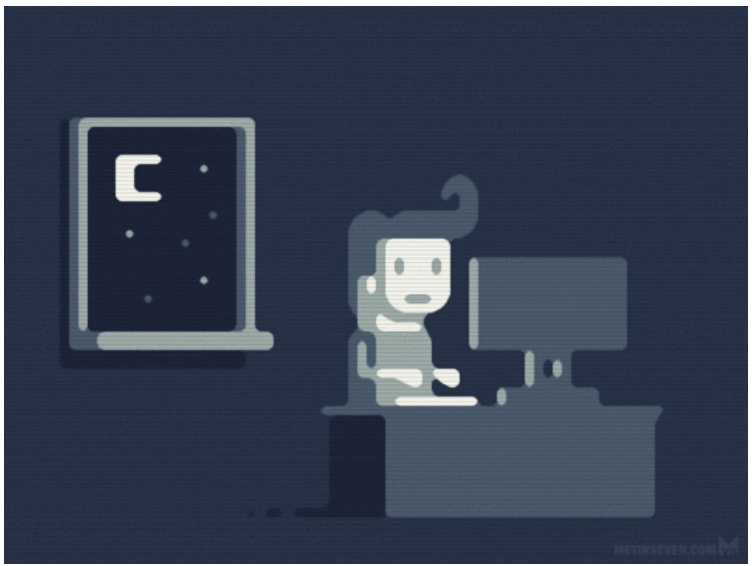
02 Variables & expressions

Programming fundamentals

YP0616 - YP0601

Elke Boonen & Tristan Vandevelde

Learning objectives (ECTS)



- **Basic principles** (types, operators, expressions) & **structures** (loop & if)
- **Arrays**, **lists**, **dictionaries**
- **Methods** and **functions**
- Basic principles of **OO**
- **Files**, in-and output **IO**
- **Exception** handling

Learning materials

- **Canvas** LMS https://thomasmore.instructure.com/
  - Presentations
  - E-Book: Fundamentals of Computer Programming with C#
  - Cheatsheet C#
  - Assigments (CodeGrade)
- **Online**
  - https://docs.microsoft.com/en-us/dotnet/csharp/
  - https://github.com/ElkeBoonen/ProgrammingFundamentals (code from slides)
  - https://github.com/ElkeBoonen/ProgrammingFundamentals-Students (code from class)
- **Software**
  - Visual Studio (Community) https://visualstudio.Microsoft.com/

Schedule

Before autumn break	**After autumn break**
01 Hello world	07 Exception handling
02 Variables & expression	08 Recap
03 If-structures	09 Collections
04 Loops	10 Methods
05 Files (IO)	11 OO
06 Arrays	12 OO
	13 Exam prep

Schedule is always subject to unexpected circumstances

Evaluation

- **1st term**
  - Permanent Evaluation (30 %):
    - CodeGrade exercises (each week, from week 02)
  - Computer Exam (70 %) use of cheatsheet only!
- **2nd term**
  - Computer Exam (100 %) use of cheatsheet only!

02 Variables & expressions

- Variables
- Operators and expressions
- Input & output
- Debugging
- CodeGrade

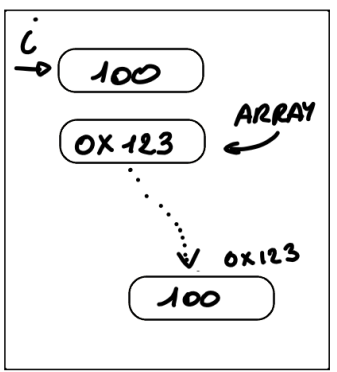
What is a variable?

- Container for **storing data** values
- C# **strong typed variables: type must be declared** before using!

