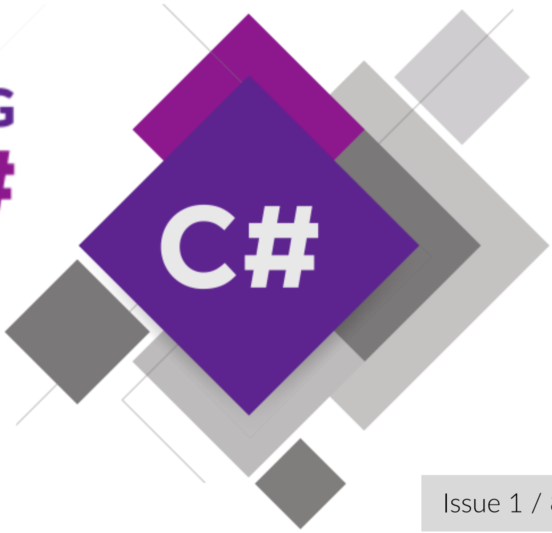
PROGRAMMING BASICS WITH C#



Issue 1 / 8

FIRST STEPS IN PROGRAMMING

Learn about writing commands, creating and running simple programs in C#, about languages, compilers, interpreters, development and runtime environments, the C# language and the Visual Studio IDE, creating console apps, solving practical problems.

Get an idea how to build simple Desktop GUI and Web applications in C# using Visual Studio.

Dr. Svetlin Nakov

Software University – https://softuni.org

First Steps in Programming

In this mini book, we are going to find out **what programming is** in its core and how to write simple programs and apps.

- We will get familiar with the idea of **programming languages** and development platforms, along with concepts like **compilation** and code **execution**.
- We are going to look at the **environments for software development** (IDEs) and how to work with them, in particular with **Visual Studio**.
- We will write and execute our **first program** with the programming language **C#** in **Visual Studio**.
- We will **exercise** with a couple of tasks: we will create **console-based** programs, a graphical application (**GUI**) and a **Web** application.
- We will learn how to check the correctness of the solutions from this book in **the Judge system of SoftUni**.
- We will get to know some of the **typical mistakes**, which are often made during code writing and how to prevent doing them.

Video: Overview

Watch a video about what we shall learn in this section here: https://youtu.be/6RVKMlXtWg4.

Introduction to Coding by Examples

Coding means to write **commands** for the computer, e.g.

```
Console.WriteLine("Welcome to coding");
```

Run the above code example: https://repl.it/@nakov/welcome-to-coding-csharp.

When **executed**, the above command prints the following text:

```
Welcome to coding
```

Several commands can be written as a sequence, called "**computer program**":

```
var size = 5;
Console.WriteLine("Size = " + size);
Console.WriteLine("Area = " + size * size);
```

Run the above code example: https://repl.it/@nakov/square-area-csharp.

The **result** (output) from the above program is as follows:

```
Size = 5
Area = 25
```

The above **program** (sequence of commands) consists of 3 commands:

1. Defines a variable `size` and stores an integer value `5` in it.

2. Prints the value of the variable `a`, along with some text.
3. Calculates and prints the value of the expression `a * a`.

Let's explain in greater detail what is **programming**, what is programing **language**, how to write **commands** and simple **programs** in the **C#** language, using the Visual Studio development environment.

Computer Programs – Concepts

Let's start with the **concepts of computer programming**: computer programs, algorithms, programming languages, code compilation and execution.

Video: Computer Programs, Compilers, Interpreters

Watch a video lesson about the concepts of programming, programs, compilers and interpreters here: https://youtu.be/U16C61p6m1k.

What It Means "To Program"?

To program means to **give commands** to the computer, for example "*to play a sound*", "*to print something on the screen*" or "*to multiply two numbers*". When the commands are one after another, they are called **a computer program**. The text of computer programs is called **a program code** (or **a source code**, or even shorter – **code**).

Example of **command** for the computer:

```
Console.WriteLine("Welcome to coding");
```

Run the above code example: https://repl.it/@nakov/welcome-to-coding-csharp.

When **executed**, the above command prints the following text:

```
Welcome to coding
```

Computer Programs

Computer programs represent **a sequence of commands** that are written in certain **programming language**, like C#, Java, JavaScript, Python, C++, PHP, C, Ruby, Swift, Go or another.

Example of **computer program** in C#:

```
using System;

class SquareArea
{
    public static void Main()
    {
        var size = 5;
        Console.WriteLine("Size = " + size);
        Console.WriteLine("Area = " + size * size);
    }
}
```

Run the above code example: https://repl.it/@nakov/square-area-csharp.

The above program defines a **class** `SquareArea`, holding a **method** `Main()`, which holds a sequence of **3 commands**:

1. Declaring and assigning a **variable**: `var size = 5;`
2. Calculating and **printing** an **expression**: `Console.WriteLine("Size = " + size);`
3. Calculating and **printing** an **expression**: `Console.WriteLine("Area = " + size * size);`

The **result** (output) from the above program is as follows:

```
Size = 5
Area = 25
```

We shall **explain in detail how to write programs in C#**, why we need to define a **class** and why we need to define a **method** `Main()` a bit later. Now, assume that the C# language requires all the above code in order to execute a sequence of command.

In order to write commands, we should know **the syntax and the semantics of the language** which we are working with, in our case – **C#**. Therefore, we are going to get familiar with the syntax and the semantics of the language C#, and with programming generally, in the current book, by learning step by step code writing from the simpler to the more complex programming constructions.

Algorithms

Computer programs usually execute some algorithm. **Algorithms** are a **sequence of steps**, necessary for the completion of a certain task and for gaining some expected result, something like a "recipe".

For example, if we fry eggs, we follow some recipe (an algorithm): we warm up the oil in a pan, break the eggs inside it, wait for them to fry and move them away from the stove.

Similarly, in programming **the computer programs execute algorithms**: a sequence of commands, necessary for the completion of a certain task. For example, to arrange a sequence of numbers in an ascending order, an algorithm is needed, e.g. find the smallest number and print it, then find the smallest number among the rest of the numbers and print it, and this is repeated until there are no more numbers left.

For convenience when creating programs, for writing programming code, for execution of programs and other operations related to programming, we need a **development environment**, for example Visual Studio.

Programming Languages, Compilers, Interpreters and Development Environments

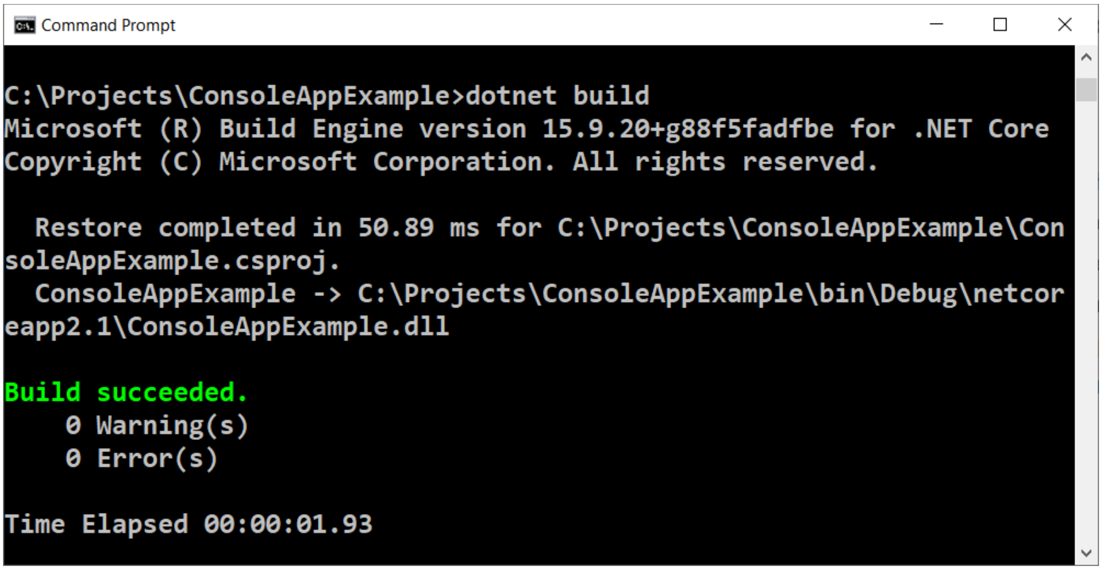
Let's review some concepts from computer programming: programming languages, compilers, interpreters and development environments (IDEs).

Programming Languages

A **programming language** is an artificial language (syntax for expression), meant for **giving commands** that we want the computer to read, process and execute. Using programming languages, we write sequences of commands (**programs**), which **define what the computer should do**. Examples of programming languages are C#, Java, JavaScript, Python, C, C++, PHP, Swift, Go and many others. These languages differ in their philosophy, syntax, purpose, programming constructions and execution environment. The execution of computer programs can be done with **a compiler** or with **an interpreter**.

Compilers

The compiler translates the code from programming language to **machine code**, as for each of the constructions (commands) in the code it chooses a proper, previously prepared fragment of machine code and in the meantime it **checks the text of the program for errors**. Together, the compiled fragments comprise the program into a machine code, as the microprocessor of the computer expects it. After the program has been compiled, it can be executed directly from the microprocessor in cooperation with the operating system. With compiler-based programming languages **the compilation of the program** is done obligatory before its execution, and syntax errors (wrong commands) are found during compile time. Languages like C++, C#, Java, Swift and Go work with a **compiler**. This an example how the compiler execution may look (the console-based `dotnet` compiler):

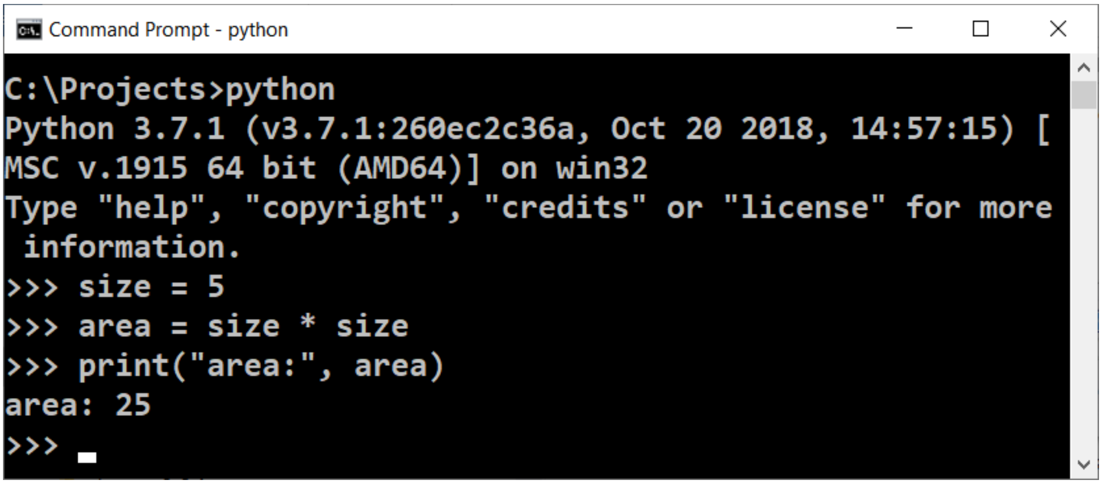


```
C:\Projects\ConsoleAppExample>dotnet build
Microsoft (R) Build Engine version 15.9.20+g88f5fadfbe for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 50.89 ms for C:\Projects\ConsoleAppExample\Con
soleAppExample.csproj.
  ConsoleAppExample -> C:\Projects\ConsoleAppExample\bin\Debug\netcor
eapp2.1\ConsoleAppExample.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.93
```

