

The image features the text "LINQ 2" in a bold, black, sans-serif font, centered horizontally. The text is flanked by two large, black, L-shaped brackets. The left bracket is positioned in the upper-left quadrant, with its vertical bar extending downwards and its horizontal bar extending to the right. The right bracket is positioned in the lower-right quadrant, with its horizontal bar extending to the left and its vertical bar extending upwards. The background is a light, neutral color.

LINQ 2

Метод ElementAt

```
public static TSource ElementAt<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int index);
```

Метод ElementAt

```
string[] names = { "Hartono, Tommy", "Adams,  
Terry", "Andersen, Henriette Thaulow", "Hedlund,  
Magnus", "Ito, Shu" };
```

```
Random random = new  
Random(DateTime.Now.Millisecond);
```

```
string name = names.ElementAt(random.Next(0,  
names.Length));
```

```
Console.WriteLine("The name chosen at random is  
'{0}'.", name);
```

Метод `ElementAtOrDefault`

```
public static TSource ElementAtOrDefault<TSource>  
(this System.Collections.Generic.IEnumerable<TSource>  
source, int index);
```

Метод `ElementAtOrDefault`

```
int index = 20;
```

```
string name = names.ElementAtOrDefault(index);
```

```
Console.WriteLine("The name chosen at index {0} is  
'{1}'.", index, String.IsNullOrEmpty(name) ? "<no name  
at this index>" : name);
```

Метод Reverse

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Reverse<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source) ;
```

Метод Reverse

```
char[] apple = { 'a', 'p', 'p', 'l', 'e' };  
char[] reversed = apple.Reverse().ToArray();  
foreach (char chr in reversed)  
{  
    Console.Write(chr + " ");  
}  
Console.WriteLine();
```

Метод Enumerable.OrderBy

```
public static  
System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, TKey> keySelector);
```


Метод Enumerable.OrderBy

```
public static void OrderByEx1 ()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                  new Pet { Name="Boots", Age=4 },
                  new Pet { Name="Whiskers", Age=1 } };

    IEnumerable<Pet> query = pets.OrderBy(pet => pet.Age);

    foreach (Pet pet in query)
    {
        Console.WriteLine("{0} - {1}", pet.Name, pet.Age);
    }
}
```

Метод Enumerable.OrderBy

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TKey> keySelector,  
System.Collections.Generic.IComparer<TKey> comparer) ;
```

Метод

Enumerable.OrderByDescending

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderByDescending<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TKey> keySelector,  
System.Collections.Generic.IComparer<TKey> comparer) ;
```

Метод

Enumerable.OrderByDescending

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderByDescending<TSource, TKey> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, TKey> keySelector) ;
```

Метод Enumerable.OrderBy

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Linq.IOrderedEnumerable<TSource> source,  
Func<TSource, TKey> keySelector,  
System.Collections.Generic.IComparer<TKey> comparer) ;
```

Метод Enumerable.OrderBy

```
public static System.Linq.IOrderedEnumerable<TSource>  
OrderBy<TSource, TKey> (this  
System.Linq.IOrderedEnumerable<TSource> source,  
Func<TSource, TKey> keySelector);
```

Метод Enumerable.OrderBy

```
string[] fruits = { "grape", "passionfruit", "banana",  
"mango", "orange", "raspberry", "apple", "blueberry" };
```

```
IEnumerable<string> query = fruits.OrderBy(fruit =>  
fruit.Length).ThenBy(fruit => fruit);
```

```
foreach (string fruit in query)  
{  
    Console.WriteLine(fruit);  
}
```

Метод Enumerable.OrderByDescending

```
public class CaseInsensitiveComparer : IComparer<string>
{
    public int Compare(string x, string y)
    {
        return string.Compare(x, y, true);
    }
}
```


Метод

Enumerable.OrderByDescending

```
public static void ThenByDescendingEx1 ()
{
    string[] fruits = { "apPLe", "baNanA", "apple", "APple",
        "orange", "BAnana", "ORANGE", "apPLE" };

    IEnumerable<string> query = fruits
        .OrderBy(fruit => fruit.Length)
        .ThenByDescending(fruit => fruit, new
            CaseInsensitiveComparer());
}
```

Метод Take

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Take<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int count);
```

Метод Take

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };
```

```
IEnumerable<int> topThreeGrades =  
    grades.OrderByDescending(grade =>  
grade).Take(3);
```

```
Console.WriteLine("The top three grades are:");
```

```
foreach (int grade in topThreeGrades)
```

```
{
```

```
    Console.WriteLine(grade);
```

```
}
```

Метод TakeWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
TakeWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource> source,  
Func<TSource, bool> predicate) ;
```

Метод TakeWhile

```
string[] fruits = { "apple", "banana", "mango",  
"orange", "passionfruit", "grape" };
```

```
IEnumerable<string> query = fruits.TakeWhile(fruit =>  
String.Compare("orange", fruit, true) != 0);
```

```
foreach (string fruit in query)  
{  
    Console.WriteLine(fruit);  
}
```

Метод TakeWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
TakeWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource,int,bool> predicate);
```

Метод TakeWhile

```
string[] fruits = { "apple", "passionfruit", "banana",  
"mango", "orange", "blueberry", "grape", "strawberry" };
```

```
IEnumerable<string> query =  
    fruits.TakeWhile((fruit, index) => fruit.Length >=  
index);
```

```
foreach (string fruit in query)  
{  
    Console.WriteLine(fruit);  
}
```

Метод Skip

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Skip<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int count);
```


