



# Programming 1 (C#)

Week 4

# Programma periode 1.1 (Programming 1)

01 (wk 36)	Introduction C# / Visual Studio 2022 (Community), basic problem solving
02 (wk 37)	branching, methods
03 (wk 38)	loops, basic version control setup
04 (wk 39)	classes, enums, arrays
05 (wk 40)	public/private, fields/properties, values & references
06 (wk 41)	inheritance version control
07 (wk 42)	Repetition / practice exam
08 (wk-43)	<i>no classes</i>
09 (wk-44)	exam ( <i>practical, computer</i> )
10 (wk-45)	-

# Classes

- Tot nu toe waren onze programma's als volgt opgebouwd:
  - Method1
  - Method2
  - Main code, using Method1 and Method2
- Naarmate de complexiteit van programma's toeneemt, is een dergelijk model onhoudbaar.
  - Om de complexiteit verder te verminderen, zullen we objectgeoriënteerde programmeertechnieken gebruiken.
- In objectgeoriënteerd programmeren is een klasse een uitbreidbaar programma-code-sjabloon voor het maken van objecten, die initiële waarden levert voor de toestand (member variables) en implementaties van gedrag (member methods).
- Wanneer een object wordt gemaakt door een constructor van de klasse, wordt het resulterende object een instantie van de klasse genoemd.

## Classes – example

- Een klasse bevat 'fields' en 'methods' die erbij horen.
- Ieder 'Employee' (object) heeft een voornaam, achternaam, afdeling en salaris, en we kunnen het salaris verhogen.
- Het is een good practice om een klasse te definiëren in een apart bronbestand.

```
class Employee
{
    // fields
    string firstName, lastName;
    string department;
    float salary;

    // method
    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

# Classes and initializing objects

- Hier worden twee objecten gemaakt:
  - een object dat Kevin voorstelt
  - een object dat Maria voorstelt
- Let op de verschillen, de manier waarop de velden worden ingevuld.
- Alleen het salaris van Kevin wordt verhoogd (van 2500 naar 2600)

```
// one way to create and fill an object
Employee employee1 = new Employee();
employee1.firstName = "Kevin";
employee1.lastName = "Armstrong";
employee1.department = "Human Resources";
employee1.salary = 2500;

// another way to create and fill an object
Employee employee2 = new Employee()
{
    firstName = "Mary",
    lastName = "Clark",
    department = "Software Development",
    salary = 2850
};

// increase salary of employee1
employee1.IncreaseSalary(100);
```

# Constructors

- Een constructor is een methode die (automatisch) wordt aangeroepen wanneer een object wordt gemaakt
  - Het wordt gebruikt om een nieuw object te initialiseren
  - Naam van deze methode = naam van de klasse

```
class Employee
{
    // fields
    string firstName, lastName;
    string department;
    float salary;

    // constructor
    public Employee()
    {
        firstName = "";
        lastName = "";
    }