Interpreters

Some programming languages do not use a compiler and are being **interpreted directly** by a specialized software called an "interpreter". **The interpreter** is "**a program for executing programs**", written in some programming language. It executes the commands in the program one after another, as it understands not only a single command and sequences of commands, but also other language constructions (evaluations, iterations, functions, etc.). Languages like Python, PHP and JavaScript work with an interpreter and are being executed without being compiled. Due to the absence of previous compilation, in interpreted languages **the errors are being found during**

the execution time, after the program starts running, not previously. This is an example how an interpreter may look (the `python` interpreter in the console):



Command Prompt - python

```
C:\Projects>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [
MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
 information.
>>> size = 5
>>> area = size * size
>>> print("area:", area)
area:  25
>>> _
```

Development Environments (IDE)

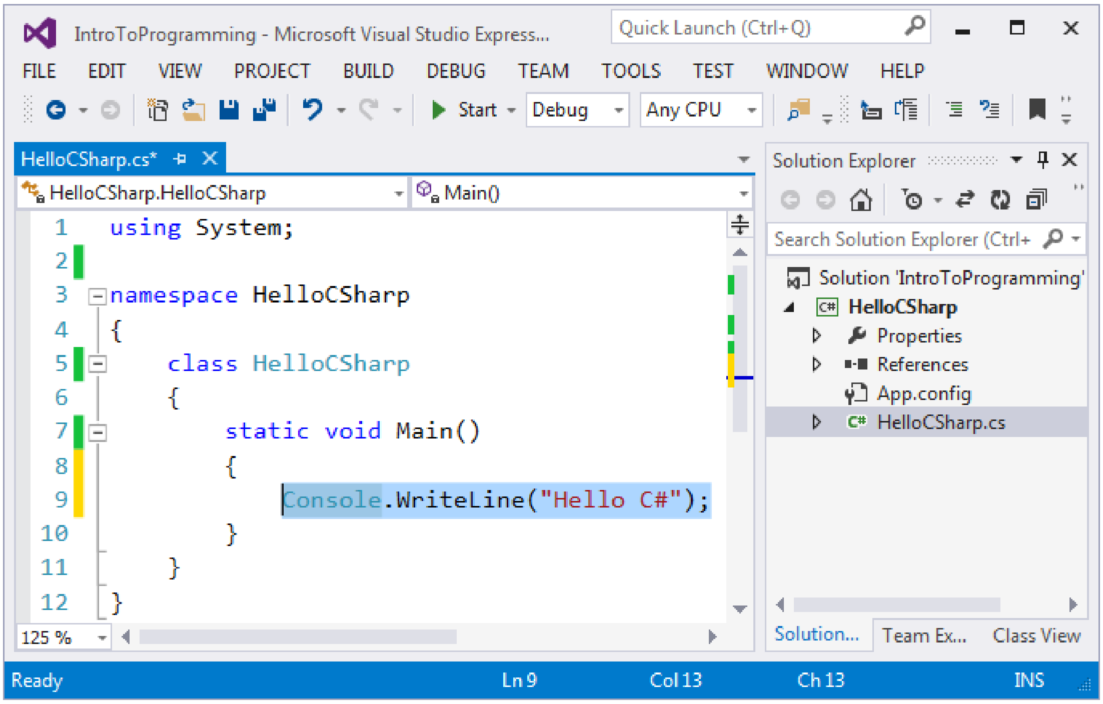
An environment for development (Integrated Development Environment – **IDE**) is a combination of traditional tools for development of software applications. In the development environment we write code, compile and execute the programs. Development environments integrate in them **a text editor** for writing code, **a programming language**, **a compiler or an interpreter** and **a runtime environment** for executing programs, **a debugger** for tracking the program and seeking out errors, **tools for user interface design** and other tools and add-ons.

Environments for development are convenient, because they integrate everything necessary for the development of the program, without the need to exit the environment. If we don't use an environment for development, we will have to write the code in a text editor, to compile it with a command on the console, to run it with another command on the console and to write more additional commands when needed, which is very time consuming. That is why most of the programmers use an IDE in their everyday work.

For programming with **the C# language** the most commonly used IDE is **Visual Studio**, which is developed and distributed freely by Microsoft and can be downloaded from: https://www.visualstudio.com/downloads. Alternatives of Visual Studio are:

- **Rider** – https://www.jetbrains.com/rider
- **MonoDevelop / Xamarin Studio** – https://www.monodevelop.com
- **SharpDevelop** – http://www.icsharpcode.net/OpenSource/SD

In the current book, we are going to use the development environment Visual Studio. This an example how a development IDE may look (the Visual Studio IDE for C#):


IntroToProgramming - Microsoft Visual Studio Express...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Start Debug Any CPU
HelloCSharp.cs*
HelloCSharp.HelloCSharp
Main()
Solution Explorer
Search Solution Explorer (Ctrl+
Solution 'IntroToProgramming'
HelloCSharp
Properties
References
App.config
HelloCSharp.cs
125 %
Solution... Team Ex... Class View
Ready Ln 9 Col 13 Ch 13 INS

```
using System;

namespace HelloCSharp
{
    class HelloCSharp
    {
        static void Main()
        {
            Console.WriteLine("Hello C#");
        }
    }
}
```

Runtime Environments, Low-Level and High-Level Languages

A program, in its essence, is a sequence of instructions that make the computer do a certain task. They are being entered by the programmer and are being executed unconditionally by the machine.

Video: Runtime Environments and Programming Languages

Watch a video lesson about runtime environments and programming languages here: https://youtu.be/ziG5v36lSVk.

Runtime Environments

Runtime environments are needed by some languages to execute the compiled programs. For example, the compiled C# programs are executed by the .NET Core runtime environment and the compiled Java programs are executed by Java JRE runtime environment. Other languages do not need compilation, but still require a runtime environment. For example, the Python programs are executed by the Python interpreter and runtime environment and the JavaScript programs are executed by the Node.js runtime environment or by a Web browser (which provides another JS runtime environment).

Programming Languages: Low-Level and High-Level

There are different kinds of programming languages. Via languages of the lowest level you can write the instructions that manage the processor, for example, using the "assembler" language. With a bit higher level languages, like C and C++, you can create an operating system, drivers for hardware management (for example a video card driver), web browsers, compilers, engines for graphics and games (game engines) and other system components and programs. With languages of even higher level, like **C#**, **Python** and **JavaScript** you can create application programs, for example a program for reading emails or a chat program.

Low level languages manage the hardware directly and require a lot of effort and a large count of commands to do a single task. **Languages of higher level** require less code for a single task, but do not have a direct access to hardware. Application software is developed using such languages, for example web applications and mobile applications.

Most of the software that we use daily, like music players, video players, GPS trackers, etc., are written with **languages for application programming** that are high-level, like C#, Java, Python, C++, JavaScript, PHP and others.

C# is a compiled language, which means that we write commands that are being compiled before they're being executed. Exactly these commands, through a help program (a compiler), are being transformed into a file, which can be executed (executable). To write a language like **C#** we need a text editor or a development environment and **.NET Runtime Environment** (like .NET Core).

.NET Runtime Environment

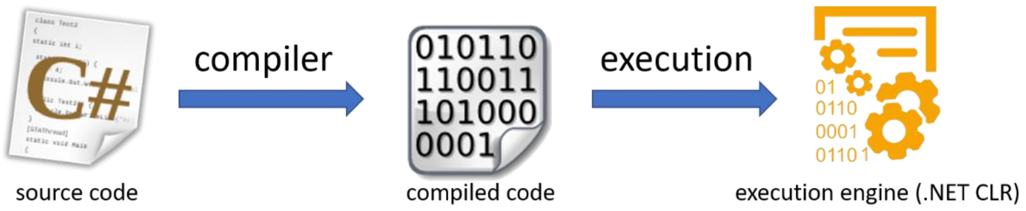
.NET Runtime Environment represents a virtual machine, something like a computer in the computer, which can run a compiled C# code. With the risk of going too deep into details, we have to explain that the language C# is compiled into an intermediary .NET code and is executed from the .NET environment, which compiles this intermediary code additionally into machine instructions (machine code) in order to be executed by the microprocessor. .NET environment contains libraries with classes, **CSC** compiler, **CLR** (Common Language Runtime – CLR) and other components, which are required for working with the language C# and run C# programs.

The .NET environment is available as a free software with open source code for every modern operating system (like Windows, Linux and Mac OS X). It has two variations, **.NET Framework** (the older one) and **.NET Core** (the newer one), but none of that is essential when it comes to getting into programming. Let us focus on writing programs with the C# language.

Compilation and Execution of C# Programs

As we have already mentioned, a program is **a sequence of commands**, otherwise said, it describes a sequence of calculations, evaluations, iterations and all kinds of similar operations, which aim to obtain certain result.

A C# program is written in a text format, and the text of the program is called a **source code**. It gets compiled into an **executable file** (for example `Program.cs` gets compiled to `Program.exe`) or it is **executed directly** from the .NET environment.



The process of **compilation** of the code before its execution is used only in compiled languages like C#, Java and C++. With **scripts and interpreted languages**, like JavaScript, Python and PHP, the source code gets executed step by step by an interpreter.

Computer Programs – Examples

Let's start with a few simple examples of short C# programs.

Video: Computer Programs – Examples

Watch a video lesson about the explained below sample computer programs: https://youtu.be/TlwcDNJFid4.

Example: A Program That Plays the Musical Note "A"

Our **first program** is going to be a single **C# command** that **plays the musical note** "A" (432 Hz) with a duration of half a second (500 milliseconds):

```
Console.Beep(432, 500);
```

A bit later we will find out how we can execute this command and hear the sound of the note, but for now let's just look at what the commands in programming represent. Let's get to know a couple more examples.

Example: A Program That Plays a Sequence of Musical Notes

We can complicate the previous program by giving for execution **repeating commands in a loop** for playing a sequence of notes with rising height:

```
for (i = 200; i <= 4000; i += 200)
{
    Console.Beep(i, 100);
}
```

In the example above we made the computer play one after another for a very short time (100 milliseconds) all the notes with height 200, 400, 600 etc. Hz until they reach 4000 Hz. The result of the program is playing something like a **melody**.

Example: A Program That Converts USD to EUR

Let's look at another simple program that reads from the user some **amount** of money in U.S. Dollars (USD), an integer, converts it into **Euro** (EUR) by **dividing it** by the Euro's rate and **prints** the obtained result. This is a program of 3 consecutive commands:

```
var dollars = int.Parse(Console.ReadLine());
var euro = dollars * 0.883795087;
Console.WriteLine(euro);
```

Run the above code example: https://repl.it/@nakov/dollars-to-euro-converter-csharp.

