data
XML

LINQ: Simple Operations

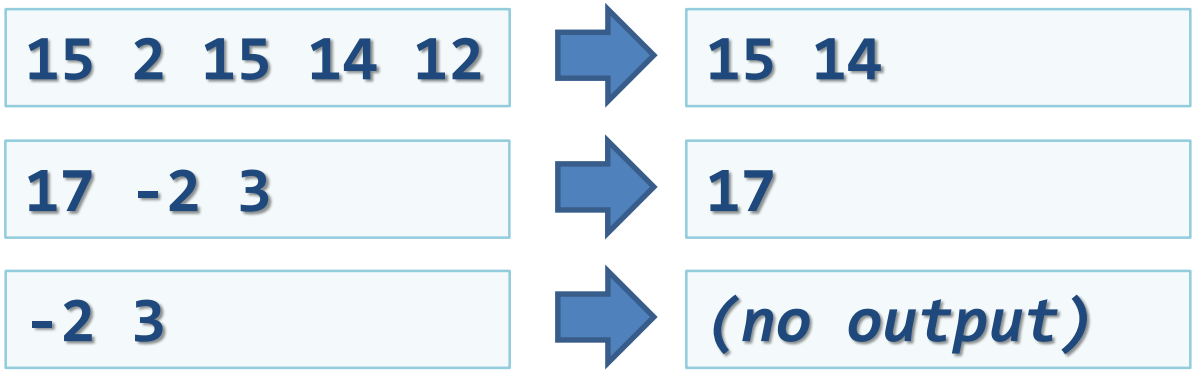
- **Where()**
  - Searches by given condition
- **First() / FirstOrDefault()**
  - Gets the first matched element
- **Last() / LastOrDefault()**
  - Gets the last matched element
- **Select()**
  - Makes projection (conversion) to another type
- **OrderBy() / ThenBy() / OrderByDescending()**
  - Orders a collection

LINQ: Simple Operations (2)

- `Any()`
  - Checks if any element matches a condition
- `All()`
  - Checks if all elements match a condition
- `Distinct()`
  - Returns only the unique elements
- `Skip()` / `Take()`
  - Skips or takes X number of elements

Problem: Take Two

- Create a program that:
  - Reads a sequence of integers
  - Finds all unique elements, such that $10 \leq n \leq 20$
  - Prints only the first 2 elements

