

Dictionaries, Lambda and LINQ



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

sli.do

#prgm-for-qa

1. Dictionaries

- Dictionary <key, value>
- SortedDictionary <key, value>

2. Lambda Expressions

3. LINQ

- Filtering
- Mapping
- Ordering






Dictionaries

A Collection of Key and Value Pairs

Dictionaries

- **Dictionaries** are arrays indexed by keys
 - Not by the numbers 0, 1, 2, ... (like arrays)
- Hold a set of pairs {**key** → **value**}



| Key | Value |
|------------|-------------|
| John Smith | +1-555-8976 |
| Lisa Smith | +1-555-1234 |
| Sam Doe | +1-555-5030 |

- **Dictionary**<K, V> – a collection of key and value pairs
- The keys are **unique**
- Keeps the keys in their **order of addition**
- Uses a hash-table + list

```
var fruits = new Dictionary<string, double>();  
fruits["banana"] = 2.20;  
fruits["apple"] = 1.40;  
fruits["kiwi"] = 3.20;
```

- **SortedDictionary<K, V>**
- Keeps its keys always sorted
- Uses a balanced search tree

```
var fruits = new SortedDictionary<string, double>();  
fruits["kiwi"] = 4.50;  
fruits["orange"] = 2.50;  
fruits["banana"] = 2.20;
```

- Create an empty list of integers

```
var phoneNumbers = new Dictionary<string, string>();  
// Add elements  
phoneNumbers["Peter"] = "+359 882 11 22 33";  
phoneNumbers["Ana"] = "+359 2 99 88 77";
```

- Using a target-type **new** expression

```
Dictionary<string, int> fruits = new() {  
    { "Kiwi", 3 },  
    { "Apple", 5 }  
};
```


- **Add(key, value)** method

```
var airplanes = new Dictionary<string, int>();  
airplanes.Add("Boeing 737", 130);  
airplanes.Add("Airbus A320", 150);
```

- **Remove(key)** method

```
var airplanes = new Dictionary<string, int>();  
airplanes.Add("Boeing 737", 130);  
airplanes.Remove("Boeing 737");
```

■ ContainsKey(key)

```
var dictionary = new Dictionary<string, int>();  
dictionary.Add("Airbus A320", 150);  
if (dictionary.ContainsKey("Airbus A320"))  
    Console.WriteLine($"Airbus A320 key exists");
```

■ ContainsValue(value)

```
var dictionary = new Dictionary<string, int>();  
dictionary.Add("Airbus A320", 150);  
Console.WriteLine(dictionary.ContainsValue(150)); // True  
Console.WriteLine(dictionary.ContainsValue(100)); // False
```

Problem: Count Real Numbers

- Read a list of real numbers and print them in ascending order along with their number of occurrences

8 2.5 2.5 8 2.5



2.5 -> 3

8 -> 2

1.5 5 1.5 3



1.5 -> 2

3 -> 1

5 -> 1

Solution: Count Real Numbers

```
double[] nums = Console.ReadLine().Split(' ')
    .Select(double.Parse).ToArray();
var counts = new SortedDictionary<double, int>();
foreach (var num in nums)
    if (counts.ContainsKey(num))
        counts[num]++;
    else
        counts[num] = 1;
foreach (var num in counts)
    Console.WriteLine($"{num.Key} -> {num.Value}");
```