We examined **three examples of computer programs**: a single command, series of commands in a loop and 3 consecutive commands. Now let's get to the more interesting part: how we can write our own programs in **C#** and how we can compile them and run them.

How to Write a Console Application?

As a next step, let's pass through **the steps of creating and executing a computer program** that reads and writes its data from and on the **text console** (a window for entering and printing text). These programs are called "**console programs**". But before that, we should **install and prepare the development environment**, in which we are going to write and run the C# programs from this book and the exercises in it.

Development Environments (IDE) and Visual Studio

As it has already been said, in order to program we need an **Integrated Development Environment** (IDE). This is actually an editor for programs, in which we write the program code and we can compile it and run it to see the errors, fix them and start the program again.

- For programming with C# we use **Visual Studio** IDE for Windows operating system and **MonoDevelop** or **Raider** for Linux or Mac OS X.
- If we program with Java, the environments **IntelliJ IDEA**, **Eclipse** or **NetBeans** are suitable.
- If we write in Python, we can use the **PyCharm** environment.

Video: Installing and Running Visual Studio

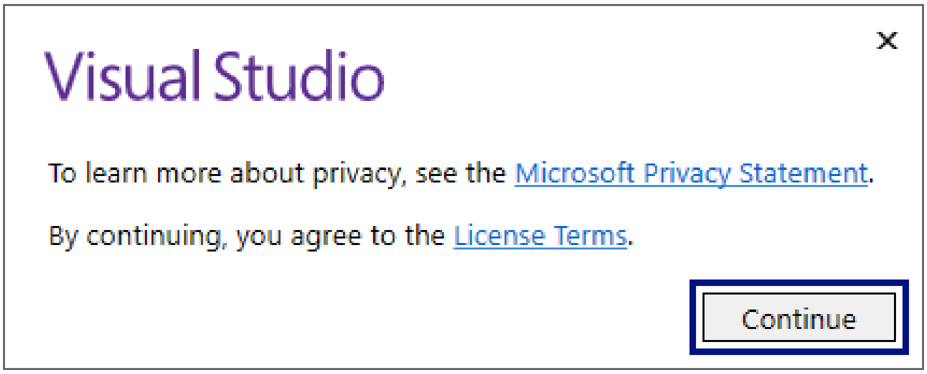
Watch the following video lesson about how to install and run the Visual Studio IDE: https://youtu.be/6AhALTJEagA.

Installing Visual Studio

We begin with the installation of the integrated environment **Microsoft Visual Studio** (Community, version 2017, latest as of June 2017).

The **Community** version of Visual Studio (VS) is distributed freely by Microsoft and can be downloaded from: https://www.visualstudio.com/vs/community. The installation is typical for Windows with [**Next**], [**Next**] and [**Finish**], but it's important to include the components for "**desktop development**" and "**ASP.NET**". It is not necessary to change the rest of the settings for the installation.

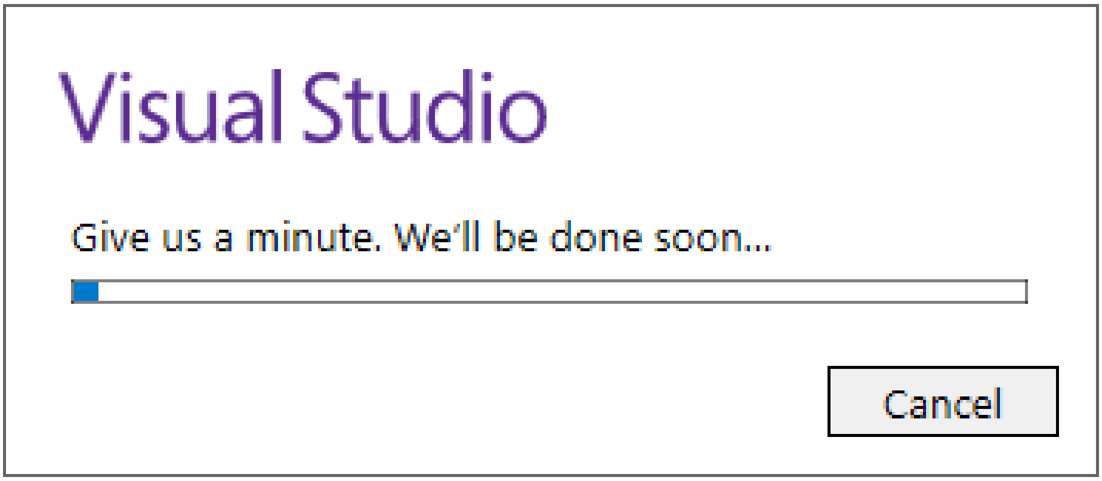
The next lines describe in detail **the steps for the installation of Visual Studio** (version Community 2017). After we download the installation file and start it, the following screen appears:


Visual Studio

To learn more about privacy, see the Microsoft Privacy Statement.

By continuing, you agree to the License Terms.

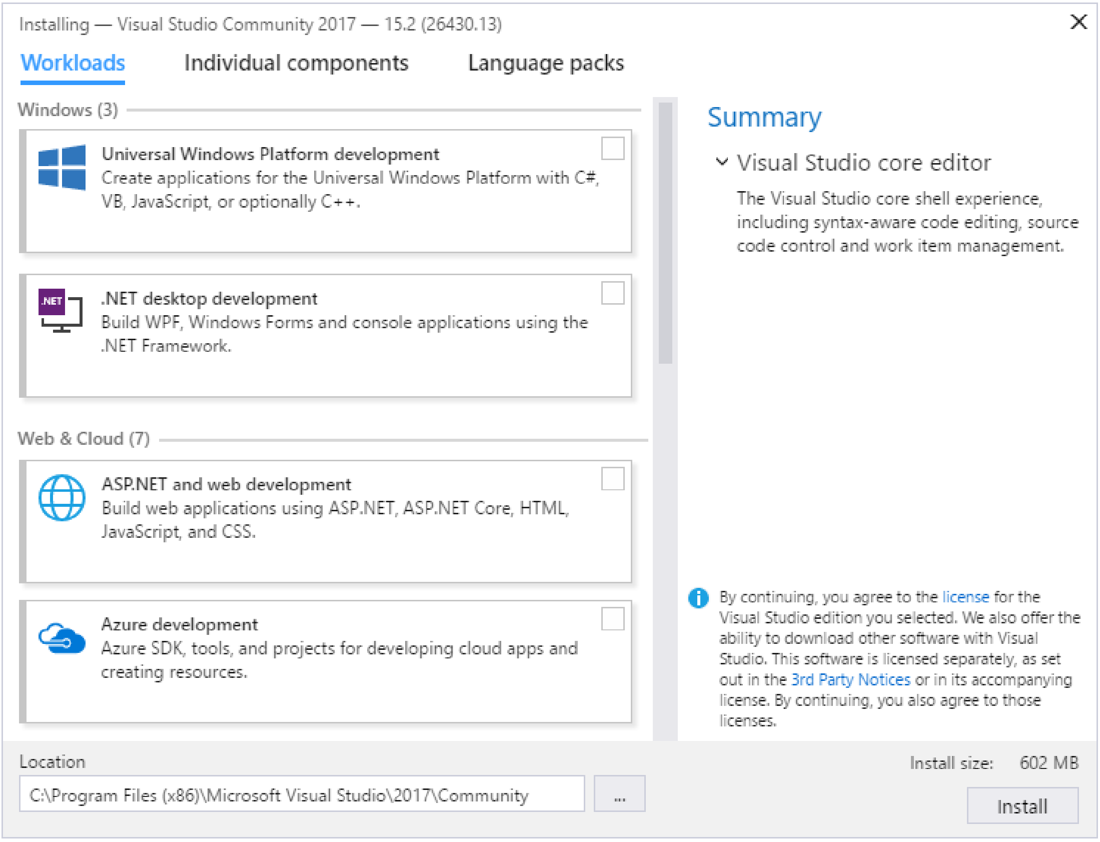
Press the [**Continue**] button and you will see the screen bellow:



Visual Studio

Give us a minute. We'll be done soon...

A screen with the installation panel of Visual Studio is being loaded.



Installing — Visual Studio Community 2017 — 15.2 (26430.13)

Windows (3)

Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

.NET desktop development
Build WPF, Windows Forms and console applications using the .NET Framework.

Web & Cloud (7)

ASP.NET and web development
Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and CSS.

Azure development
Azure SDK, tools, and projects for developing cloud apps and creating resources.

Summary

Visual Studio core editor

The Visual Studio core shell experience, including syntax-aware code editing, source code control and work item management.

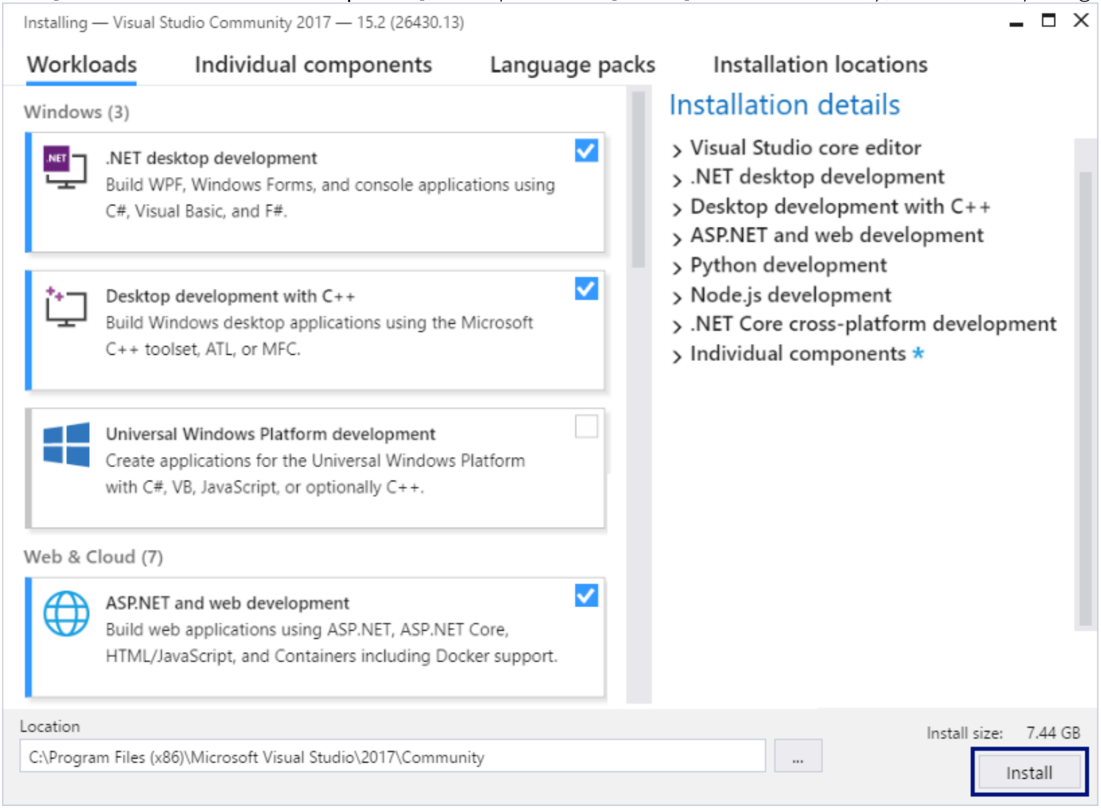
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Location

C:\Program Files (x86)\Microsoft Visual Studio\2017\Community

Install size: 602 MB

Put a check mark on [Universal Windows Platform development], [.NET desktop development] and [ASP.NET and web development], then press the [Install] button. Basically, this is everything.



