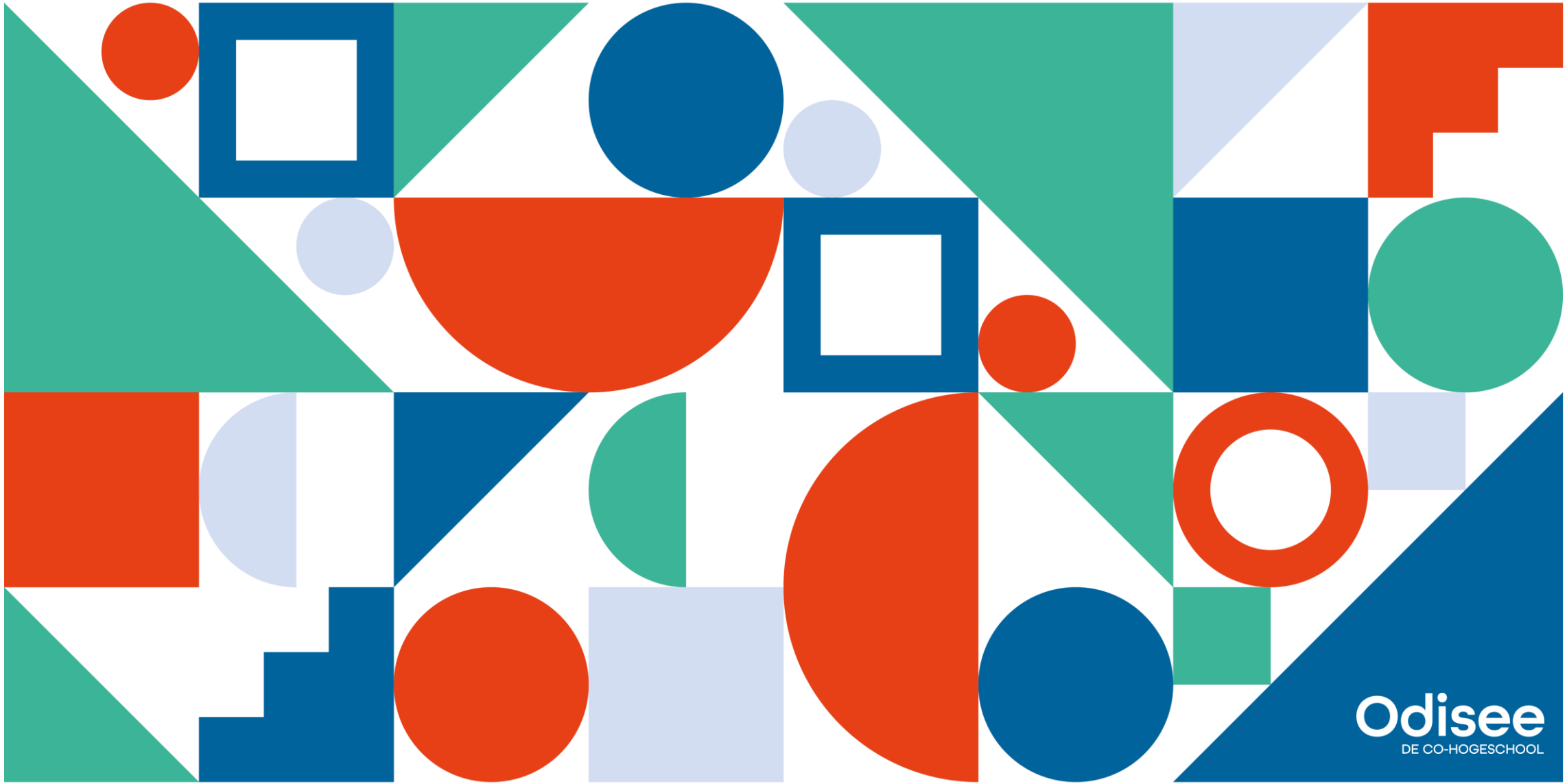

Odisee
DE CO-HOGESCHOOL

# LINQ

# Opdracht

- Schrijf een applicatie die op basis van een lijst van getallen het gemiddelde terug geeft

# Opdracht - Oplossing

```
0 references
static void Main(string[] args)
{
    List<int> numbers = new List<int> { 24, 96, 84, 51, 52 };

    int sum = 0;
    foreach(var number in numbers)
    {
        sum += number;
    }

    double average = sum / numbers.Count;

    Console.WriteLine(average);
    Console.ReadKey();

}
```

# Opdracht – Oplossing met LINQ

```
double average2 = numbers.Average();

Console.WriteLine(average2);

Console.ReadKey();
```