    // method
    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

# Constructors – overloading

- Een klasse kan meerdere constructors bevatten (met verschillende parameters)

```
class Employee
{
    string firstName, lastName;
    string department;
    float salary;

    // constructor 1
    public Employee()
    {
        firstName = "";
        lastName = "";
    }

    // constructor 2
    public Employee(string firstName, string lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

# Creating instances

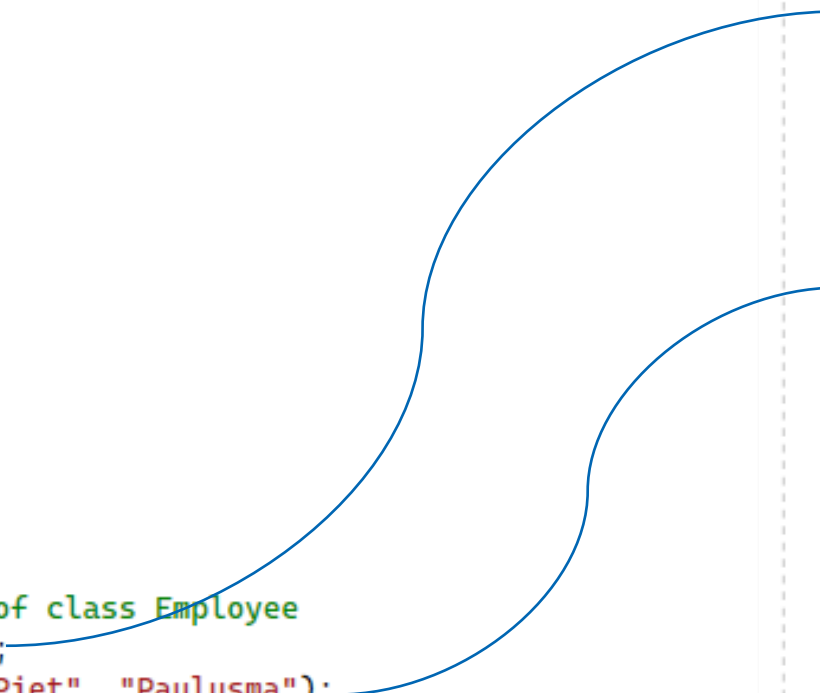
```
// create instances (objects) of class Employee
Employee emp1 = new Employee();
Employee emp2 = new Employee("Piet", "Paulusma");
```

```
class Employee
{
    // fields
    public string firstName, lastName;
    public string department;
    public float salary;

    public Employee()
    {
        // default salary is 0
        salary = 0;
    }

    public Employee(string fn, string ln)
    {
        firstName = fn;
        lastName = ln;
        salary = 0;
    }

    // method
    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```





# Default constructor

- Een constructor zonder parameters wordt de default constructor genoemd.
- Als je geen constructor opgeeft, is de default constructor beschikbaar.

## Visual Studio 2022 (Community) - demo



*(how to create a new class in a separate file and use)*

# Enumerations

Kijk eens naar de volgende methode:

```
public bool IsWeekend(int dayOfWeek)
{
    return ((dayOfWeek == 0) || (dayOfWeek == 6));
}
```

Een ander voorbeeld:

```
// impossible... but the compiler will not complain!
bool result = IsWeekend(100);

// valid input, but which day do we mean?
result = IsWeekend(1);
```

## Enumerations

- Een enumeration is een verzameling van (gerelateerde) named constants, e.g.:
  - enum Level: Low, Medium, High
  - enum StudentType: FulltimeStudent, ParttimeStudent
- Enumerations zijn "strongly typed constants"  
*(om een enumeratie om te zetten moeten we expliciet een conversie gebruiken; dit voorkomt dat we programmeerfouten maken!)*

# Enumerations

```
void Start()
{
    bool result1 = IsWeekend(DayOfWeek.Monday); // valid input
    bool result2 = IsWeekend(1);                 // invalid input
}

bool IsWeekend(DayOfWeek day)
{
    return ((day == DayOfWeek.Saturday) || (day == DayOfWeek.Sunday));
}
```

```
public enum DayOfWeek
{
    Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
}
```

- (leesbaarder EN compiler zal klagen als het niet correct wordt gebruikt)

# Enumerations

Alle opties van een Enumeratie hebben een geheel getal, beginnend bij 0.

Dit kan gewijzigd worden:

```
public enum DayOfWeek
{
    Sunday, Monday, Tuesday, Wednesday = 8, Thursday, Friday, Saturday
}
```

Nu heeft zondag nog steeds de waarde 0, maar woensdag=8, donderdag=9, etc.

