100 LINQ PUZZLES

for Job Interviews

and Certification Exams

Cristian Scutaru

Table of Contents

Introduction

These 100 puzzles, with multiple levels of difficulty, can help you quickly improve your essential knowledge and problem solving skills in **LINQ (Language-Integrated Query)**.

Master the functional programming style of LINQ for Objects on immutable sequences. We focused on the fluent method notation. You'll find some query notation as well, but no LINQ for XML. We covered most if not all Enumerable standard query operators. Our strategy was to skip the trivial, avoid the clutter, but remember the basics and repeat what really matters.

You must be already proficient in C# and comfortable with lambdas, extension methods and other advanced techniques. You must already know the basics of LINQ, as we don't teach LINQ here: these puzzles help you get better in LINQ.

The target audience starts with beginner C# developers and extends to expert C# programmers looking to test their skills. And have some fun in the process. Intermediate C# developers could better understand LINQ and its functional programming style.

We also target Software Engineers preparing for job coding interviews, or certification exams that require coding. LINQ is an essential component in solving complex algorithms and efficiently parsing data collections.

We've split the content into 5 quizzes with 20 puzzles each. The puzzles are introduced as either coding problems with alternative possible solutions, or multiple-choice knowledge-related questions. In a separate section, all puzzles have detailed answers, explanations and references you can check after you first try to solve them with no hints.

More than 50 coding problems have **one-click live C# source code** you can run online and change as you wish. Dozens of problems are inspired from the most voted

questions on **Stack Overflow**. Last quiz has problems asked in real **Job Coding Technical Interviews**, as described on LeetCode and elsewhere.

Start with the first quiz, which will tests your basic knowledge on LINQ, with essential questions. Write down each question number with your choice, on a separate piece of paper. Don't look at the answers yet.

Once you're done with the 20 puzzles of a quiz, check the following **Answers and Explanations** section, and review both the right and wrong choices. Read the detailed **Explanation** for each puzzle. Click on the **Try It Here** link – if there is any – to run your code online and change it as you wish. Go through the recommended external **References** to learn more.

Continue with the other quizzes, in increased level of difficulty. The quizzes are from beginner to expert, from easy to intermediate to difficult - great to prepare you for the complexity of any **Certification Exams** that require coding. Knowledge-based questions are gradually replaced by coding problems in LINQ, with multiple alternatives. Repeat all quizzes and check your marks: you should start seeing better times and higher passing scores. *Good luck!*

An interactive version of this book has been implemented on Udemy as **100 Interactive LINQ Puzzles**.

Puzzles

Easy Basic LINQ

Question 1:

Which of the following return a sequence of IGrouping objects? (check all that apply)

A)

```
persons.OrderBy(p => p.Name);
```

B)

```
persons.GroupBy(p => p.Name);
```

C)

```
persons.ToLookup(p => p.Name);
```

D)

```
persons.Select(p => p.Name);
```

Question 2:

What can you use in LINQ to emulate the SQL LIKE operator? (check all that apply)

A) Contains, StartsWith and EndsWith.
B) The standard query operator Exists.
C) Regular expressions.
D) IndexOf and LastIndexOf.

Question 3:

What if a Select call returns nothing? (select one)

A) The result will be null.
B) An exception will be thrown, unless you use SelectOrDefault.
C) The result is an empty IEnumerable sequence.
D) The result is an IEnumerable sequence object with the Empty property true.

Question 4:

You need to check if there is at least one sequence element satisfying a condition.

Should you use Any or All? (select one)

A) Use rather Any, because if will stop as soon as an element is found.
B) Use rather All, because it will always check for all elements.
C) It doesn't matter, they have similar performance.
D) Use rather Exists.

Question 5:

In which of the following use cases would you use Single vs First? (check all that apply)

A) You use Single when you're looking for a Customer by the primary key.
B) You use Single when you want to extract only one element from a collection.
C) You use First when your collection is sorted in ascending order, otherwise use Last.
D) You use First when you want only the most recent News entry in an ordered collection.