# Wat is LINQ

# Wat is LINQ?

- Language Integrated Query
- Uniforme syntax voor het opvragen en manipuleren van gegevens afkomstig van verscheidene databronnen
  - SQL-database, XML-bestand, .Net Collections zoals array's, Lists, Dictionaries, …)

- Twee manieren
  - Query Syntax
  - Method Syntax

```
IEnumerable<int> result = from g in numbers where (g % 2) == 0 select g;

IEnumerable<int> result2 = numbers.Where(g => g % 2 == 0);
```

# IEnumerable

- Interface die beschikbaar is in het .NET Framework (System.Collections namespace)
- De definitie van de IEnumerable interface ziet er als volgt uit:

```
public interface IEnumerable<out T> : IEnumerable
{
    new IEnumerator<T> GetEnumerator();
}
```

- Interface bevat een aantal methoden waarmee de elementen van een collective overlopen kunnen worden (“itereren”)
  - Current, MoveNext(), Reset()

# IEnumerable

- foreach: Collectie moet de IEnumerable interface implementeren
  - Alle bestaande collecties doen dit

```
public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>, IEnumerable, ILis
{
```

# IEnumerable

- Extension methods
  - Om eenvoudiger te werken met collections
  - Gedefinieerd in een andere klasse maar kan gebruikt worden alsof ze deel uitmaakt van de klasse zelf
  - Filteren, Sorteren, Berekeningen, …
  - Dit werkt voor Lists en Arrays ook.

# Extension Methods

- Sum()
- Min()
- Max()
- Average()

```
List<int> numbers2 = new List<int> { 24, 96, 84, 51, 52 };

double average3 = numbers2.Average();
double max = numbers2.Max();
double min = numbers2.Min();
double sum2 = numbers2.Sum();
```

- Dit werkt enkel bij collecties van getallen: Max() van strings?

# LINQ

- Stel we hebben een klasse Student

```
1 reference
class Student
{
    1 reference
    public string FirstName { get; set; }
    1 reference
    public string LastName { get; set; }
    1 reference
    public int Age { get; set; }

    0 references
    public Student(string firstname, string lastname, int age)
    {
        FirstName = firstname;
        LastName = lastname;
        Age = age;
    }

}
```

# LINQ

- En een lijst met studenten

```
// part students
List<Student> students = new List<Student>()
{
    new Student("Roma", "Kuvalis", 18),
    new Student("Margaret", "Rath", 19),
    new Student("Tina", "Marquardt", 20),
    new Student("Destiny", "Bechtelar", 21),
    new Student("Asia", "Gibson", 19)
};
```

- Wat is de gemiddelde leeftijd?

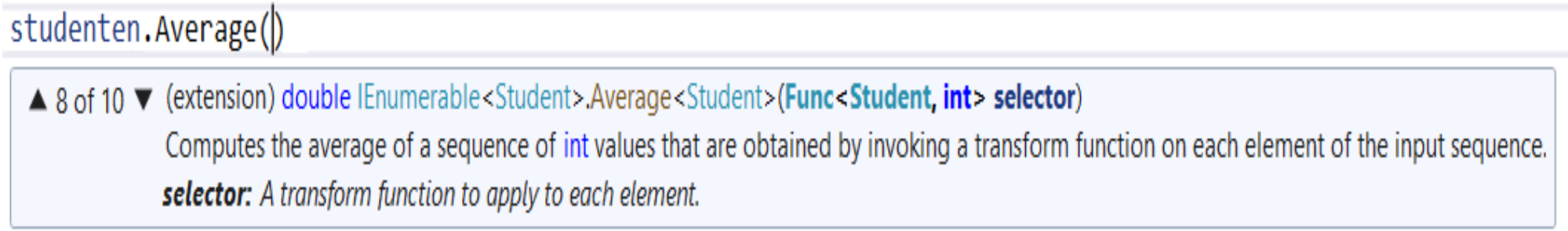
# LINQ

- Oplossing:

```
// part students
List<Student> students = new List<Student>()
{
    new Student("Roma", "Kuvalis", 18),
    new Student("Margaret", "Rath", 19),
    new Student("Tina", "Marquardt", 20),
    new Student("Destiny", "Bechtelar", 21),
    new Student("Asia", "Gibson", 19)
};

double sumAge = 0;
foreach (var student in students)
{
    sumAge += student.Age;
}

double averageAge = sumAge / students.Count;

Console.WriteLine(averageAge);

Console.ReadKey();
```