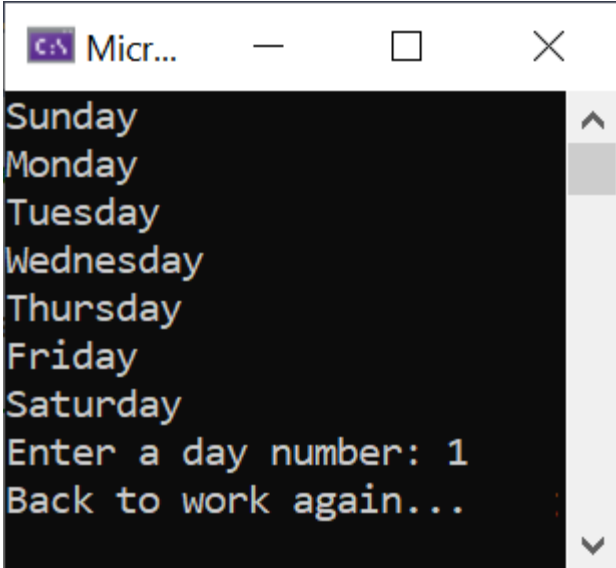
## Enumerations – example

```
void Start()
{
    // loop through all days
    for (DayOfWeek d = DayOfWeek.Sunday; d <= DayOfWeek.Saturday; d++)
        Console.WriteLine(d);

    // read day number (e.g. "1")
    Console.Write("Enter a day number: ");
    string input = Console.ReadLine();
    DayOfWeek day = (DayOfWeek)int.Parse(input);

    switch (day)
    {
        case DayOfWeek.Friday:
            Console.WriteLine("Almost weekend!");
            break;
        case DayOfWeek.Monday:
            Console.WriteLine("Back to work again...");
            break;
    }
}
```

```
public enum DayOfWeek
{
    Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
}
```



```
Micr...
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Enter a day number: 1
Back to work again...
```

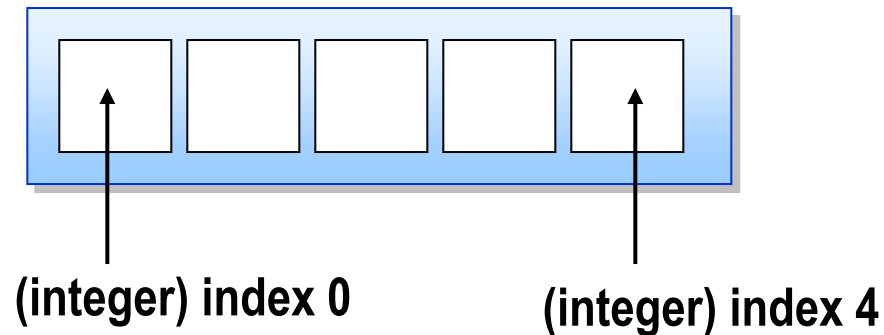
# Arrays

- Variabelen van het type int, double, float, bool kunnen slechts één waarde bevatten/opslaan
- Arrays kunnen meerdere waarden (van hetzelfde type) bevatten/opslaan; Volgorde van waarden
- Array-waarden worden vaak verwerkt via een lus (iteratie)



# Arrays

- Een array bevat een opeenvolging van elementen die afzonderlijk kunnen worden geadresseerd met een ('zero-based') index
- Het eerste element van een array heeft index 0; Het laatste element van een array heeft een index count-1



# Declaring an array

- Declaratie van een array in C# bestaat uit 3 delen:
  1. The element type of the array
  2. The rank of the array (*number of dimensions*)
  3. The name of the array variable

```
// declaration of a variable  
int number;
```

```
// declaration of an array  
int[] numbers;
```