Installing — Visual Studio Community 2017 — 15.2 (26430.13)

Windows (3)

- [✓] .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.
- [✓] Desktop development with C++ — Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.
- [ ] Universal Windows Platform development — Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

Web & Cloud (7)

- [✓] ASP.NET and web development — Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.

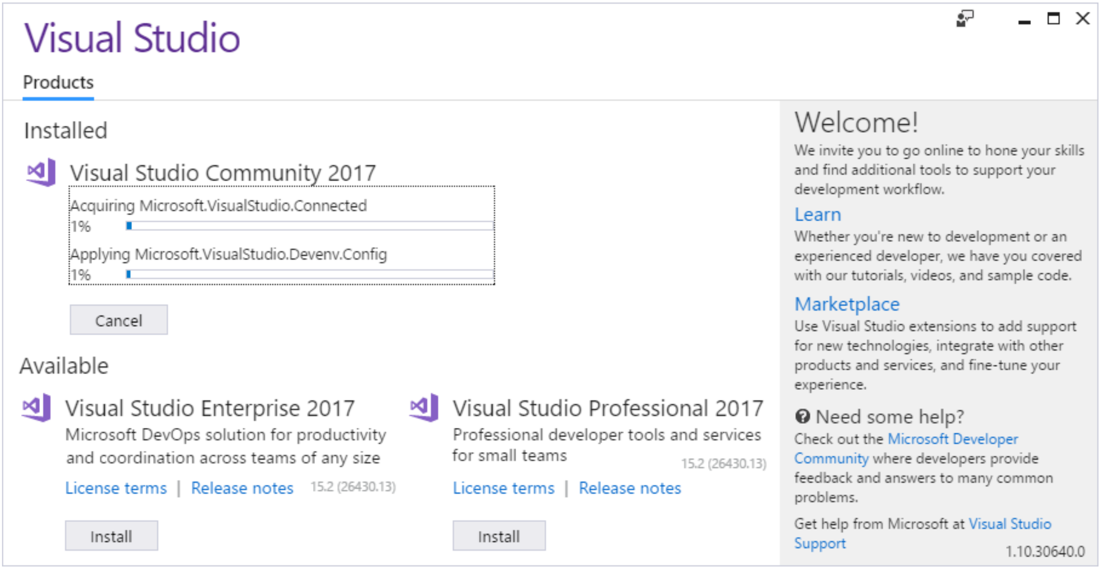
Installation details

- Visual Studio core editor
- .NET desktop development
- Desktop development with C++
- ASP.NET and web development
- Python development
- Node.js development
- .NET Core cross-platform development
- Individual components *

Location: C:\Program Files (x86)\Microsoft Visual Studio\2017\Community

Install size: 7.44 GB

The installation of Visual Studio begins, and a screen like the one bellow will appear:



Visual Studio

Products

Installed

Visual Studio Community 2017
Acquiring Microsoft.VisualStudio.Connected
1%
Applying Microsoft.VisualStudio.Devenv.Config
1%

Available

Visual Studio Enterprise 2017
Microsoft DevOps solution for productivity and coordination across teams of any size
License terms | Release notes 15.2 (26430.13)

Visual Studio Professional 2017
Professional developer tools and services for small teams 15.2 (26430.13)
License terms | Release notes

Welcome!
We invite you to go online to hone your skills and find additional tools to support your development workflow.

Learn
Whether you're new to development or an experienced developer, we have you covered with our tutorials, videos, and sample code.

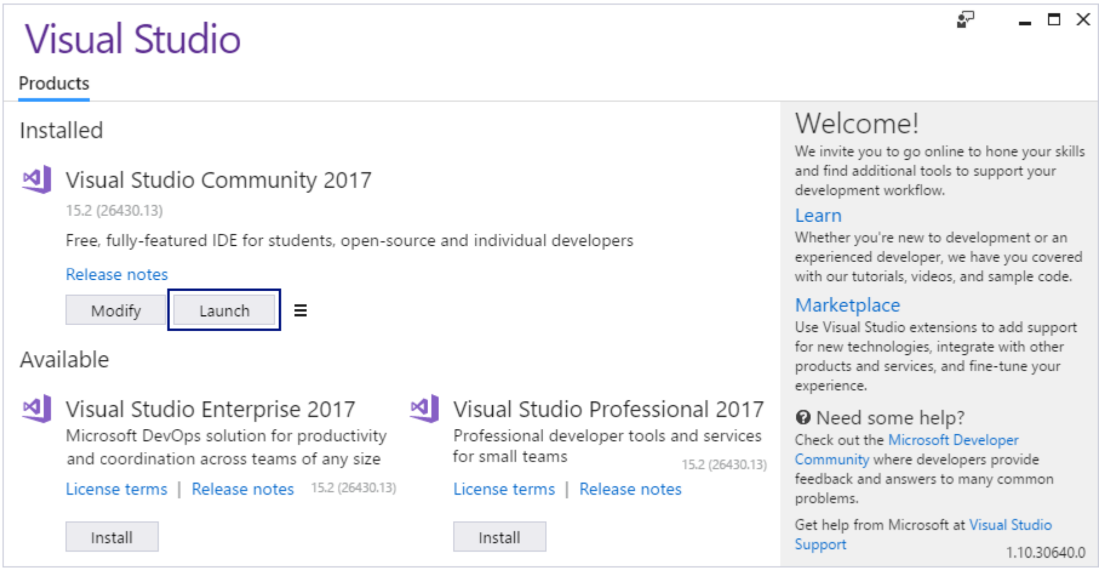
Marketplace
Use Visual Studio extensions to add support for new technologies, integrate with other products and services, and fine-tune your experience.

Need some help?
Check out the Microsoft Developer Community where developers provide feedback and answers to many common problems.

Get help from Microsoft at Visual Studio Support

1.10.30640.0

After Visual Studio is installed, an informative screen will appear. Press the [Launch] button to start it.



Visual Studio

Products

Installed

Visual Studio Community 2017

15.2 (26430.13)

Free, fully-featured IDE for students, open-source and individual developers

Release notes

Modify | Launch

Available

Visual Studio Enterprise 2017

Microsoft DevOps solution for productivity and coordination across teams of any size

License terms | Release notes 15.2 (26430.13)

Install

Visual Studio Professional 2017

Professional developer tools and services for small teams

15.2 (26430.13)

License terms | Release notes

Install

Welcome!

We invite you to go online to hone your skills and find additional tools to support your development workflow.

Learn

Whether you're new to development or an experienced developer, we have you covered with our tutorials, videos, and sample code.

Marketplace

Use Visual Studio extensions to add support for new technologies, integrate with other products and services, and fine-tune your experience.

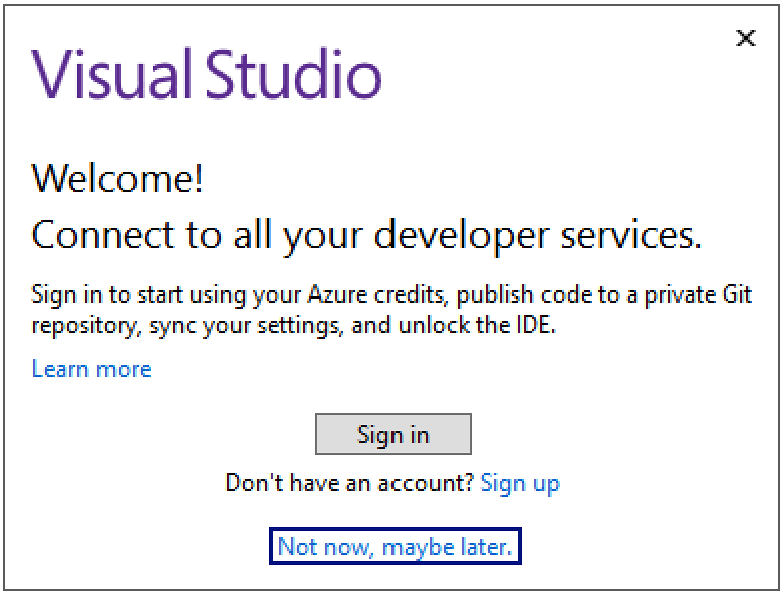
Need some help?

Check out the Microsoft Developer Community where developers provide feedback and answers to many common problems.

Get help from Microsoft at Visual Studio Support

1.10.30640.0

Upon **starting VS** a screen appears like the one bellow. On it you can choose whether you will enter Visual Studio using a Microsoft account. For now, we choose to continue without being logged into our Microsoft account, and therefore we choose the option [**Not now, maybe later.**]. At a later point, if you have such an account, you may log in, and if you don't have one, and you have difficulties with its creation, you can always write in the SoftUni's discussion forum: http://forum.softuni.org.



Visual Studio

Welcome!

Connect to all your developer services.

Sign in to start using your Azure credits, publish code to a private Git repository, sync your settings, and unlock the IDE.

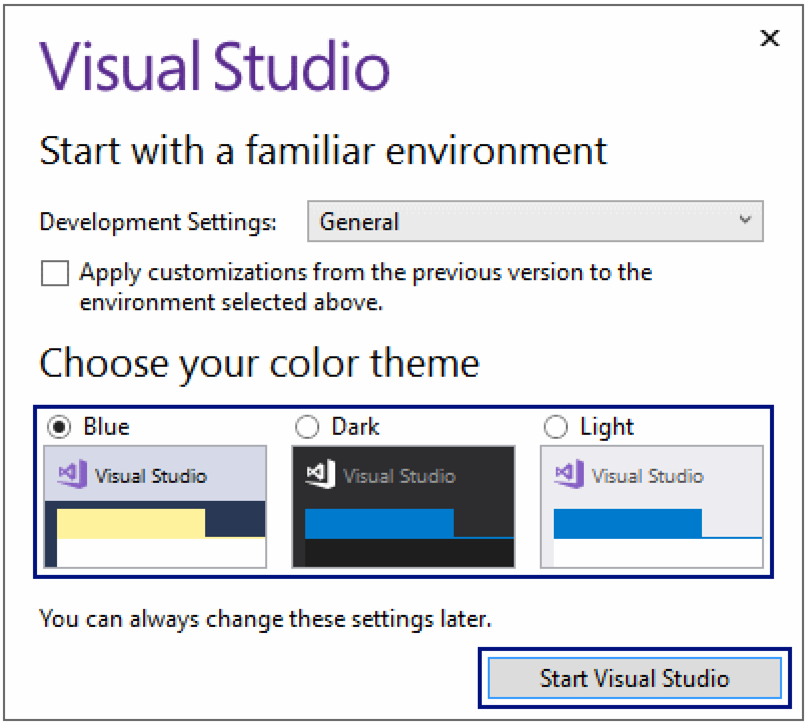
Learn more

Sign in

Don't have an account? Sign up

Not now, maybe later.