Question 6:

What are typical ways to duplicate a collection? (check all that apply)

A) With the Clone method, if it implements ICloneable.
B) With the CopyTo method, if it implements ICopyable.
C) With specialized ToList, ToArray, ToDictionary etc, if implemented by LINQ.

Question 7:

Provide a fix for the following query, that uses an undefined w variable: (select one)

```
from word in words
    w = word.ToLower()
    where w[0] == 'a'
    select word;
```

A) Add VAR before w.
B) Add LET before w.
C) You cannot define new variables within a LINQ query.

Question 8:

A custom string collection you want to enumerate already defines a Where method with the exact same signature as Enumerable.Where.

How can you call the LINQ method? (check all that apply)

A)

```
coll.Where(x => false);
```

B)

```
coll.ToEnumerable().Where(x => false);
```

C)

```
coll.AsEnumerable().Where(x => false);
```

D)

```
((IEnumerable<string>) coll).Where(x => false);
```

E)

```
coll.Cast<IEnumerable<string>>().Where(x => false);
```

Question 9:

When would you need both SkipWhile and TakeWhile on a list of strings? (select one)

A) When you want to return a portion between "start" and "stop".
B) When you want to return K items coming after "start" in the list.
C) When you skip K list items and return all items until "end".

Question 10:

You find no LINQ query operators associated to a typical System.Array.

What could be the cause? (select one)

A) LINQ must be separately downloaded and installed.
B) System.Linq assembly is not loaded and referenced.
C) System.Linq namespace is not referenced.

Question 11:

Which of the following LINQ queries are syntactically correct? (check all that apply)

A)

```
select p
    from persons
    where Name starts with "J";
```

B)

```
select p
    from p in persons
    where Name.StartsWith("J");
```

C)

```
from p in persons
    where Name.StartsWith("J")
    select p;
```

D)

```
from p in persons
    where p.Name.StartsWith("J")
    select p;
```

E)

```
from p in persons
    where Name like "J*"
    select p;
```

Question 12:

Which of the following pairs of operator methods uses deferred execution? (select one)

A) Select and Where.
B) First and Single.
C) ToList and ToArray.
D) Min and Average.

Question 13:

Return the first Person with an empty Name, if any. (select one)

A)

```
persons.First(p => p.Name == string.Empty);
```

B)

```
persons.Where(p => p.Name == string.Empty).First();
```

C)

```
persons.FirstOrDefault(p => p.Name == string.Empty);
```

Question 14:

Which of the following pairs of LINQ operations do NOT preserve the order of the initial collection? (check all that apply)

A) OrderBy and OrderByDescending.
B) Reverse and ThenBy.
C) ToArray and ToList.
D) ToLookup and ToDictionary.

Question 15:

To what specific method do you map the SQL IN clause in LINQ? (select one)

```
SELECT * FROM Users
    WHERE Role IN ("Admin", "User", "Limited");
```

A) Where
B) Include
C) In
D) Contains

Question 16:

Which of the following standard collections and types are not "LINQ-enumerable"? (check all that apply)

A) DataRowCollection and DataTable.
B) List and IQueryable.
C) String and Array.

D) Range and Index.
E) Stack and Queue.
F) Hashtable and HashSet<T>.

Question 17:

Proper way to sort a list of Person objects by FirstName and LastName? (select one)

A)

```
persons.OrderBy(p => p.FirstName + " " + p.LastName);
```

B)

```
persons.OrderBy(p => new { p.FirstName, p.LastName });
```

C)

```
persons.OrderBy(p => p.FirstName)
    .OrderBy(p => p.LastName);
```

D)

```
persons.OrderBy(p => p.FirstName)
    .ThenBy(p => p.LastName);
```

Question 18:

Which query operators perform in-place update operations? (select one)

A) Append and Prepend.
B) Add and Insert.
C) BinarySearch.
D) OrderBy and ThenBy.
E) There is no query operator that does this.