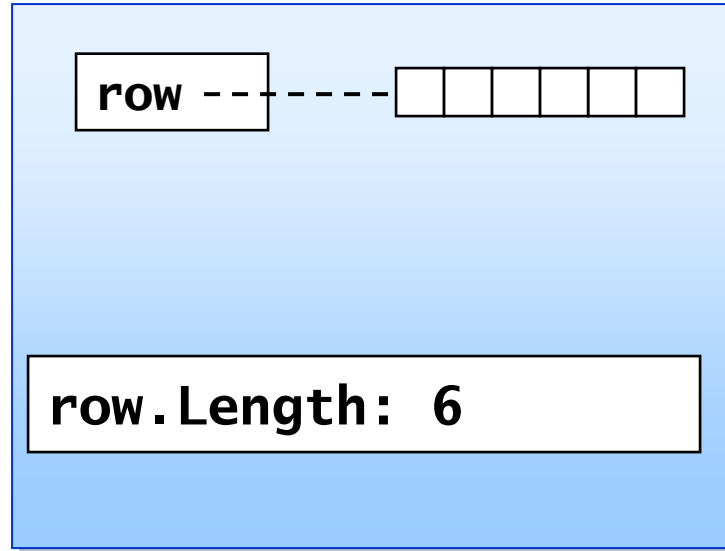
The **name** of the array variable

The **rank** of the array

The element **type** of the array

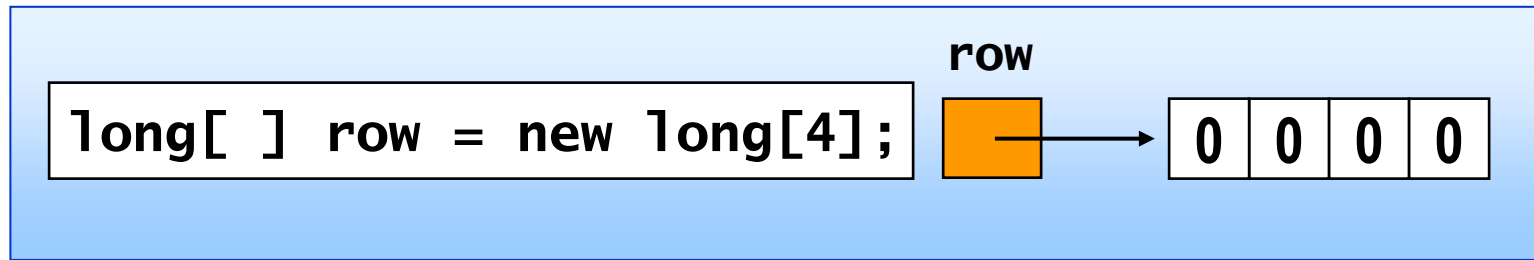
# Array 'bounds' (borders)

- Het onjuist adresseren van een array (slechte index) veroorzaakt een 'IndexOutOfRangeException'
- Gebruik de property Length



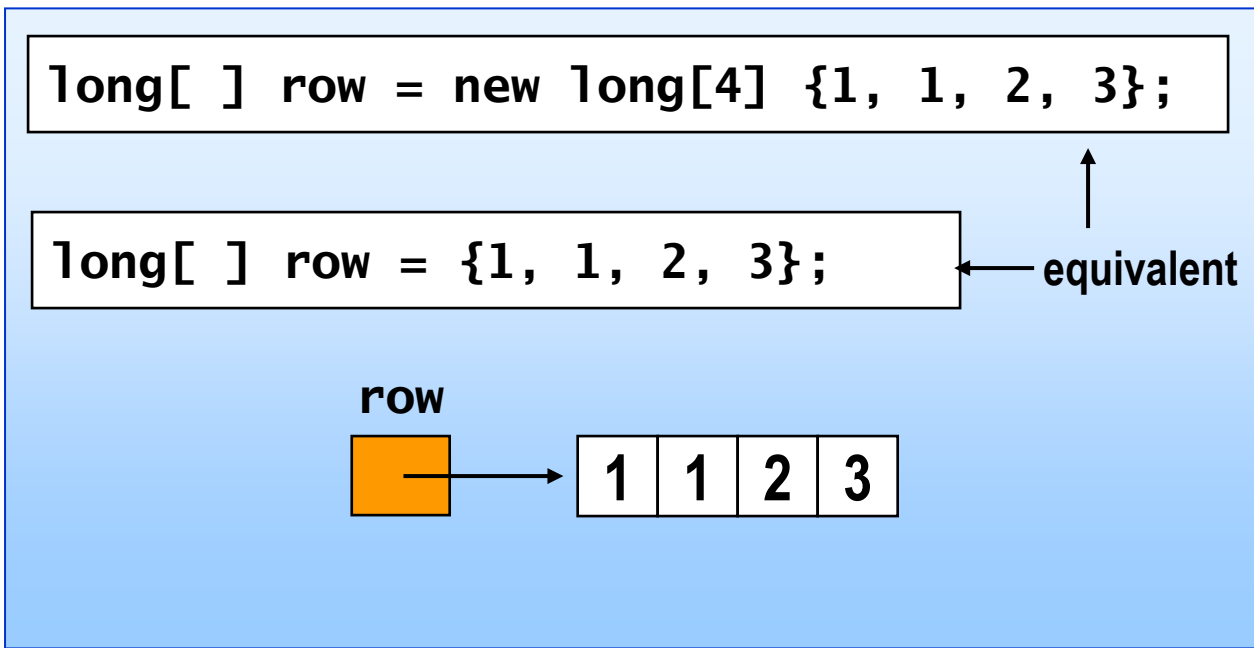
# Creating arrays

- Het declareren van een array variabele vult de array elementen niet!
- Gebruik *new* om een array te maken (ruimte maken voor alle elementen)
- De standaardwaarde van array elementen is 0



