# LINQ - Restriction Operators

**Use ListGenerators.cs & Customers.xml**

1. Find all products that are out of stock.

2. Find all products that are in stock and cost more than 3.00 per unit.

3. Returns digits whose name is shorter than their value.

string[] Arr = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };

# LINQ - Element Operators

**Use ListGenerators.cs & Customers.xml**

1. Get first Product out of Stock

2. Return the first product whose Price > 1000, unless there is no match, in which case null is returned.

3. Retrieve the second number greater than 5

int[] Arr = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };.

# LINQ - Set Operators

**Use ListGenerators.cs & Customers.xml**

1. Find the unique Category names from Product List

2. Produce a Sequence containing the unique first letter from both product and customer names.

3. Create one sequence that contains the common first letter from both product and customer names.

4. Create one sequence that contains the first letters of product names that are not also first letters of customer names.

5. Create one sequence that contains the last Three Characters in each names of all customers and products, including any duplicates

# LINQ - Aggregate Operators

1. Uses Count to get the number of odd numbers in the array

int[] Arr = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

**Use ListGenerators.cs & Customers.xml**

2. Return a list of customers and how many orders each has.

3. Return a list of categories and how many products each has

4. Get the total of the numbers in an array.

int[] Arr = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

5. Get the total number of characters of all words in dictionary\_english.txt (Read dictionary\_english.txt into Array of String First).

**Use ListGenerators.cs & Customers.xml**

6. Get the total units in stock for each product category.

7. Get the length of the shortest word in dictionary\_english.txt (Read dictionary\_english.txt into Array of String First).

**Use ListGenerators.cs & Customers.xml**

8. Get the cheapest price among each category's products

9. Get the products with the cheapest price in each category (Use **Let**)

10. Get the length of the longest word in dictionary\_english.txt (Read dictionary\_english.txt into Array of String First).

**Use ListGenerators.cs & Customers.xml**

11. Get the most expensive price among each category's products.

12. Get the products with the most expensive price in each category.

13. Get the average length of the words in dictionary\_english.txt (Read dictionary\_english.txt into Array of String First).

14. Get the average price of each category's products.

# LINQ - Ordering Operators

**Use ListGenerators.cs & Customers.xml**

1. Sort a list of products by name

2. Uses a custom comparer to do a case-insensitive sort of the words in an array.

string[] Arr = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry" };

**Use ListGenerators.cs & Customers.xml**

3. Sort a list of products by units in stock from highest to lowest.

4. Sort a list of digits, first by length of their name, and then alphabetically by the name itself.

string[] Arr = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };

5. Sort first by word length and then by a case-insensitive sort of the words in an array.

string[] words = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry" };

**Use ListGenerators.cs & Customers.xml**

6. Sort a list of products, first by category, and then by unit price, from highest to lowest.

7. Sort first by word length and then by a case-insensitive descending sort of the words in an array.

string[] Arr = { "aPPLE", "AbAcUs", "bRaNcH", "BlUeBeRrY", "ClOvEr", "cHeRry" };

8. Create a list of all digits in the array whose second letter is 'i' that is reversed from the order in the original array.

string[] Arr = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };

# LINQ - Partitioning Operators

**Use ListGenerators.cs & Customers.xml**

1. Get the first 3 orders from customers in Washington

2. Get all but the first 2 orders from customers in Washington.

3. Return elements starting from the beginning of the array until a number is hit that is less than its position in the array.

int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

4. Get the elements of the array starting from the first element divisible by 3.

int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

5. Get the elements of the array starting from the first element less than its position.

int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

# LINQ - Projection Operators

**Use ListGenerators.cs & Customers.xml**

1. Return a sequence of just the names of a list of products.

2. Produce a sequence of the uppercase and lowercase versions of each word in the original array (Anonymous Types).

string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry" };

**Use ListGenerators.cs & Customers.xml**

3. Produce a sequence containing some properties of Products, including UnitPrice which is renamed to Price in the resulting type.

4. Determine if the value of ints in an array match their position in the array.

int[] Arr = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

**Result**

Number: In-place?  
5: False  
4: False  
1: False  
3: True  
9: False  
8: False  
6: True  
7: True  
2: False  
0: False

5. Returns all pairs of numbers from both arrays such that the number from numbersA is less than the number from numbersB.

int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };

int[] numbersB = { 1, 3, 5, 7, 8 };

### ****Result****

Pairs where a < b:  
0 is less than 1  
0 is less than 3  
0 is less than 5  
0 is less than 7  
0 is less than 8  
2 is less than 3  
2 is less than 5  
2 is less than 7  
2 is less than 8  
4 is less than 5  
4 is less than 7  
4 is less than 8  
5 is less than 7  
5 is less than 8  
6 is less than 7  
6 is less than 8

**Use ListGenerators.cs & Customers.xml**

6. Select all orders where the order total is less than 500.00.

7. Select all orders where the order was made in 1998 or later.

# LINQ - Quantifiers

1. Determine if any of the words in dictionary\_english.txt (Read dictionary\_english.txt into Array of String First) contain the substring 'ei'.

**Use ListGenerators.cs & Customers.xml**

2. Return a grouped a list of products only for categories that have at least one product that is out of stock.

3. Return a grouped a list of products only for categories that have all of their products in stock.

**LINQ - Grouping Operators**

**1. Use group by to partition a list of numbers by their remainder when divided by 5**

Output:

Numbers with a remainder of 0 when divided by 5:  
0  
5

10  
Numbers with a remainder of 1 when divided by 5:  
1  
6

11  
Numbers with a remainder of 2 when divided by 5:  
7  
2

12

Numbers with a remainder of 3 when divided by 5:  
3  
8  
13

Numbers with a remainder of 4 when divided by 5:  
4  
9

14

**2. Uses group by to partition a list of words by their first letter.**

**Use dictionary\_english.txt for Input**

**3. Consider this Array as an Input**

string[] Arr = { "from ", " salt", " earn ", " last ", " near ", " form " };

Use Group By with **a custom comparer** that matches words that are consists of the same Characters Together

**Result**

...  
from   
form   
...  
salt  
last   
...  
earn   
near