* You are required to calculate the value of an nth-degree polynomial. The function is called Poly and the arguments are:
  + int x = the value of x for the polynomial
  + IEnumerable<int> coeffs = the polynomial coefficients
* Example:
  + Input 2, [3,4,5]
  + Output 25 (calculated as 3\*2^2+4\*2+5)

public static int Poly(int x,

IEnumerable<int> coeffs)

{

}

public class Person

{

private string firstName, middleName, lastName;

public Person(string firstName, string middleName, string lastName)

{

this.firstName = firstName;

this.middleName = middleName;

this.lastName = lastName;

}

public IEnumerable<string> Names

{

get

{

yield return firstName;

yield return middleName;

yield return lastName;

}

}

}

var n = new Person("Sam", "Edward", "Hill");

foreach( var nm in n.Names)

{

Console.WriteLine("{0}", nm);

}

var list = new ArrayList();

list.Add(1);

list.Add(2);

list.Add(3);

// this won't work

//Console.WriteLine(list.Select(i => (int)i).Sum());

Console.WriteLine(list.Cast<int>().Sum());

// cast not valid

//Console.WriteLine(list.Cast<float>().Average());

var numbers = Enumerable.Range(1,4);

var squares = numbers.Select(x => x\*x);

Console.WriteLine(squares);

Console.WriteLine(String.Join(" ",squares));

string sentence = "This is a nice sentence";

var wordLengths =

sentence.Split().Select(w => w.Length);

Console.WriteLine(wordLengths);

Console.WriteLine(String.Join(" ",wordLengths));

// keep both word and length

var wordsWithLength =

sentence.Split().Select(w => new { Word = w, Size = w.Length});

Console.WriteLine(wordsWithLength);

foreach( var item in wordsWithLength )

{

Console.WriteLine("Word:{0},Length:{1}", item.Word, item.Size);

}

// get all the words

var sequences = new[]{ "red,green,blue", "orange", "white,pink" };

**// use Select**

var allWords = sequences.SelectMany(s => s.Split(',')); // select

Console.WriteLine(allWords);

Console.WriteLine(String.Join(" ",allWords));

// select each pair (cross product) from two collections

string [] objects = { "house", "car", "bicycle" };

string [] colors = { "red", "green", "gray" };

var pairs = colors.SelectMany(z => objects, (c, o) => $"{c} {o}");

var pairs = colors.SelectMany(z => objects, (c, o) => c + " " + o );

Console.WriteLine(pairs);

Console.WriteLine(String.Join(" ", pairs));

var numbers = Enumerable.Range(1,10);

var evenNumbers = numbers.Where(n => n%2 == 0);

Console.WriteLine(evenNumbers);

// combine projection and filtering

var oddSquares = numbers.Select(x => x\*x).Where(y => y%2 != 0);

Console.WriteLine(oddSquares);

// filter by type

object [] values = { 1, 2.5, 3, 4.56 };

var wholeNumbers = values.OfType<int>(); // try float or double

Console.WriteLine(wholeNumbers);

void Main()

{

var rand = new Random();

var randomValues = Enumerable.Range(1,10).Select(\_ => rand.Next(10) - 5);

var csvString = new Func<IEnumerable<int>,string>(values =>

{

return string.Join(",", values.Select(v => v.ToString()).ToArray());

});

// different set of values each time

Console.WriteLine(csvString(randomValues));

Console.WriteLine(csvString(randomValues.OrderBy(x => x)));

Console.WriteLine(csvString(randomValues.OrderByDescending(x => x)));

class Person

{

string name = "";

int age = 0;

public Person()

{

}

public Person( string name, int age )

{

this.name = name;

this.age = age;

}

public string Name

{

get { return name; }

set { name = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

}

var people = new List<Person>

{

new Person{ Name = "Adam", Age = 36 },

new Person { Name = "Boris", Age = 18 },

new Person { Name = "Claire", Age = 36 },

new Person { Name = "Adam", Age = 20 },

new Person { Name = "Jack", Age = 20 }

};

foreach (Person p in people)

{

Console.WriteLine("{0} {1}",p.Name,p.Age);

}

foreach (Person p in people.OrderBy(p => p.Age))

{

Console.WriteLine("{0} {1}",p.Name,p.Age);

}

foreach (Person p in people.OrderBy(p => p.Age).ThenByDescending(p => p.Name))

{

Console.WriteLine("{0} {1}",p.Name,p.Age);

}

// reverse a string

string s = "This is a test";

Console.WriteLine(new string(s.Reverse().ToArray()));

}

// Define other methods and classes here

class Person

{

public string Name;

public int Age;

}

void Main()

{

var people = new List<Person>{

new Person { Name = "Adam", Age = 20 },

new Person{ Name = "Adam", Age = 36 },

new Person { Name = "Boris", Age = 18 },

new Person { Name = "Claire", Age = 36 },

new Person { Name = "Adam", Age = 20 }, // dup

new Person { Name = "Jack", Age = 20 }

};

IEnumerable<IGrouping<string, Person>> byName = people.GroupBy(p => p.Name);

foreach( var bn in byName )

{

Console.WriteLine("--"+bn.Key+"--");

foreach( var n in bn )

{

Console.WriteLine("{0} {1}", n.Name, n.Age );

}

}

var byAgeNames = people.GroupBy(p => p.Age, p => p.Name);

foreach (var item in byAgeNames)

{

Console.WriteLine("These people are {0} years old", item.Key);

foreach (var name in item)

{

Console.WriteLine(name);

}

}

class Person

{

public string Name;

public int Age;