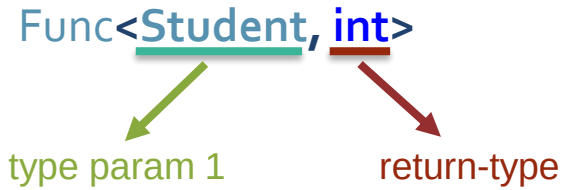
# LINQ

- De extension method Average() van IEnumerable<T> heeft een overload om een Student-object om te zetten naar een getal




- Func<> is een type (delegate) dat kan verwijzen naar een methode
- De types dusen de < en > symbolen bepalen de signatuur van de methode
  - Laatste is het return type

# LINQ




- Welke methode stelt dit voor?

```
0 references
static int GetAge(Student student)
{
    return student.Age;
}
```

# LINQ

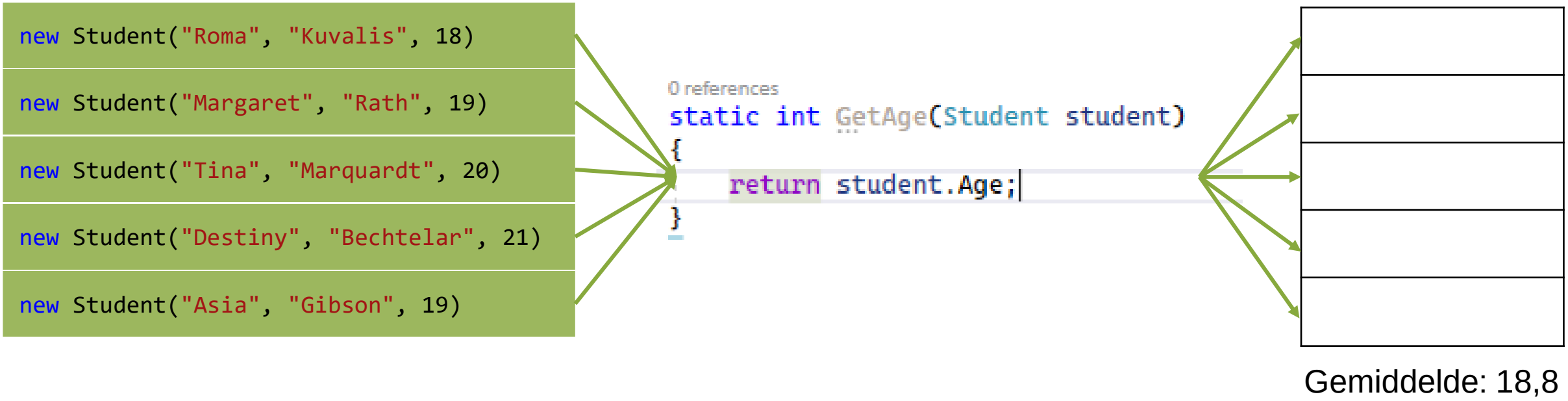
- De methode GetAge() kan dan meegegeven worden aan de Average() functie

```
averageAge = students.Average(GetAge);
```

- Hierdoor gebeurt het volgende
  - Average itereert over de hele collectie
  - Voor elk object in de collectie wordt de GetAge opgeroepen
  - Het gemiddelde van de resultaten van de GetAge-functie wordt berekend.

# LINQ

- De werking van Average(..) visueel voorgesteld:



# LINQ

- Kan je alle functies gebruiken?
  - Er moet een voorgedefnieerde Func<…,…> type zijn
  - Zou bestaan er veel (Tot 16 argumenten en 1 return waarde)

- Voorbeeld: Func<float, int, double, string, bool>:

```
0 references
static bool Method(float first, int second, double third, string fourth)
{
    return false;
}
```

# LINQ – Lambda expressions

- Nadeel van de Func<..,..> structuur is dat het omslachtig is om steeds de nieuwe methode te maken.
- Oplossing: Lambda expressions
  - Concept wordt in veel programmeertalen gebruikt
  - Stelt een naamloze (anonieme) methode voor die gebruikt kan worden als argument bij de aanroep van een andere methode

  - (input-parameters) => { statements }

# LINQ – Lambda's

- Voorbeeld:
  - Func<int, int, int> multiply = (int a, int b) => { return a * b; }
- Mogelijke vereenvoudigen
  - Geen verwarring mogelijk bij de types van de parameters => mag weggelaten worden
    - Func<int, int, int> multiply = (a, b) => { return a * b; }