The next step is to choose **the color theme**, in which Visual Studio is visualized. The choice here lays completely on the preferences of the user and it doesn't matter which option will be chosen.


Visual Studio

Start with a familiar environment

Development Settings: General

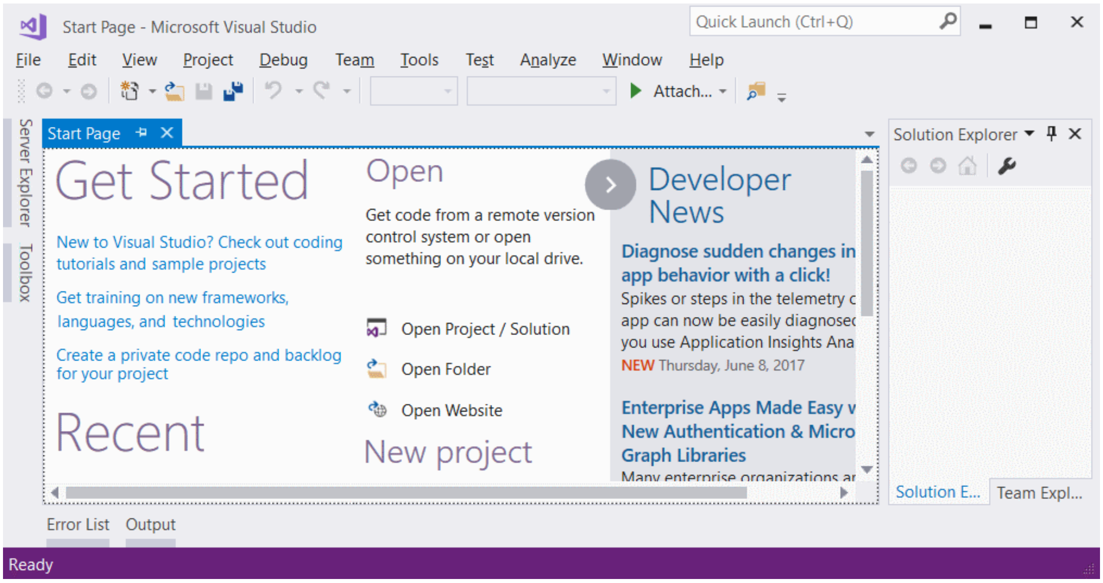
☐ Apply customizations from the previous version to the environment selected above.

Choose your color theme

[✓ Blue] Dark Light

You can always change these settings later.

Press the [Start Visual Studio] button and the main view of Visual Studio Community will be displayed:



Get Started

New to Visual Studio? Check out coding tutorials and sample projects

Get training on new frameworks, languages, and technologies

Create a private code repo and backlog for your project

Recent

Open

Get code from a remote version control system or open something on your local drive.

Open Project / Solution

Open Folder

Open Website

New project

Developer News

Diagnose sudden changes in app behavior with a click!

Spikes or steps in the telemetry c app can now be easily diagnosec you use Application Insights Ana

NEW Thursday, June 8, 2017

Enterprise Apps Made Easy v New Authentication & Micro Graph Libraries

Many enterprise organizations ar

That's all. We are ready to work with Visual Studio.

Older Versions of Visual Studio

You can use older versions of Visual Studio (for example version 2015 or 2013 or even 2010 or 2005), but **it is not recommended**, as they don't contain some of the newer options for development, and not all the examples from the book will run the same way.

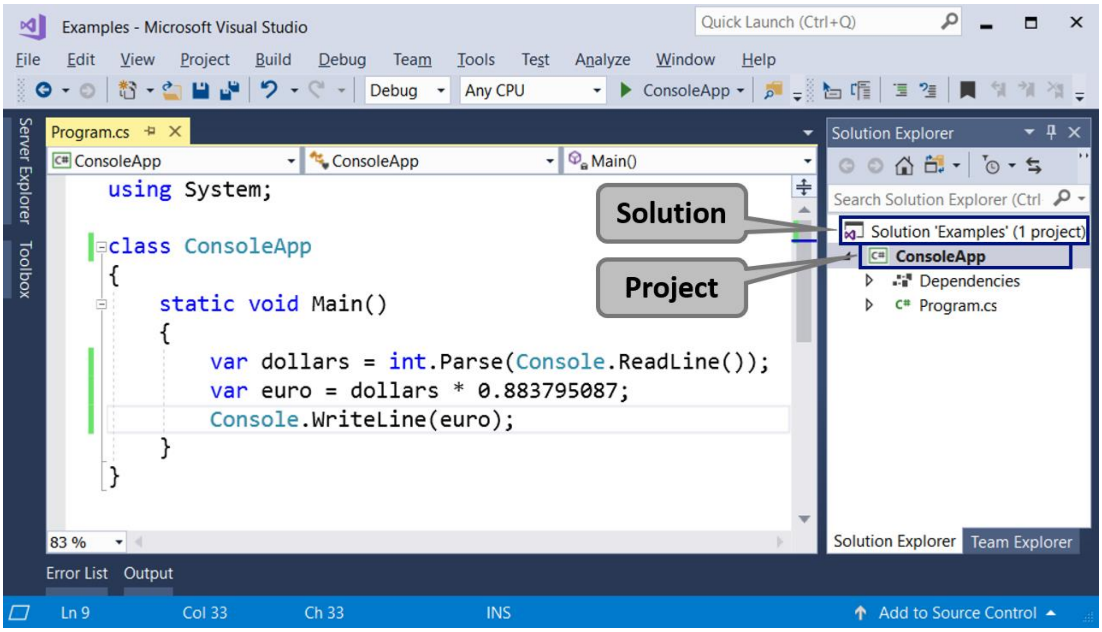
Online Development Environments

There are **alternative environments for development online** directly into your web browser. These environments are not very convenient, but if you don't have other opportunity, you can start your training with them and install Visual Studio later. Here are some useful links:

- For the language C# the site **.NET Fiddle** allows code writing and its execution online: https://dotnetfiddle.net.
- For Java you can use the following online Java IDE: https://www.compilejava.net.
- For JavaScript you can write a JS code directly in the console of a given browser when you press **[F12]**.
- The site Repl.it provides online coding environment for multiple languages (C#, Java, JS, Python, C++ and many more): https://repl.it.

Project Solutions and Projects in Visual Studio

Before we start working with Visual Studio, it is necessary to get familiar with the concepts of a **Visual Studio Solution** and a **Visual Studio Project**, which are an inevitable part of it.



```
using System;

class ConsoleApp
{
    static void Main()
    {
        var dollars = int.Parse(Console.ReadLine());
        var euro = dollars * 0.883795087;
        Console.WriteLine(euro);
    }
}
```

Visual Studio Project represents "the **project**" we are working on. In the beginning, these will be our console applications, which we are going to learn writing with the help of the current book, the resources in it and the course Programming Basics in SoftUni. With deeper learning, time and practice, these projects will move into the direction of desktop applications, web applications and other developments. A project in VS **logically groups multiple files** constructing a given application or a component. A **C# project** contains one or more **C# source files**, configuration files and other resources. In every C# source file, there is one or more **definition of types** (classes or other definitions). In **the classes** there are **methods** (actions), and they contain **a sequence of commands**. It sounds complicated, but with bigger projects a structure like this one is very convenient and allows good organization of the work files.

Visual Studio Solution represents a container (a work **solution**), in which **a few projects are logically bound**. The purpose of the binding of these VS Projects is to create an opportunity for the code from any of the projects to collaborate with the code from the rest of the VS projects, to ensure the application or the website to work correctly. When the software product or service that we develop is big, it is built as a **VS Solution**, and this Solution is split into **projects** (VS Projects) and inside each project there are **folders with source files**. This hierarchical organization is much more convenient with more serious projects (let's say over 50 000 lines of code).

For **smaller projects** VS Solutions and VS Projects are **complicating the work**, rather than helping, but you will get used to it quickly.

Example: Creating a Console Application "Hello C#"

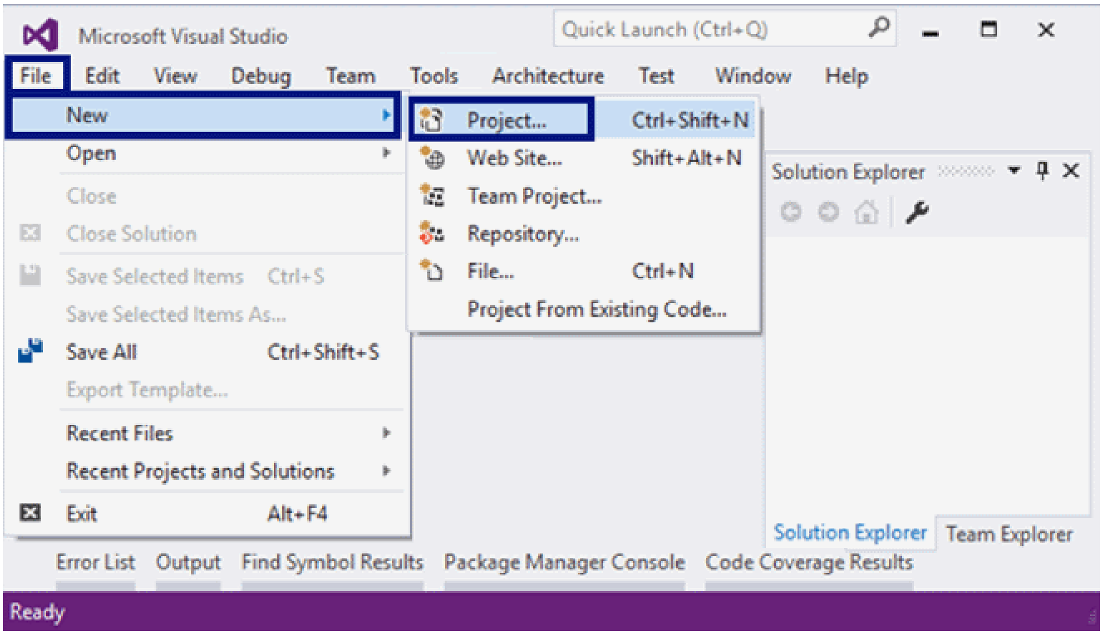
Let's create our first **console program** in Visual Studio. We will start the Visual Studio IDE, will create a new console-based C# project, will write a few lines of C# code and will compile and run the program. Finally, we will submit our C# code for evaluation in the automated Judge system.