}

var word1 = "helloooo";

var word2 = "help";

// distinct letters in helloooo

Console.WriteLine(String.Join("",word1.Distinct()));

// letters in both word1 and word2

var lettersInBoth = word1.Intersect(word2);

Console.WriteLine(String.Join("",lettersInBoth));

// letters in all words

Console.WriteLine(String.Join(""word1.Union(word2)));

// letters in word1, but not in word1

Console.WriteLine(word1.Except(word2));

public class Exercise2

{

public static IEnumerable<int> merge(IEnumerable<int> a, IEnumerable<int> b)

{

var uniqueInOne = a.Except(b);

var uniqueInTwo = b.Except(a);

var combined = uniqueInOne.Union(uniqueInTwo);

return combined;

}

}

var ans = Exercise2.merge(

new int[] { 1, 2, 3, 4, 5 },

new int[] { 1, 3, 5, 7, 9 });

**Quantifier Operations**

int [] numbers = { 1, 2, 3, 4, 5};

Console.WriteLine("Are all numbers > 0? " +

numbers.All(x => x > 0));

Console.WriteLine("Are all numbers odd? " +

numbers.All(x => x%2 == 1));

// once the iterator finds a match, it doesn't check other elements

Console.WriteLine("Any number less than two? " + numbers.Any(x => x < 2));

// also, Any() is a way to check whether a collection is empty

Console.WriteLine(new int[]{/\*42\*/}.Any());

Console.WriteLine("Contains 5? " + numbers.Contains(5)); // not a lambda here!

Console.WriteLine("Number of odd elements: " + numbers.Count(x => x % 2 == 1));

Console.WriteLine("Total no. of elements: " + numbers.Count()); // beware!

**Partition Data**

var numbers = new[]{ 3, 3, 2, 2, 1, 1, 2, 2, 3, 3 };

Console.WriteLine(numbers.Skip(2).Take(6));

Console.WriteLine(new int[]{}.Take(2));

Console.WriteLine(numbers.SkipWhile(i => i == 3));

Console.WriteLine(numbers.TakeWhile(i => i > 1));

**Exercise 3**

public static int LengthOfPositive(IEnumerable<int> input)

{

var skipStartingNegatives = input.SkipWhile(i => i < 0);

var getNextPositives = skipStartingNegatives.TakeWhile(i => i > 0);

return getNextPositives.Count();

}

**int cnt = Exercise3.LengthOfPositive(new int[] { -3, -1, 3, 7, 1, -3, 7 });**

**Equality**

var arr1 = new[]{1,2,3};

var arr2 = new[]{1,2,3};

// LINQ

Console.WriteLine(arr1.SequenceEqual(arr2));

var list1 = new List<int>{1,2,3};

Console.WriteLine(arr1.SequenceEqual(list1));

**Element Operations**

// first: returns first element satisfying a predicate or throws

var numbers = new List<int>{1,2,3};

Console.WriteLine(numbers.First());

Console.WriteLine(numbers.First(x => x > 2)); // try 10

Console.WriteLine(numbers.FirstOrDefault(x => x > 10)); // string - null

// same for last value

Console.WriteLine(numbers.Last());

Console.WriteLine(numbers.Last(x => x < 3));

// single: ensures that there's only one value, otherwise throws

// throws because non-singular

//Console.WriteLine(numbers.Single());

// also throws

//Console.WriteLine(numbers.SingleOrDefault());

// doesn't throw only if sequence is empty

Console.WriteLine("Empty array: " + new int[]{}.SingleOrDefault());

Console.WriteLine("Item at position 1: " + numbers.ElementAt(1));

Console.WriteLine("Item at position 4: " + numbers.ElementAtOrDefault(4));

**Concatenation**

var integralTypes = new[]{ typeof(int), typeof(short) };

var fpTypes = new[]{ typeof(float), typeof(double) };

Console.WriteLine(

integralTypes.Concat(fpTypes));

void Main()

{

var integralTypes = new[]{ typeof(int), typeof(short) };

var fpTypes = new[]{ typeof(float), typeof(double) };

Console.WriteLine(

integralTypes.Concat(fpTypes)

.Prepend(typeof(byte))

);

}

// Define other methods and classes here

static class ExtensionMethods

{

public static IEnumerable<T> Prepend<T>(

this IEnumerable<T> values, T value)

{

yield return value;

foreach (var item in values) {

yield return item;

}

}

}

**Aggregation Operations**

var numbers = Enumerable.Range(1,10);

Console.WriteLine("We have " + numbers.Count() + " elements");

//Console.WriteLine(numbers);

// 1 2 3 4 5 ...

// 1 2 -> 3

// 3 3 -> 6

Console.WriteLine("Sum = " +

numbers.Sum());

// 1 1 -> 1

// 1 2 -> 2

// 2 3 -> 6

Console.WriteLine("Average = " +

numbers.Average());

var words = new[] { "one", "two", "three" };

Console.WriteLine(words.Aggregate("hello", (p,x) => p + "," + x));

// Rectangle.Union(r1, r2)

var rectangles = new[] {

new Rectangle(0,0,20,20),

new Rectangle(20,20,60,60),

new Rectangle(80,80,20,20)

};

Console.WriteLine(rectangles.Aggregate(Rectangle.Union));

**Exercise4**

public static int Poly(int x, IEnumerable<int> coeffs)

{

int seed = 0;

var su = coeffs.Reverse().Aggregate(seed, (p, q) => p + q \* (int)Math.Pow(x,seed++) );

return su;

}