# LINQ – Lambda's

- Mogelijke vereenvoudigen
  - Geen verwarring mogelijk bij de types van de parameters => mag weggelaten worden
  - Indien er slechts 1 statement is => accolades en return mogen weggelaten worden
    - Func< int, int , int> multiply = (a, b) => a * b;

# LINQ – Lambda's

- Mogelijke vereenvoudigen
  - Geen verwarring mogelijk bij de types van de parameters => mag weggelaten worden
  - Indien er slechts 1 statement is => accolades en return mogen weggelaten worden
  - Indien er slechts 1 argument is, mogen de haakjes weg
    - Func<int, int> oneLarger = (a) => a + 1;
    - Func<int, int> oneLarger = a => a + 1;

# Oefening Lambda’s

- Zet nu de GetAge methode om naar een lambda expression

# Oefening Lambda’s

- Zet nu de GetAge methode om naar een lambda expression

```
averageAge = students.Average(student => student.Age);
```

# LINQ

- Dit gaat ook voor andere berekeningen:
  - Min()

```
averageAge = students.Min(student => student.Age);
```

  - Max()

```
averageAge = students.Max(student => student.Age);
```

  - Sum()

```
averageAge = students.Sum(student => student.Age);
```

# LINQ - Filteren

- Where()
  - Filteren op basis van een voorwaarde / predicaat
  - Methode dat een Boolean returned of het object in het resultaat moet zitten of niet
  - Returned een IEnumerable<T> met T dezelfde klasse als het origineel

```
List<int> numbers3 = new List<int> { 24, 96, 84, 51, 52 };

IEnumerable<int> even = numbers3.Where(x => x % 2 == 0);

foreach(var x in even)
{
    Console.WriteLine(x);
}
```

# Oefening - Where

- Voeg ten eerste een geslacht toe aan de studenten
- Pas de lijst met studenten aan zodat sommige studenten mannelijk zijn en sommige vrouwelijk
- Schrijf een lambda expressie om deze studenten te selecteren
  - Leeftijd is groter dan 18 jaar
  - En hun geslacht is mannelijk

# Oplossing

```
List<Student> students = new List<Student>()
{
    new Student("Roma", "Kuvalis", 18, Sex.MALE),
    new Student("Margaret", "Rath", 19, Sex.FEMALE),
    new Student("Tina", "Marquardt", 20, Sex.FEMALE),
    new Student("Destiny", "Bechtelar", 21, Sex.FEMALE),
    new Student("Asia", "Gibson", 19, Sex.MALE)
};

IEnumerable<Student> male_students = students.Where(
    student => student.Age > 18 && student.Sex == Sex.MALE
);

foreach (var s in male_students)
{
    Console.WriteLine(s.FirstName + " " + s.LastName);
}
```

# LINQ - Filteren

- Select()
  - Zet een collectie om naar een andere collective (Transformatie)
  - Return is een IEnumerable
  - Type is niet noodzakelijk hetzelfde

```
// select - numbers
List<int> numbers4 = new List<int> { 24, 96, 84, 51, 52 };

IEnumerable<int> squares = numbers3.Select(x => x * x);

foreach (var x in even)
{
    Console.WriteLine(x);
}
```

# Oefening - Select

- Selecteer de volledige naam (voor en achternaam) van de studenten

# Oplossing

```
// select - students
IEnumerable<string> names = students.Select(
    student => student.FirstName + " " + student.LastName
);

foreach (var n in names)
{
    Console.WriteLine(n);
}
```

# Method Chaining

- Veel LINQ-methodes hebben een IEnumerable als return-type
  - Maakt het mogelijk om opnieuw een LINQ methode aan te roepen
  - Dit wordt method chaining genoemd

```
// methodchaining - students
IEnumerable<string> firstnames = students
    .Select(student => student.FirstName)
    .Where(name => name.Length > 4);

firstnames = from s in students
             where s.FirstName.Length > 4
             select s.FirstName;
```

# Sorteren

- OrderBy / OrderByDescending
  - Sorteert items van klein naar groot (of van groot naar klein voor descending)
  - Agument is lambda-expressie om objecten om te zetten naar een getal om op te sorteren

```
IEnumerable<int> gesorteert = numbers3.OrderBy(number => number);
IEnumerable<Student> students_sorted =
    students.OrderByDescending(student => student.Age);
```

# Zoeken

- First()
  - Returned het eerste element in een collective
  - Kan een functie als argument hebben als predicaat/conditie

```
// searching - students
Student first = students.First(
    student => student.Sex == Sex.FEMALE
);
```

# LINQ

- Volledige documentatie met alle mogelijke functies vind je hier:
  - https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable?view=netframework-4.8#methods