- C# types: value types & reference types
  - **Value**: holds data directly in (RAM) memory
  - **Reference**: holds reference to memory location of data

```
int i = 100;

int[] array = {100}
```



RAM MEMORY

Value types - numbers

- Whole numbers aka **integers**
  - **byte**: 1 byte (0 to 255)
  - **short**: 2 bytes (-32768 to 32967)
  - **int**: 4 bytes (-2,147483,648 to 2,147483,647)
  - **long**: 8 bytes
- **Decimals** aka floating point numbers
  - **float**: 4 bytes (precision of 7 decimal digits)
  - **double**: 8 bytes (precision of 15 decimal digits)
- **Size** matters in **storage** (eg smartphone apps), **precision** is important in **calculations** (eg. space shuttle, deviation...) (see Ariane 5 video!)

https://www.youtube.com/embed/5tJPXYA0Nec?enablejsapi=1

Value types - text

- **char**: stores 1 single character,
  surrounded by single quotes ' '
  eg: 'a', '1', '-'
- **string**: stores a sequence of characters,
  surrounded by double quotes " "
  eg: "elke", "this is a sequence", 'bla bla
  bla"

What's in a name

- Variable names are **free to choose**
- Can only contain letters, digits & underscores (no spaces!)
- First character must be a letter
- Case sensitive!
- Keywords can’t be used as names
- Name styling: **camelCase**
  - name starts with lower case
  - every other word is capitalized

Finally get this show on the road!

- **Declare** variable: give it a name (can be grouped by type)
- **Initialize** variabele: give its first value = assignment

- Declaration and initialization can be done together

```
1  int number;
2  double lengthObject, widthObject;
3  char firstLetterOfAlphabet;
4
5  number = 5;
6  lengthObject = 5.6;
7  widthObject = 8.4;
8  firstLetterOfAlphabet = 'a';
9
10 string myName = "elke";
11 int myAge = 41;
```

A variable is never afraid of change!

- Variables can **always change value**, but **not** the **name nor** the **type**
- **Note**: variables can also pass their values to each other.


```
1  int number;
2  double lengthObject, widthObject;
3  char firstLetterOfAlphabet;
4
5  number = 5;
6  lengthObject = 5.6;
7  widthObject = 8.4;
8  firstLetterOfAlphabet = 'a';
9
10 string myName = "elke";
11 int myAge = 40;
12
13 widthObject = 1.2;
14 lengthObject = widthObject;
15 myName = "elke boonen";
16 number = myAge;
17 number = 10;
```

"No comment" is a comment

- A **comment** is an **explanation or annotation** in the code
- They are added to clarify code and are ignored by the compiler

```
1 //this is one line of comment
2
3 /*
4  These are mulitple lines of comment!
5 */
```

02 Variables & expressions

- ~~Variables~~
- Operators and expressions
- Input & output
- Debugging
- CodeGrade

Let's express ourselves

- An expression
  - resolves in a value
  - is a combination of values, variables and operators


```
1   int a, b, c;
2   a = 5; //value
3   b = a; //variable
4   c = a * b; //calculation
```

Arithmetic unary operators

- 1 operand (pre- or postfix)
  - increment ++
  - decrement --

```
1   int a;
2   a = 5;
3
4   //a = a + 1
5   a++;
6   ++a;
7
8   //a = a - 1
9   a--;
10  --a;
```

Post- or prefix operator?

- The result of x++ (postfix) is the value of x before the operation.
- The result of ++x (prefix) is the value of x after the operation.

```
1 int i = 3;
2 Console.WriteLine(i);   // output : 3
3 Console.WriteLine(i++); // output : 3
4 Console.WriteLine(i);   // output : 4
5 double d = 1.5;
6 Console.WriteLine(d);   // output : 1.5
7 Console.WriteLine(++d); // output : 2.5
8 Console.WriteLine(d);   // output : 2.5
```

Arithmetic binary operators

- 2 operands
  - + - * / (known arithmetic operators)
  - % modulo (remainder after division)