counts[num] will
hold the count of
times a num occurs in
nums

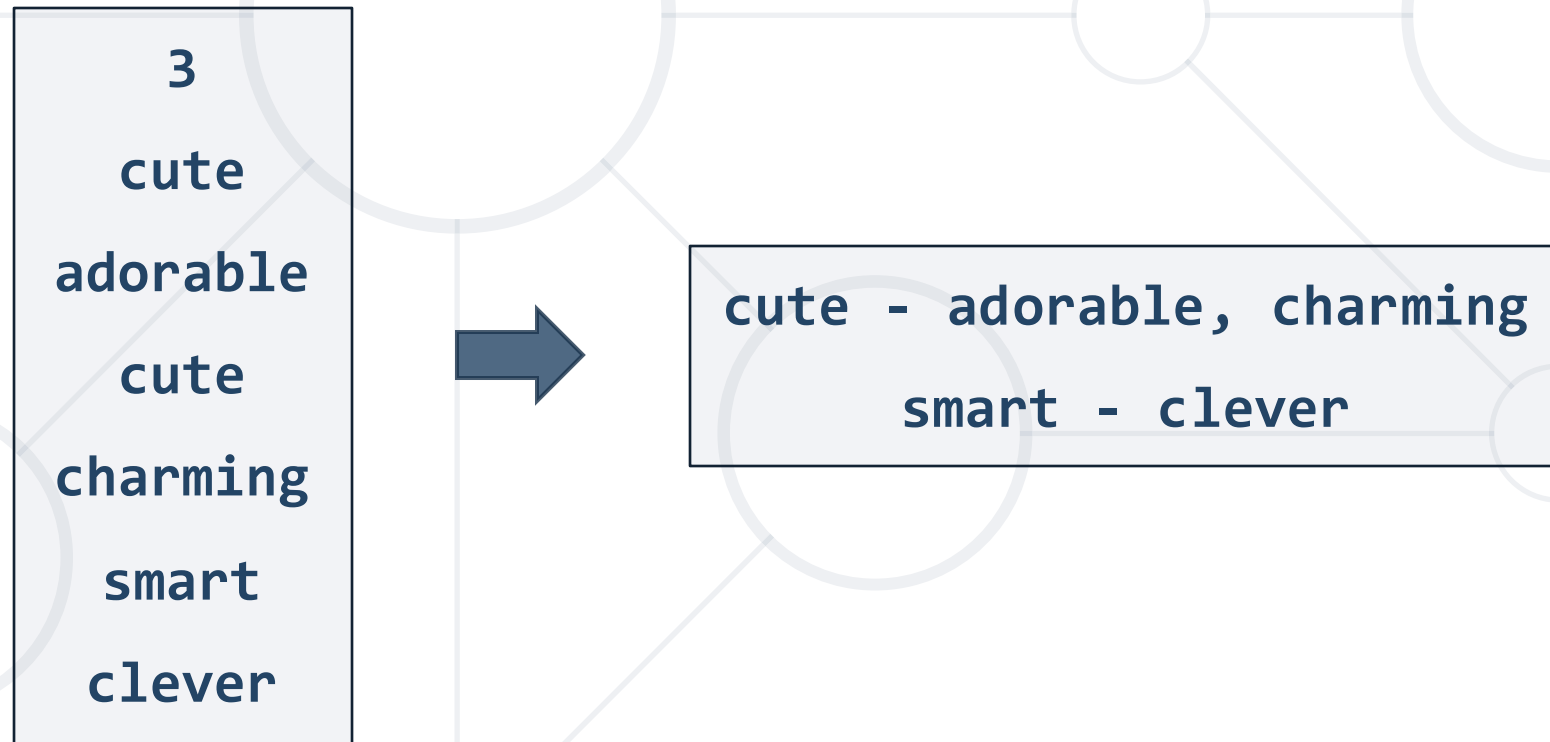
- Using **foreach** loop
- Iterates through objects of type **KeyValuePair<K, V>**
- Cannot modify the dictionary (**read-only**)

```
var fruits = new Dictionary<string, double>();  
fruits.Add("banana", 2.20);  
fruits.Add("kiwi", 4.50);  
foreach (var fruit in fruits)  
    Console.WriteLine($"{fruit.Key} -> {fruit.Value}");
```

fruit.**Key** → fruit name
fruit.**Value** → fruit price

Problem: Word Synonyms

- Read $2 * N$ lines of pairs - word and synonym
- Each word can have multiple synonyms



Solution: Word Synonyms

```
int n = int.Parse(Console.ReadLine());  
var words = new Dictionary<string, List<string>>();  
for (int i = 0; i < n; i++) {  
    string word = Console.ReadLine();  
    string synonym = Console.ReadLine();  
    if (words.ContainsKey(word) == false)  
        words.Add(word, new List<string>());  
    words[word].Add(synonym);  
}
```



Anonymous Functions

Lambda Expressions

Lambda Expressions

- A lambda expression is an anonymous function containing expressions and statements

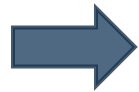
```
a => a > 5;
```

- Lambda expressions
 - Use the lambda operator `=>`
 - Read as "**goes to**"
 - The **left** side specifies the **input** parameters
 - The **right** side holds the **expression** or **statement**



- Lambda functions are inline methods (functions) that take input parameters and return values

`x => x / 2`



```
static int Func(int x) { return x / 2; }
```

`x => x != 0`



```
static bool Func(int x) { return x != 0; }
```

`() => 42`



```
static int Func() { return 42; }
```

- **Min()** – finds the **smallest** element in a collection

```
new List<int>() { 1, 2, 3, 4, -1, -5, 0, 50 }.Min() // -5
```

- **Max()** – finds the **largest** element in a collection

```
new int[] { 1, 2, 3, 40, -1, -5, 0, 5 }.Max() // 40
```

- **Sum()** – finds the **sum** of all elements in a collection

```
new long[] {1, 2, 3, 4, -1, -5, 0, 50}.Sum() // 54
```

- **Average()** – finds the **average** of all elements in a collection

```
new int[] {1, 2, 3, 4, -1, -5, 0, 50}.Average() // 6.75
```

- **Select()** manipulates elements in a collection

```
var nums = Console.ReadLine()  
    .Split()  
    .Select(int.Parse);
```

```
string[] words = { "abc", "def" } ;  
var result = words.Select(w => w + "x");  
// words -> abcx, defx
```