Video: Console Application in Visual Studio

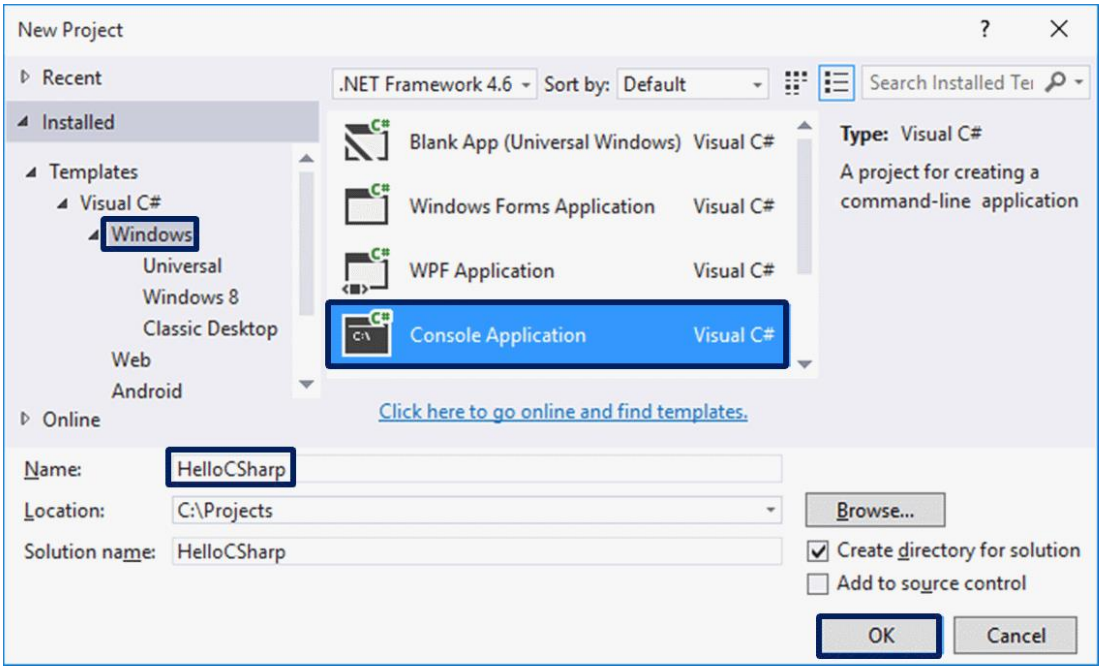
Watch a video lesson about how to create a console app in Visual Studio: https://youtu.be/ecAXCjjk6Nw.

Console App in Visual Studio: Step by Step

We already have Visual Studio and we can start it. Then, we create a new console project: [**File**] → [**New**] → [**Project**] → [**Visual C#**] → [**Windows**] → [**Console Application**].


Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Debug Team Tools Architecture Test Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
Repository...
File... Ctrl+N
Project From Existing Code...
Solution Explorer
Solution Explorer Team Explorer
Error List Output Find Symbol Results Package Manager Console Code Coverage Results
Ready

We set **a meaningful name** to our program, for example `HelloCSharp`:



New Project

Name: HelloCSharp
Location: C:\Projects
Solution name: HelloCSharp

☑ Create directory for solution
☐ Add to source control

Type: Visual C#
A project for creating a command-line application

?
Recent
Installed
Templates
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
Android
Online
.NET Framework 4.6
Sort by: Default
Search Installed Te
Blank App (Universal Windows) Visual C#
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Visual Studio is going to create for us **an empty C# program**, which we have to finish writing (VS Solution with VS Project in it, with C# source file in it, with one C# class in it, with `Main()` method in it).

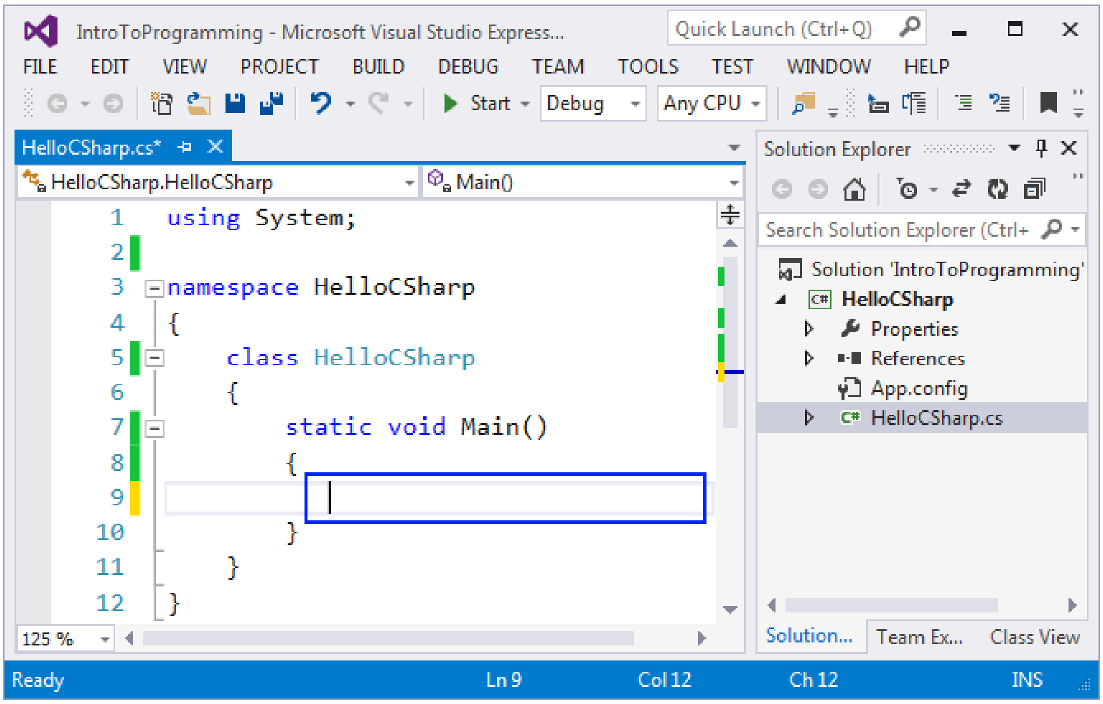
Writing the Program Code

The source code of the C# program is written in the section `Main(string[] args)`, between the opening and the closing parentheses `{ }`. This is the main method (action), that is being executed with the start of a C# program. This `Main()` method can be written in two ways:

- `static void Main(string[] args)` – with parameters from the command line (we are not going into details)
- `static void Main()` – without parameters from the command line.

Both ways are valid, as **the second one is recommended**, because it is shorter and clearer. By default, though, when creating a console application, Visual Studio uses the first way, which we can edit manually if we want to, and delete the part with the parameters `string[] args`.

Press [**Enter**] after **the opening parentheses** `{` and **start writing**. The code of the program is written **inwards**, as this is a part of shaping up the text for convenience during a review and/or debugging.

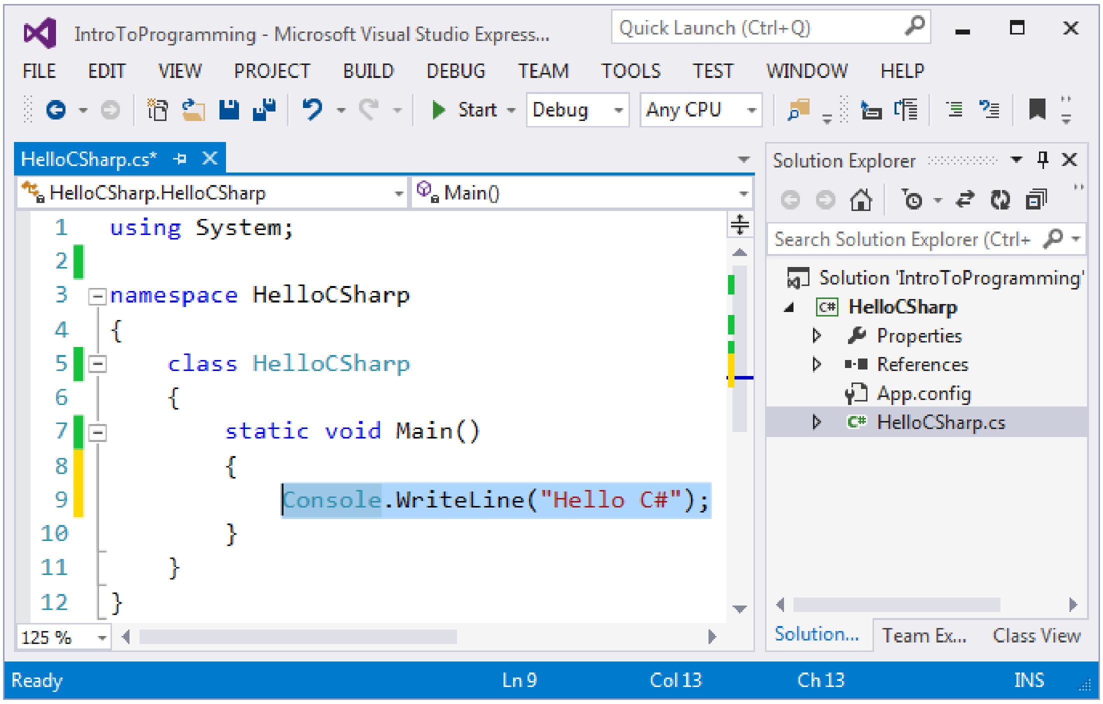

IntroToProgramming - Microsoft Visual Studio Express...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Start Debug Any CPU
HelloCSharp.cs*
HelloCSharp.HelloCSharp
Main()
125 %
Solution Explorer
Search Solution Explorer (Ctrl+
Solution 'IntroToProgramming'
HelloCSharp
Properties
References
App.config
HelloCSharp.cs
Solution... Team Ex... Class View
Ready Ln 9 Col 12 Ch 12 INS

```
using System;

namespace HelloCSharp
{
    class HelloCSharp
    {
        static void Main()
        {

        }
    }
}
```

Write the following command:

```
Console.WriteLine("Hello C#");
```

Here is how our program should look like in Visual Studio:


IntroToProgramming - Microsoft Visual Studio Express...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Start Debug Any CPU
HelloCSharp.cs*
HelloCSharp.HelloCSharp
Main()
125 %
Solution Explorer
Search Solution Explorer (Ctrl+
Solution 'IntroToProgramming'
HelloCSharp
Properties
References
App.config
HelloCSharp.cs
Solution... Team Ex... Class View
Ready Ln 9 Col 13 Ch 13 INS

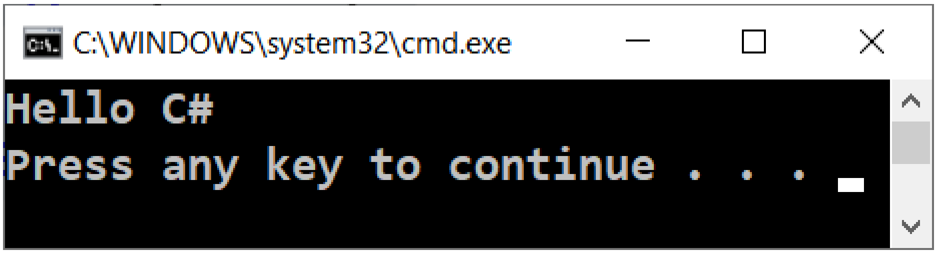
```
using System;

namespace HelloCSharp
{
    class HelloCSharp
    {
        static void Main()
        {
            Console.WriteLine("Hello C#");
        }
    }
}
```

The command `Console.WriteLine("Hello C#")` in the C# language means to execute printing (`WriteLine(…)`) on the console (`Console`) and to print the text message `Hello C#`, which we should surround by quotation marks, in order to clarify that this is a text. In the end of each command in the C# language the symbol `;` is being put and it says that the command ends in that place (it doesn't continue on the next line).