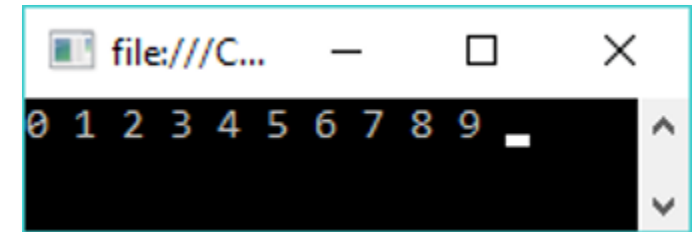
# Initializing arrays

- Bij het maken van een array kunnen elementwaarden direct worden opgegeven



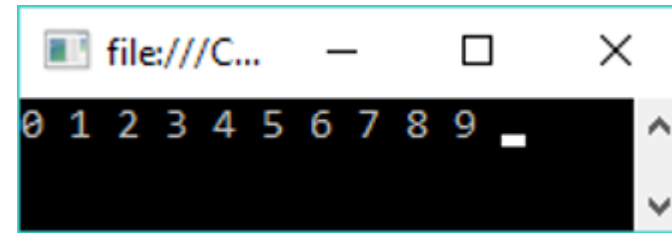
# Array – an example

```
void Start()  
{  
    int[] numbers = new int[10];  
  
    // fill array with numbers  
    for (int i = 0; i < numbers.Length; i++)  
    {  
        numbers[i] = i;  
    }  
  
    // display array values  
    for (int i = 0; i < numbers.Length; i++)  
    {  
        Console.Write($"{numbers[i]} ");  
    }  
}
```



# Array – foreach

```
void Start()  
{  
    int[] numbers = new int[10];  
  
    // fill array with numbers  
    for (int i = 0; i < numbers.Length; i++)  
    {  
        numbers[i] = i;  
    }  
  
    // display array values  
    foreach (int number in numbers)  
    {  
        Console.Write($"{numbers} ");  
    }  
}
```



# Dynamic array length

- De grootte van een array kan tijdens runtime worden ingesteld

```
void Start()
{
    // length of array (4) known when compiling
    long[] row1 = new long[4];

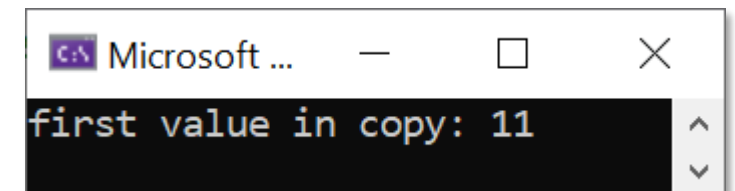
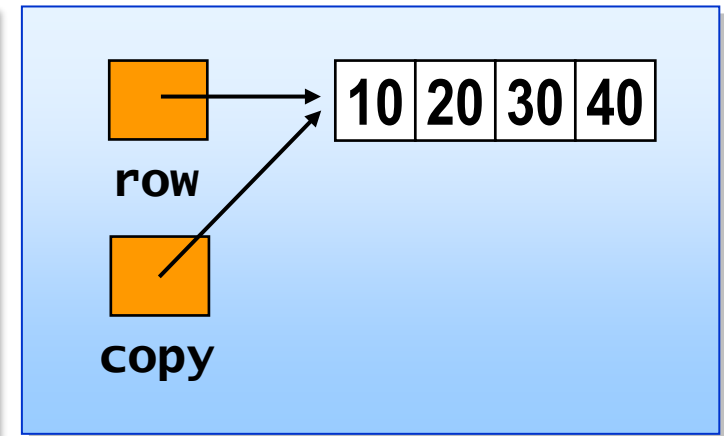
    // length of array not known when compiling (run-time value)
    string input = Console.ReadLine();
    int size = int.Parse(input);
    long[] row2 = new long[size];
}
```