- Using **ToArray()**, **ToList()** to convert collections

```
int[] nums = Console.ReadLine()  
    .Split()  
    .Select(number => int.Parse(number))  
    .ToArray();
```

```
List<double> nums = Console.ReadLine()  
    .Split()  
    .Select(double.Parse)  
    .ToList();
```

- Using **Where()**

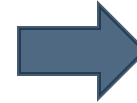
```
int[] nums = Console.ReadLine()  
    .Split()  
    .Select(int.Parse)  
    .Where(n => n > 0)  
    .ToArray();
```



Problem: Word Filter

- Read a string array
- Print only words, whose length is even

kiwi orange banana apple



kiwi
orange
banana

pizza cake pasta chips



cake

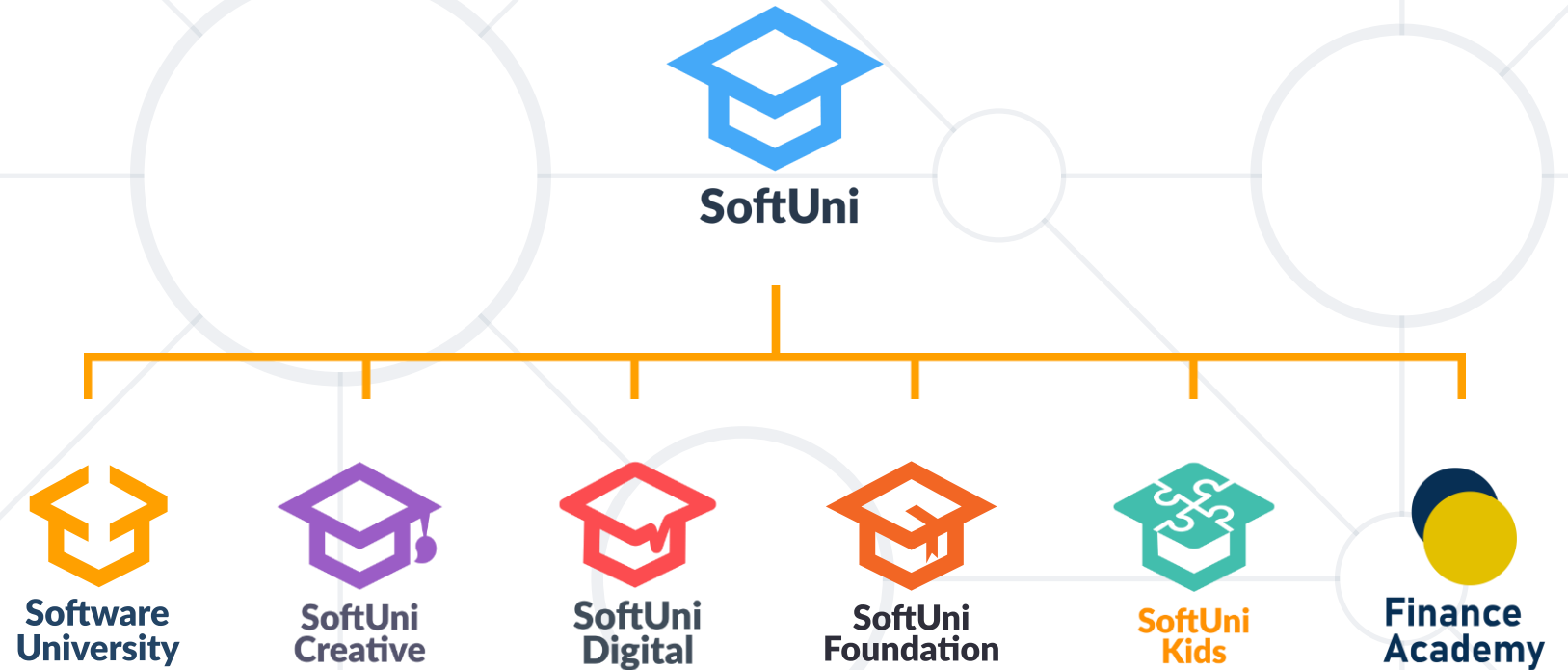
Solution: Word Filter

```
string[] words = Console.ReadLine()  
                .Split()  
                .Where(w => w.Length % 2 == 0)  
                .ToArray();  
foreach (string word in words)  
    Console.WriteLine(word);
```


- Dictionaries hold **{key → value}** pairs
 - Keys holds a set of **unique keys**
 - Values holds a collection of values
 - Iterating over dictionary takes the entries as **KeyValuePair<K, V>**
- Lambda and LINQ helps collection processing



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**

 **Flutter**TM
International

INDEAVR
Serving the high achievers



AMBITIONED

 **DRAFT
KINGS**



**SOFTWARE
GROUP**



BOSCH



Postbank

Решения за твоето утре

 **PHAR
VISION**



SmartIT

DXC
TECHNOLOGY

createX

- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, about.softuni.bg
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