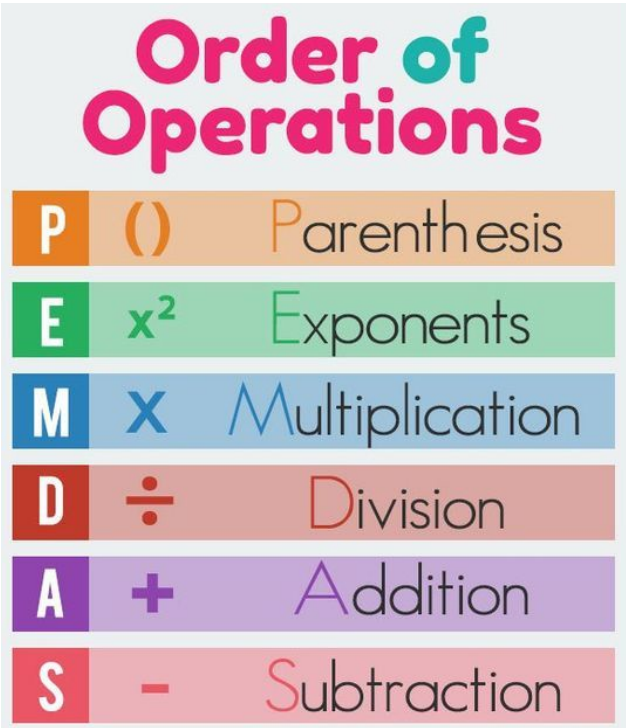
```
1  Console.WriteLine(8 + 2);        // output : 10
2  Console.WriteLine(8 - 2);        // output : 6
3  Console.WriteLine(8 * 2);        // output : 16
4  Console.WriteLine(7.0 / 2.0);    // output : 3.5
5  Console.WriteLine(7 / 2.0);      // output : 3.5
6  Console.WriteLine(7 / 2);        // output : 3
7  //integer division: no digits after the decimal point
8
9  //modulo: remainder after integer division
10 Console.WriteLine(7 % 2);        // output : 1 --> 7/2 = 3 => 7-(2 * 3) = 1
```

- *If at least one operator is decimal, the result is decimal!*

Arithmetic binary operators

- **Use brackets** when writing expressions that er more complex (e.g. having more operators) **to avoid difficulties**!

```
1  int x = 10;
2  int y = 200;
3
4  int z;
5
6  // Ambiguous
7  z = x + y / 100
8
9  // Unambiguous, recommended
10 z = x + (y / 100)
```



Compound assignment operators

- **Assignment operator =** assigning a value to a variable
- **Compound assignment operator op=** (aka shorthand operators) define a shorter syntax for assigning the result of an arithmetic operator
  - +=, -=, -*=, /=, %=

```
1 int x = 10;
2
3 x += 10;          // x = 20, same operation as x = x + 10
4 x -= 5;           // x = 15
5 x *= 2;           // x = 30
6 x /= 3;           // x = 10
7 x %= 3;           // x = 1
```

Comparison operators

- **Compare two values**
  - greater than (>) and less than (<)
  - greater than or equal to (>=) and less than or equal to (<=)
  - equality (==)
  - difference (!=)

```
1 int x = 10;
2 int y = 20
3
4 Console.WriteLine(x < y)          // True
5 Console.WriteLine(x > y)          // False
6 Console.WriteLine(x <= y)         // True
7 Console.WriteLine(x >= y)         // False
8 Console.WriteLine(x == y)         // False
9 Console.WriteLine(x != y)         // True
```

String operations

- Concatenate text with + or we can use
- If we want to add text and numbers, the number is always converted internally to text

```
1 myName = "Elke";
2 myName = myName + " Boonen";     //or myName += " Boonen"
3
4 Console.WriteLine("My name is " + myName);
5
6 Console.WriteLine(5 + 5);        // output : 10
7 Console.WriteLine("5" + 5);      // output : 55
8 Console.WriteLine(5 + "5");      // output : 55
```

String interpolation

- Another option for concatenating strings
- **Substitutes values** of variables into placeholders in a string
- Put **$ in front of the string** and use **{ }** to **surround** variables!

```
1 string firstName = "Elke";
2 string lastName = "Boonen";
3 string name = $"My full name is: {firstName} {lastName}";
4 Console.WriteLine(name);
```