# Copying arrays

- Bij het kopiëren van een array wordt alleen de referentie gekopieerd, niet de array zelf
- 2 array variabelen kunnen naar dezelfde array verwijzen

```
void Start()  
{  
    long[] row = new long[4] { 10, 20, 30, 40 };  
    long[] copy = row;  
  
    // changing first value in row  
    row[0]++;  
  
    // displaying first value in copy array  
    Console.WriteLine($"first value in copy: {copy[0]}");  
}
```



A screenshot of a Windows console window titled 'Microsoft ...'. The console displays the output of the program: 'first value in copy: 11'. The window has standard Windows window controls (minimize, maximize, close) at the top.

# Arrays – exercise 1

- Read the profit of each month (12x). Calculate the average profit and display all profits higher than the average.

# Arrays – exercise 1

```
profit ← array[12]

total = 0
for i = 0 to profit.Length - 1
    read profit[i]
    total = total + profit[i]
average = total / profit.Length

for i = 0 to profit.Length - 1
    if profit[i] > average
        display profit[i]
```

# Arrays – exercise 1

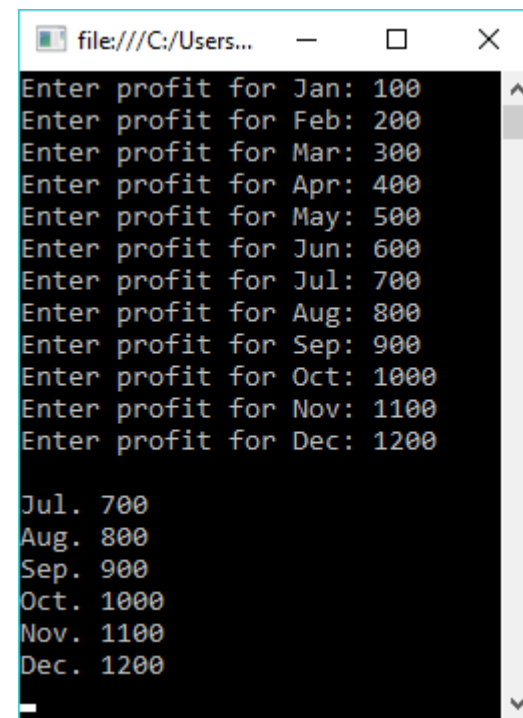
```
const int NrOfMonths = 12;

string[] monthNames = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
int[] profit = new int[NrOfMonths];

int total = 0;
for (int i = 0; i < profit.Length; i++)
{
    Console.Write("Enter profit for " + monthNames[i] + ": ");
    profit[i] = int.Parse(Console.ReadLine());
    total += profit[i];
}

double average = (double)total / profit.Length;

for (int i = 0; i < profit.Length; i++)
{
    if (profit[i] > average)
    {
        string month = monthNames[i];
        Console.WriteLine($"{month}. {profit[i]}");
    }
}
```



```
file:///C:/Users...
Enter profit for Jan: 100
Enter profit for Feb: 200
Enter profit for Mar: 300
Enter profit for Apr: 400
Enter profit for May: 500
Enter profit for Jun: 600
Enter profit for Jul: 700
Enter profit for Aug: 800
Enter profit for Sep: 900
Enter profit for Oct: 1000
Enter profit for Nov: 1100
Enter profit for Dec: 1200

Jul. 700
Aug. 800
Sep. 900
Oct. 1000
Nov. 1100
Dec. 1200
```

# Arrays – exercise 2

- Determine if a value is present in an array with 6 values

## Arrays – exercise 2

```
method GenerateNumbers(nrOfItems)
    numbers ← array[nrOfItems] ← random number
    return numbers

method FindNumberInArray(numbers, searchNumber)
    for i = 0 to numbers.Length
        if numbers[i] = searchNumber
            return index
    return -1

method Start
    numbers = GenerateNumbers(6)
    read searchValue
    foundIndex = FindNumberInArray(numbers, searchValue)
    display foundIndex
```