Question 19:

Is there any difference between First() and Take(1)? (select one)

A) No, as they both return the first element from a query result.
B) No, as they both return a sequence with one single element.
C) One returns an element, while the other returns a sequence.

Question 20:

Which standard query operators are able to return an enumerable sequence without taking another sequence as source? (select one)

A) OrderBy and OrderByDescending.
B) Empty, Repeat and Range.
C) Select and Where.
D) ToList and ToArray.

Easy Basic LINQ - Answers and Explanations

Question 1:

Which of the following return a sequence of IGrouping objects? (check all that apply)

A)

```
persons.OrderBy(p => p.Name);
```

B)

```
persons.GroupBy(p => p.Name);
```

C)

```
persons.ToLookup(p => p.Name);
```

D)

```
persons.Select(p => p.Name);
```

Answer: B C

Explanation:

Select() is the only one here returning the base IEnumerable<T> sequence type.

OrderBy() returns IOrderEnumerable<T>, which inherits from IEnumerable<T> and is prepared to further connect with eventual ThenBy second-key sorting operators.

GroupBy() has several overrides, and may return an enumerable sequence of groups, as in IEnumerable<IGrouping> (I'll use a short notation for most of these generics and method signatures from now on, to keep is understandable and avoid clutter).

ToLookup() returns an ILookup generic type, which inherits from IEnumerable<IGrouping>. ILookup defines an indexer, size property, and boolean search method for data structures that map keys to IEnumerable<T> sequences of values. This means ILookup provides an enhanced enumerator: you may enumerate IGrouping groups as before, but you may also access them directly by key.

IGrouping itself maps a group identified by a unique key to a sequence of values. It's like a KeyValuePair with a multi-value sequence per key.

Remember IEnumerable, IGrouping, ILookup, and IOrderEnumerable, as these are the most important LINQ interfaces. Also remember GroupBy returns a sequence of groups, while ToLookup returns a sequence of groups that can be also accessed directly by key.

References:

https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.groupby?view=net-5.0

https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.tolookup?view=net-5.0

https://docs.microsoft.com/en-us/dotnet/api/system.linq.ilookup-2?view=net-5.0

Question 2:

What can you use in LINQ to emulate the SQL LIKE operator? (check all that apply)

A) Contains, StartsWith and EndsWith.
B) The standard query operator Exists.
C) Regular expressions.
D) IndexOf and LastIndexOf.

Answer: A C D

Explanation:

Contains(), StartsWith(), EndsWith(), IndexOf() and LastIndexOf() are all String method that can replace the need of a missing LIKE operator in LINQ.

Regular expressions can also be used for more complex patterns.

However, Exists() is not a standard query operator, it is not an Enumerable extension method. Exists() is a List method, used to check if an item can be found in the collection. It is similar to the Enumerable.Any(), but it cannot be used to check some string content.

References:

https://stackoverflow.com/questions/835790/how-to-do-sql-like-in-linq

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.exists?view=net-5.0

Question 3:

What if a Select call returns nothing? (select one)

A) The result will be null.
B) An exception will be thrown, unless you use SelectOrDefault.
C) The result is an empty IEnumerable sequence.
D) The result is an IEnumerable sequence object with the Empty property true.

Answer: C

Explanation:

Select() always returns an enumerable sequence – an IEnumerable<T> object – that can be empty, but never null. It can have no elements, one element or more. No exception is thrown when empty, and there is no SelectOrDefault (there are however FirstOrDefault, LastOrDefault, SingleOrDefault, ElementAtOrDefault).

Empty() is not a property: it is an Enumerable operator method that can return an empty sequence.

Some LINQ operators return never-null sequences (that can be empty or have just one element), others one single object (that can be null or throw an exception if not found). It is very important to recognize this pattern and the differences.

References:

https://stackoverflow.com/questions/1191919/what-does-linq-return-when-the-results-are-empty

https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.select?view=net-5.0

https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.empty?view=net-5.0