Access chars in strings

- A **string is a sequence of characters**
- Access the characters in a string by the **index number** of the character **between [ ]**
- Note: **zero-based numbering**

```
1 string name = "Elke";
2 char first = name[0]
3 Console.WriteLine($"2nd char is {name[1]}");
```

REAL

PROGRAMMERS

COUNT FROM

0

02 Variables & expressions

- ~~Variables~~
- ~~Operators and expressions~~
- Input & output
- Debugging
- CodeGrade

What goes in, must come out

- **Output to console**
  - Console.Write(""): displays output
  - Console.WriteLine(""): displays output and provides a new line character at the end of string
- **Input from console**: returns a value
  - Console.ReadLine(): reads next line of characters (returns a string)
  - Console.Read(): reads only the next character (returns an int - ASCII Code of character)

With or without lines

```
1 string age;
2 Console.WriteLine("What is your age?");
3 age = Console.ReadLine();
4 Console.WriteLine("You are " + age + " yrs old ");
5
6 //or use string interpolation
7 Console.WriteLine($"You are {age} yrs old ");
```

```
What is your age?
40
You are 40 yrs old
You are 40 yrs old
```

```
1 int ascii;
2 Console.Write("Give me a random character : ");
3 ascii = Console.Read();
4 Console.WriteLine("ASCII value char : " + ascii);
```

```
Give me a random character :
u
ASCII value char : 117
```

It's converting time!

- Type **converting/casting** is **changing** one **type** of data to another!
- We often do it when we work with **user input!**

- **Implicit type** conversion: automatically done by the compiler (type-safe)

```
1 Console.Write("Give me a random character : ");
2 int ascii = Console.Read();
3 Console.WriteLine("ASCII value char : " + ascii); // ascii implicit to string
4
5 int nr = 5;
6 double dec = nr + 0.5;  // nr implicit to double
7 Console.WriteLine(dec); // dec implicit to string
```

It's converting time!

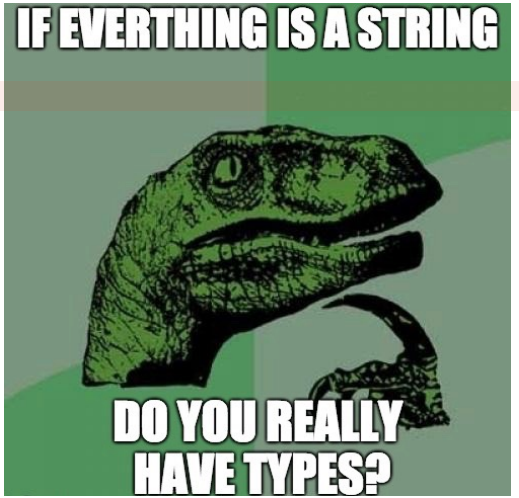
- **Explicit type** conversion: done explicitly by the programmer using
  - **pre-defined functions** (see more documentation online)

```
1 Console.WriteLine("What is your age?");
2 string age = Console.ReadLine();
3 int intAge = Convert.ToInt32(age); //convert
4 int year = DateTime.Now.Year - intAge;
5 Console.WriteLine("Birth year: " + year);
```

  - **a cast operator**

```
1 Console.Write("Give me a random character: ");
2 int ascii = Console.Read();
3 Console.WriteLine("ASCII value char: " + ascii);
4 char ch = (char)ascii; //cast
5 Console.WriteLine("Character pressed: " + ch);
```

It's converting time!