This command is very typical in programming: we say a given object should be found (in this case the console) and some action should be executed upon it (in this case it is printing something that is given inside the brackets). More technically explained, we call the method `WriteLine(…)` from the class `Console` and give as a parameter to it a text literal `"Hello C#"`.

Starting the Program

To start the program, press [Ctrl + F5]. If there aren't any errors, the program will be executed. The result will be written on the console (in the black window):



```
Hello C#
Press any key to continue . . . _
```

Notice that we start it with [Ctrl+F5], and not only [F5] or with the start button in Visual Studio. If we use [F5], the program will run shortly and right afterwards the black window will disappear, and we are not going to see the result.

Actually, the output from the program is the following text message:

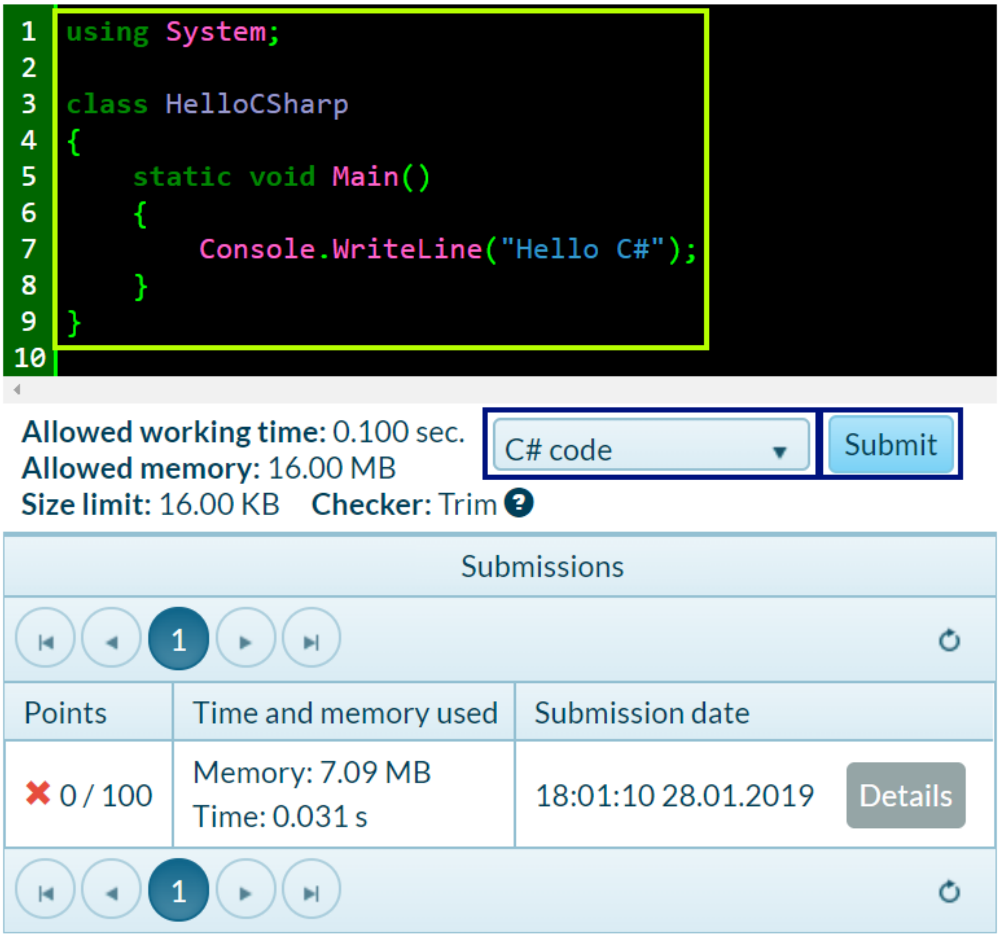
```
Hello C#
```

The message "Press any key to continue . . ." is displayed additionally on the last line on the console after the program ends, in order to push us to see the result from the execution and to press a key to close the console.

Testing the Program in the Judge System

Testing of the problems in this book is automated and is done through the Internet, using the Judge System: https://judge.softuni.org website. The evaluation of the tasks is done immediately by the system. Each task goes through a sequence of tests, as every successfully passed test gives the points assigned for it. The tests that are applied to the tasks are hidden.

We can test the above program here: https://judge.softuni.org/Contests/Practice/Index/503#0. We place the source code from the program in the black field and we choose C# code, the way it is shown:



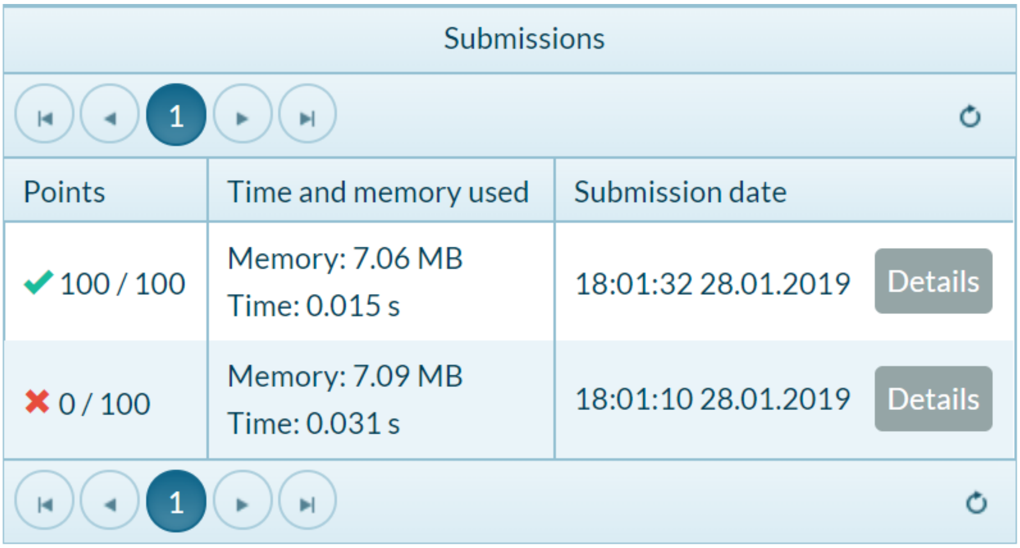
```
using System;

class HelloCSharp
{
    static void Main()
    {
        Console.WriteLine("Hello C#");
    }
}
```

Allowed working time: 0.100 sec.
Allowed memory: 16.00 MB
Size limit: 16.00 KB **Checker:** Trim

Submissions

Points	Time and memory used	Submission date
✗ 0 / 100	Memory: 7.09 MB Time: 0.031 s	18:01:10 28.01.2019 Details

We send our solution for evaluation using the [Send] button. The system gives a result back in a few seconds in the table with sent solutions. When necessary, we can press the button for renewing the results [refresh] in the upper right side of the table with sent solutions:



Submissions

Points	Time and memory used	Submission date
✓ 100 / 100	Memory: 7.06 MB Time: 0.015 s	18:01:32 28.01.2019 Details
✗ 0 / 100	Memory: 7.09 MB Time: 0.031 s	18:01:10 28.01.2019 Details

In the table with the sent solutions the judge system is going to show one of the following **possible results**:

- **Points count** (between 0 and 100), when the submitted code is compiled successfully (there are no syntax errors) and can be tested.
    - When the **solution is correct** all of the tests are marked in green and we get **100 points**.
    - When the **solution is incorrect** some of the tests are marked in red and we get incomplete or 0 points.
- When the program is incorrect, we will get **an error message** upon compiling.

How to Register in SoftUni Judge?

Use your credentials (username + password) for the site **softuni.org**. If you don't have a SoftUni registration, create one. It takes only a minute – a standard registration in an Internet site.

Testing the Programs That Play Notes

Now, after **you know how to run programs**, you can test your example programs that play musical notes. Have some fun, try these programs out. Try to change them and play with them. Change the command **`Console.WriteLine("Hello C#");`** with the command **`Console.Beep(432, 500);`** and start the program. Check if the sound of your computer is on and whether it's turned up. If you work in an online environment, you will not hear a sound, because the program is not executed on your computer, but elsewhere.

Typical Mistakes in C# Programs

Now we will review the **typical mistakes in the C# programs** of the beginners, like missing semicolon, missing quotations mark, missing parenthesis, wrong letter capitalization, etc.

Video: Typical Mistakes in C# Programs

Watch a video lesson about the typical mistakes in the C# programs of the beginners: https://youtu.be/8XwM2AVC0wU.

Writing Outside if the Main Method

One of the common mistakes with beginners is **writing outside the body of the `Main()` method**, because the integrated environment or the compiler can't read the given commands in the program correctly. Here is an example for an incorrectly written program:

```
static void Main(string[] args)
{
}
Console.WriteLine("Hello C#");
```

Wrong Letter Capitalization

Another mistake is switching **capital and small letters**, and these matter for calling the commands and their correct functioning. Here is an example of such a mistake:

```
static void Main(string[] args)
{
    Console.Writeline("Hello C#");
}
```

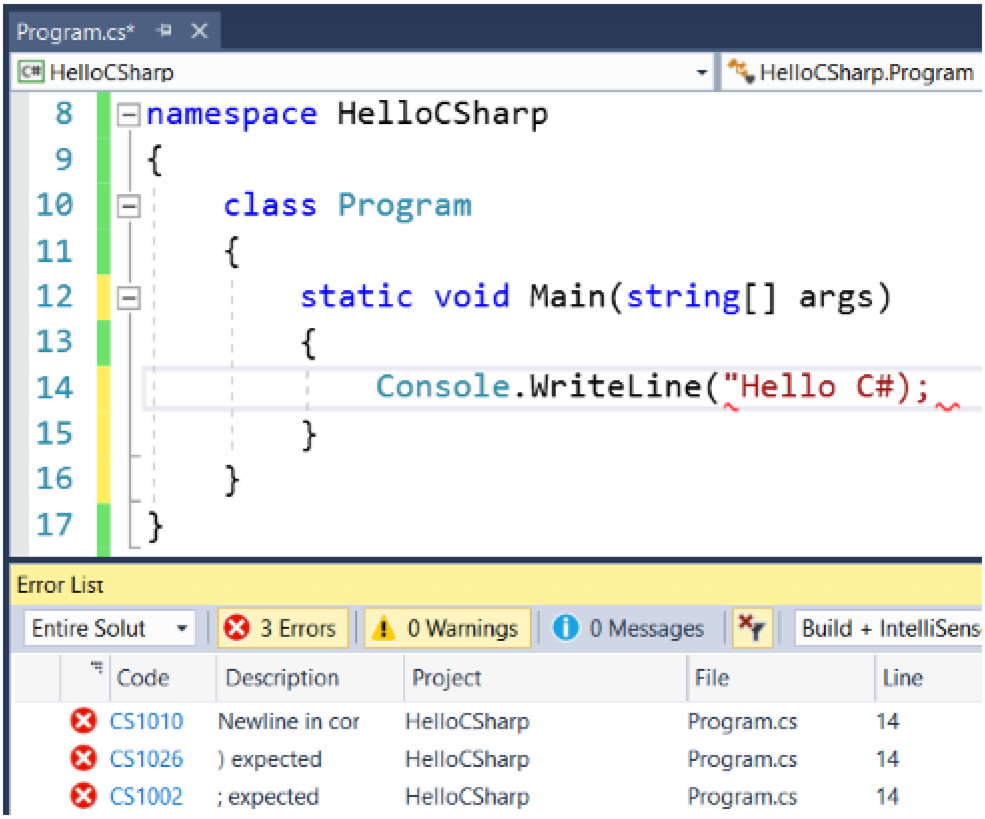
In the example above `Writeline` is written wrong and has to be fixed to `WriteLine`.