## Arrays – exercise 2

```
int[] GenerateNumbers(int nrOfItems)
{
    int[] numbers = new int[nrOfItems];

    Random rnd = new Random();
    for (int i = 0; i < numbers.Length; i++)
        numbers[i] = rnd.Next(100);

    return numbers;
}
```

```
int FindNumberInArray(int[] numbers, int number)
{
    for (int i = 0; i < numbers.Length; i++)
    {
        if (numbers[i] == number)
            return i;
    }

    return -1;
}
```

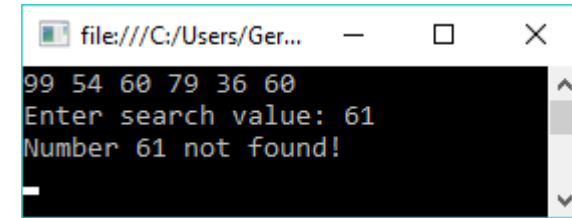
# Arrays – exercise 2

```
const int NumberOfItems = 6;

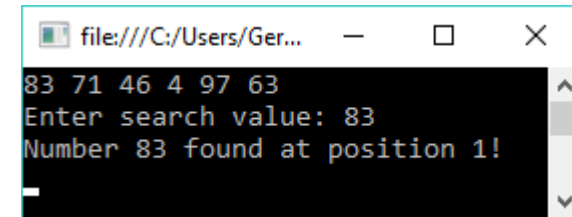
void Start()
{
    int[] numbers = GenerateNumbers(NumberOfItems);

    Console.WriteLine("Enter search value: ");
    int number = int.Parse(Console.ReadLine());

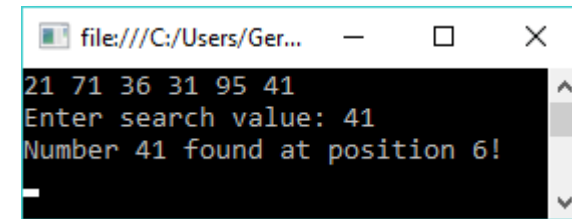
    int index = FindNumberInArray(numbers, number);
    if (index != -1)
        Console.WriteLine($"Number {number} at position {index + 1}!");
    else
        Console.WriteLine($"Number of {number} not found!");
}
```



```
file:///C:/Users/Ger...  -  □  ×
99 54 60 79 36 60
Enter search value: 61
Number 61 not found!
```



```
file:///C:/Users/Ger...  -  □  ×
83 71 46 4 97 63
Enter search value: 83
Number 83 found at position 1!
```



```
file:///C:/Users/Ger...  -  □  ×
21 71 36 31 95 41
Enter search value: 41
Number 41 found at position 6!
```



# What's wrong with the code below?

```
void Start()
{
    int[] numbers = GenerateNumbers(10);

    Console.Write("Enter a search value: ");
    int searchValue = int.Parse(Console.ReadLine());

    int index = 0;
    while (numbers[index] != searchValue)
        index++;

    Console.WriteLine($"Search value found at index {index}");
}
```

*If searchValue is not present in the array, then index will finally 'reference' beyond the last element of the array; a runtime error will occur!*

## Exercise Dice

Create an array 'dice' with 6 elements of type integer. Fill this array with the value zero (so 6 times).

Generate a random number with a value between 1 and 6 (just like a dice), repeat this 6000 times. If the value is 1 increase array element 0, the same applies to the values 2, 3, 4, 5 and 6. Element `dice[x-1]` contains the number of times x has been thrown.

At the end of the loop, show the content of the array.

## Exercise Dice

```
dice ← array[6]

for i = 0 to 5
    dice[i] = 0

for i = 1 to 6000
    value = Random.Next(1, 7) // 1..6
    dice[value - 1] = dice[value - 1] + 1

for i = 0 to 5
    display dice[i]
```

# Homework

- (praktijkles) Programmeren 1
  - week 4 opdrachten → Moodle