- **Everything in the console is a piece of text = string!**
  - To write to console = **implicit conversion**
  - To read from the console = **explicit conversion is needed**

```
1 Console.WriteLine("What's your name?");
2 string name = Console.ReadLine();                    //no conversion needed
3 Console.WriteLine($"Hello {name}!");
4
5 Console.WriteLine("Enter a number?");
6 int number = Convert.ToInt32(Console.ReadLine());    //explicit
7 number += 10;
8 Console.WriteLine($"Number + 10 = {number}!");       //implicit
```

02 Variables & expressions

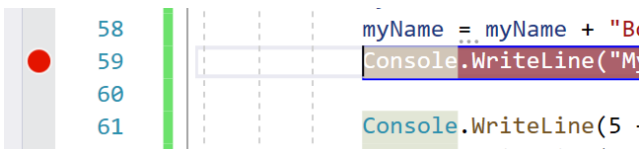
- ~~Variables~~
- ~~Operators and expressions~~
- ~~Input & output~~
- Debugging
- CodeGrade

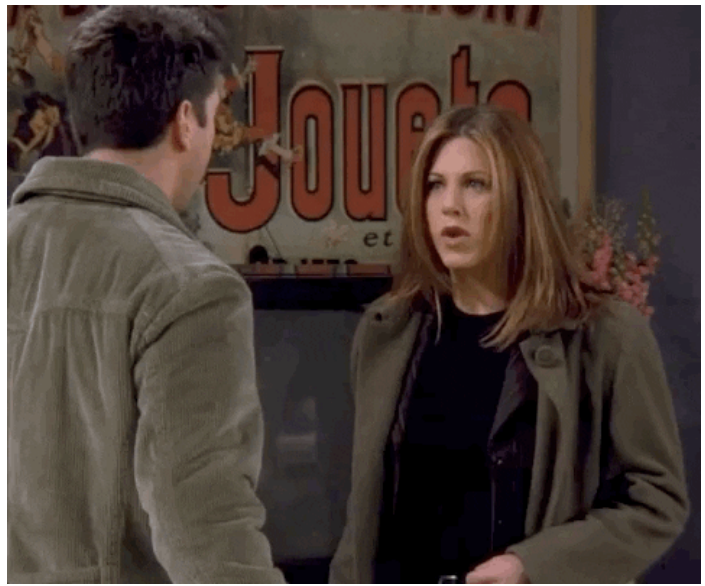
Whoopsie daisies

- Our output isn't what we had in mind!
- Where do we even start looking for the mistake?
- Dive into your code with the **debugging tools**!

We are on a break!

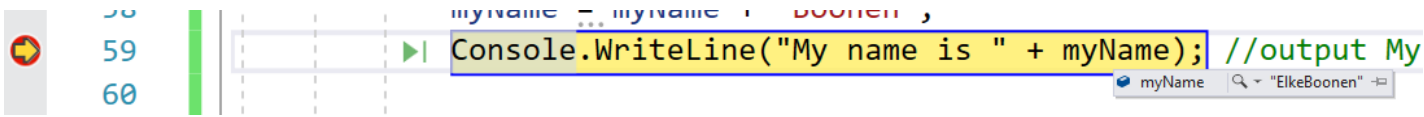
- Place a **breakpoint** in your code
  - click in the grey area left to the line numbers (click again to remove)





- Use **F10 to walk through your code** step by step
- Hover over the variables or take a look

  at the locals-window below!



02 Variables & expressions

- ~~Variables~~
- ~~Operators and expressions~~
- ~~Input & output~~
- ~~Debugging~~
- CodeGrade

We ❤ CodeGrade, yes we do!

- Find the assignments in Canvas
- Spend some hours to do them (plagiarism results in a 0!)
- Submit (only) your .cs-file
- Wait for your automatically generated result
- Hit a home run? Do a little dance ;)
- Not so successful? Tweak your solution and re-submit!
- You can keep practicing until the deadline to become better, but also to score higher points on your permanent evaluation

Practice makes perfect!

- Do your exercises, spend the hours!
- The better the exercises, the better the exam!

Say what? How many hours?

6 SP = 6 * 28 hours = 168 hours

Lessons = 12 * 3 hours = 36 hours

Exam = 2 hours

Exercise = 168-36-2 = 130 hours