Метод Skip

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };
```

```
IEnumerable<int> lowerGrades =  
    grades.OrderByDescending(g => g).Skip(3);
```

```
Console.WriteLine("All grades except the top three are:");
```

```
foreach (int grade in lowerGrades)
```

```
{
```

```
    Console.WriteLine(grade);
```

```
}
```

Метод SkipWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
SkipWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, bool> predicate) ;
```

Метод SkipWhile

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };  
IEnumerable<int> lowerGrades =  
    grades  
        .OrderByDescending(grade => grade)  
        .SkipWhile(grade => grade >= 80);  
Console.WriteLine("All grades below 80:");  
foreach (int grade in lowerGrades)  
{  
    Console.WriteLine(grade);  
}
```

Метод SkipWhile

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
SkipWhile<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource,int,bool> predicate);
```

Метод SkipWhile

```
int[] amounts = { 5000, 2500, 9000, 8000, 6500, 4000, 1500,  
5500 };
```

```
IEnumerable<int> query =
```

```
    amounts.SkipWhile((amount, index) => amount > index *  
1000);
```

```
foreach (int amount in query)
```

```
{
```

```
    Console.WriteLine(amount);
```

```
}
```

Метод SkipLast

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
SkipLast<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, int count);
```

Метод Select

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
Select<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, int, TResult> selector);
```

Метод Select

```
string[] fruits = { "apple", "banana", "mango",  
"orange", "passionfruit", "grape" };  
  
var query =  
    fruits.Select((fruit, index) =>  
        new { index, str = fruit.Substring(0, index) });  
  
foreach (var obj in query)  
{  
    Console.WriteLine("{0}", obj);  
}
```


Метод Select

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
Select<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource>  
source, Func<TSource, TResult> selector);
```

Метод Select

```
IEnumerable<int> squares =  
    Enumerable.Range(1, 10).Select(x => x * x);
```

```
foreach (int num in squares)  
{  
    Console.WriteLine(num);  
}
```

Метод SelectMany

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
SelectMany<TSource, TResult> (this  
System.Collections.Generic.IEnumerable<TSource>  
source,  
Func<TSource, System.Collections.Generic.IEnumerable  
<TResult>> selector);
```

Метод SelectMany

```
PetOwner[] petOwners =  
{ new PetOwner { Name="Higa, Sidney",  
                Pets = new List<string>{ "Scruffy", "Sam" } },  
  new PetOwner { Name="Ashkenazi, Ronen",  
                Pets = new List<string>{ "Walker", "Sugar" } },  
  new PetOwner { Name="Price, Vernetta",  
                Pets = new List<string>{ "Scratches", "Diesel" } }  
};
```

Метод SelectMany

```
IEnumerable<string> query1 =  
petOwners.SelectMany(petOwner => petOwner.Pets);
```

```
IEnumerable<List<String>> query2 =  
petOwners.Select(petOwner => petOwner.Pets);
```

Метод SelectMany

```
var query = petOwners .SelectMany(petOwner =>  
petOwner.Pets, (petOwner, petName) => new { petOwner,  
petName })
```

```
IEnumerable<string> query = petOwners.SelectMany((petOwner,  
index) => petOwner.Pets.Select(pet => index + pet));
```

Метод Zip

```
public static  
System.Collections.Generic.IEnumerable<TResult>  
Zip<TFirst, TSecond, TResult> (this  
System.Collections.Generic.IEnumerable<TFirst>  
first,  
System.Collections.Generic.IEnumerable<TSecond>  
second, Func<TFirst, TSecond, TResult>  
resultSelector);
```

Метод Zip

```
int[] numbers = { 1, 2, 3, 4 };  
string[] words = { "one", "two", "three" };  
  
var numbersAndWords = numbers.Zip(words, (first,  
second) => first + " " + second);  
  
foreach (var item in numbersAndWords)  
    Console.WriteLine(item);
```


Метод Zip

```
public static  
System.Collections.Generic.IEnumerable<ValueTuple<T  
First,TSecond>> Zip<TFirst,TSecond> (this  
System.Collections.Generic.IEnumerable<TFirst>  
first,  
System.Collections.Generic.IEnumerable<TSecond>  
second) ;
```

Метод Distinct

```
public static  
System.Collections.Generic.IEnumerable<TSource>  
Distinct<TSource> (this  
System.Collections.Generic.IEnumerable<TSource>  
source) ;
```

Метод Distinct

```
List<int> ages = new List<int> { 21, 46, 46, 55,  
17, 21, 55, 55 };
```

```
IEnumerable<int> distinctAges = ages.Distinct();
```

```
Console.WriteLine("Distinct ages:");
```

```
foreach (int age in distinctAges)
```

```
{
```

```
    Console.WriteLine(age);
```

```
}
```

Метод Distinct

```
public class Product : IEquatable<Product>
{
    public string Name { get; set; }
    public int Code { get; set; }
    public bool Equals(Product other)
    {
        if (Object.ReferenceEquals(other, null)) return false;
        if (Object.ReferenceEquals(this, other)) return true;
        return Code.Equals(other.Code) && Name.Equals(other.Name);
    }
}
```

Метод Distinct

```
public override int GetHashCode()  
{  
    int hashProductName = Name == null ? 0 :  
Name.GetHashCode();  
    int hashProductCode = Code.GetHashCode();  
    return hashProductName ^ hashProductCode;  
}
```