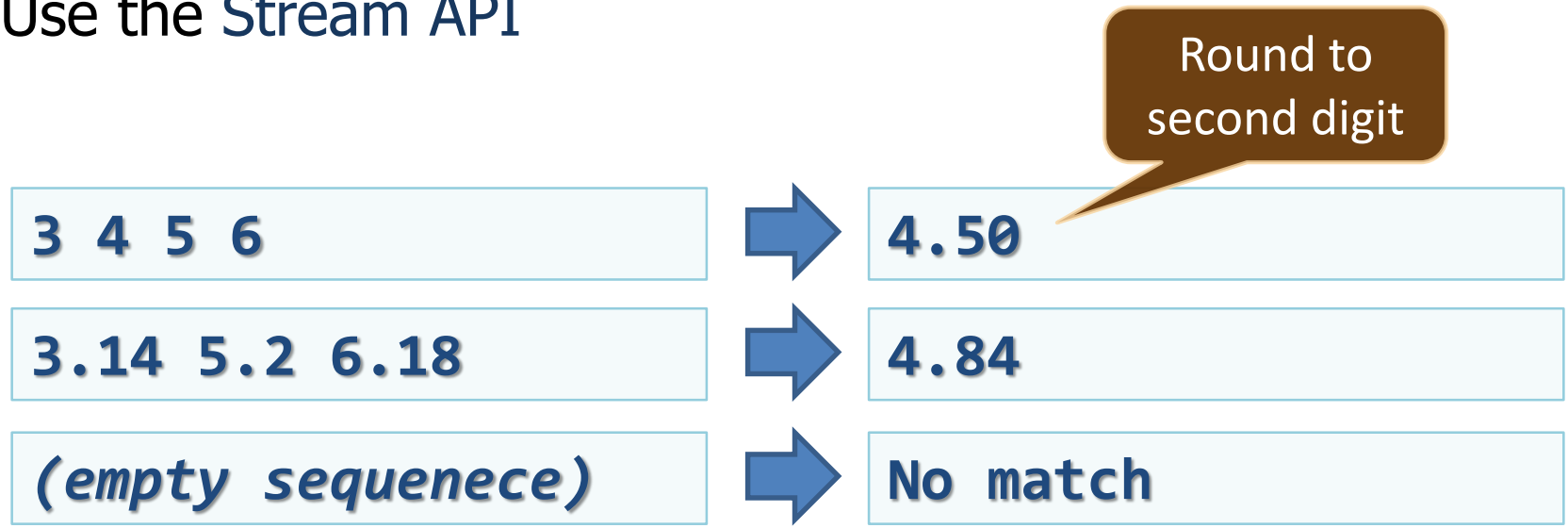


Solution: Take Two

```
Console.ReadLine()
      .Split(' ')
      .Select(int.Parse)
      .Where(n => n >= 10 && n <= 20)
      .Distinct()
      .Take(2)
      .ToList()
      .ForEach(n => Console.Write(n + " "));
```

Problem: Average of Doubles

- Read a sequence of double numbers
- Find the average of all elements
- Use the Stream API

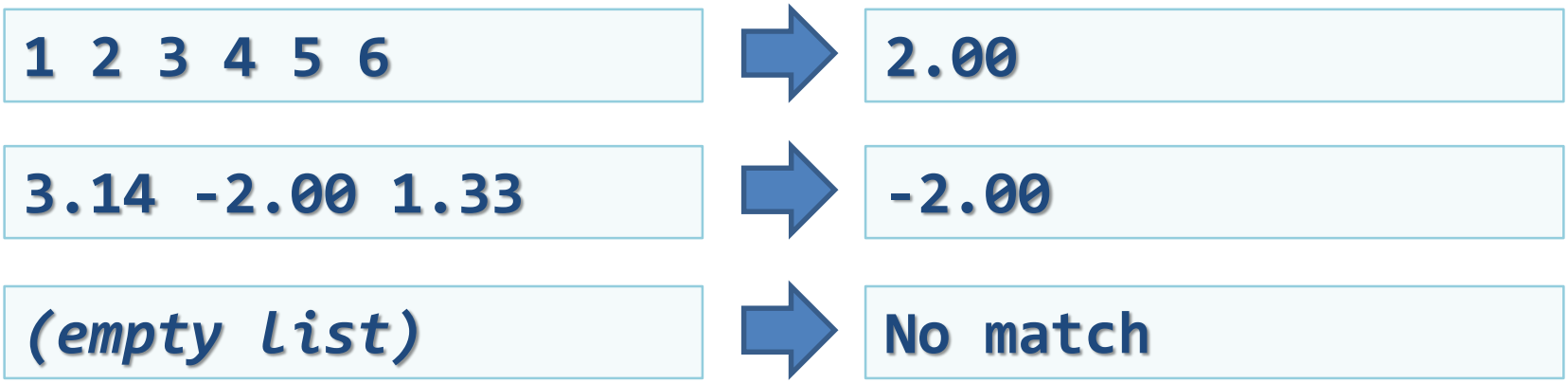


Solution: Average of Doubles

```
var average = Console.ReadLine()
                     .Split(' ')
                     .Select(double.Parse)
                     .Average();
Console.WriteLine($"{average:f2}");
```

Problem: Min Even Number

- Read a sequence of numbers
- Find the min of all even numbers
- Use the LINQ



Solution: Min Even Number

```
Optional<Double> min =
      Arrays.stream(
            scanner.nextLine().split("\\s+"))
      .filter(n -> !n.isEmpty())
      .map(Double::valueOf)
      .filter(n -> n % 2 == 0)
      .min(Double::compare);
```

GroupBy()

- Transforms a collection into groups. Each group has a key.

```
List<IGrouping<bool, int>> groups =
                arr.GroupBy(num => num % 2 == 0)
                   .ToList()
```

- In the example above we have `List<>` of `IGrouping<K, V>`.

```
foreach(var group in groups)
{
    Console.WriteLine("Is even: {0} - ", group.Key);
    Console.WriteLine(string.Join(", ", group));
}
```

ToDictionary()

- Transforms a collection to dictionary

```
Dictionary<bool, List<int>> dict =
arr.GroupBy(num => num % 2 == 0)
   .toDictionary(g => g.Key, g => g.ToList());
```

- In this example we transformed the `List<IGrouping<K, V>>` from the previous slide to `Dictionary<K, List<V>>`.

Exercises in class

Q&A