Missing Semicolon

The absence of a semicolon (`;`) in the end of the commands is one of the eternal problems of the beginner programmer. Skipping this sign leads to incorrect functioning of the program and often the problem stays unnoticed. Here is an example of a mistaken code:

```
static void Main(string[] args)
{
    Console.Writeline("Hello C#")
}
```

Missing or Wrong Quotation Mark or Parenthesis



```
namespace HelloCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello C#);
        }
    }
}
```

Code	Description	Project	File	Line
CS1010	Newline in cor	HelloCSharp	Program.cs	14
CS1026	) expected	HelloCSharp	Program.cs	14
CS1002	; expected	HelloCSharp	Program.cs	14

Missing quotation mark or the absence of opening or closing parentheses can also turn out to be a problem. Same as the semicolon, here also the problem leads to incorrect functioning of the program or overall to its failure. This mistake is hardly noticeable in a larger code. Here is an example of a program with errors:

```
static void Main(string[] args)
{
    Console.WriteLine("Hello C#);
}
```

This program will throw **a compile time error** and the build is going to fail, and even before that the code will become underlined, in order to point the programmer to the mistake that they'd made (the missing closing quotation mark):

Another example is missing `{` or `}`. It may produce unexpected error messages, not always easy to understand.

```
class Example
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello C#");
}
```

Exercises: First Steps in Coding

Welcome to the **exercises**. Now we are going to write a couple of console applications, by which we are going to make a few more steps into programming. After that we will show how we can program something more complex – programs with graphical user interface.

Video: Summary

Watch the following short video to summarize what we learned about coding in this publication: https://youtu.be/GstN43-eN2g.

What We Learned?

First, we have learned **what is programming** – **giving commands, written in a computer language**, which the machine understands and is able to execute. We understood what a **computer program** is – it represents a **sequence of commands**, arranged one after another. We got familiar with **the language for programming C#** on a base level and how **to create simple console applications** with Visual Studio. We followed **the structure of the programming code in the C# language**, as for example, the fact that commands are mainly given in the section `static void Main(string[] args)` between **the opening and closing curly parentheses**. We saw how to print with `Console.WriteLine(…)` and how to start our program with [**Ctrl + F5**]. We learned how to test our code in **SoftUni Judge**.

The Exercises

Let's get started with the **exercises**. You didn't forget that programming is learned by writing a lot of code and solving problems, did you? Let's solve a few problems to confirm what we have learned.

Problem: Expression

Write a console-based C# console program that **calculates** and **prints** the value of the following numerical expression:

```
(3522 + 52353) * 23 - (2336 * 501 + 23432 - 6743) * 3
```

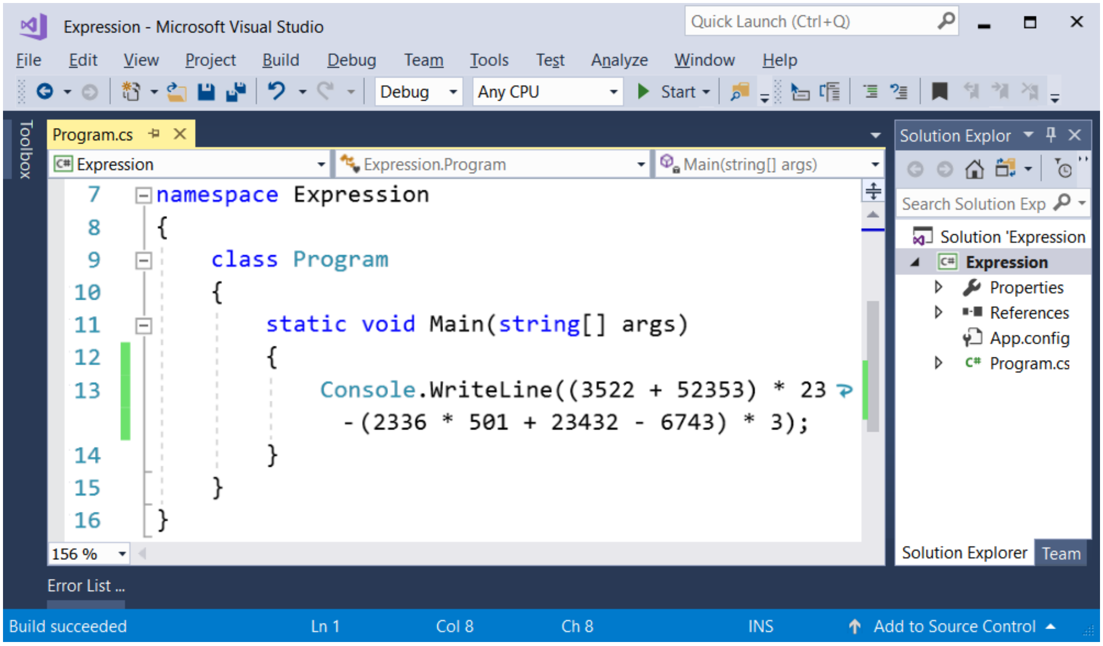
Note: **it is not allowed to previously calculate the value** (for example with Windows Calculator).

Video: Problem "Expression"

Watch a video lesson to learn how to solve the "Expression" problem step by step, with concise explanations: https://youtu.be/JEbwS2xtbLw.

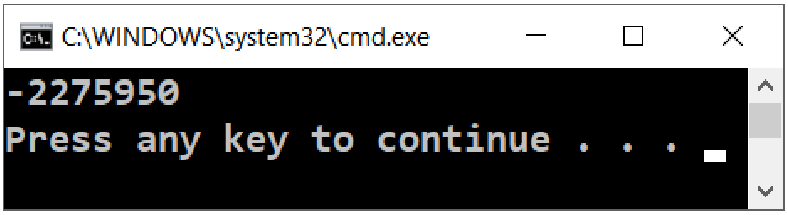
Hints and Guidelines

Create **a new C# console project** with title "**Expression**". Find the method `static void Main(string[] args)` and **go into its body** between `{` and `}`. After that, **write the code** that calculates the above numerical expression and prints its value on the console. Put the above numerical expression inside the brackets of the command `Console.WriteLine(…)`:



```
namespace Expression
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine((3522 + 52353) * 23
              - (2336 * 501 + 23432 - 6743) * 3);
        }
    }
}
```

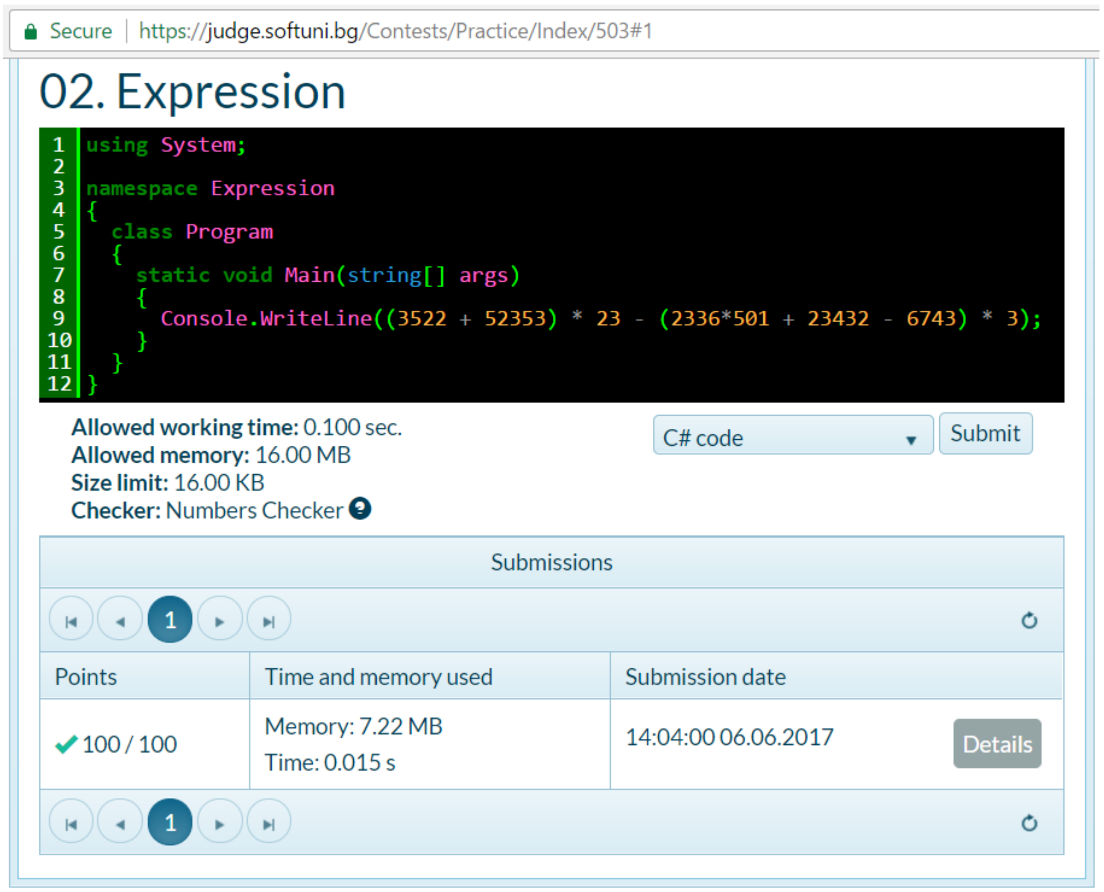
Start the program with [Ctrl+F5] and check if the result is the same as the one in the picture:



```
-2275950
Press any key to continue . . .
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/503#1.


Secure | https://judge.softuni.bg/Contests/Practice/Index/503#1

02. Expression

```
using System;

namespace Expression
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine((3522 + 52353) * 23 - (2336*501 + 23432 - 6743) * 3);
    }
  }
}
```

Allowed working time: 0.100 sec.
Allowed memory: 16.00 MB
Size limit: 16.00 KB
Checker: Numbers Checker

Submissions

Points	Time and memory used	Submission date
✓ 100 / 100	Memory: 7.22 MB Time: 0.015 s	14:04:00 06.06.2017

Problem: Numbers from 1 to 20

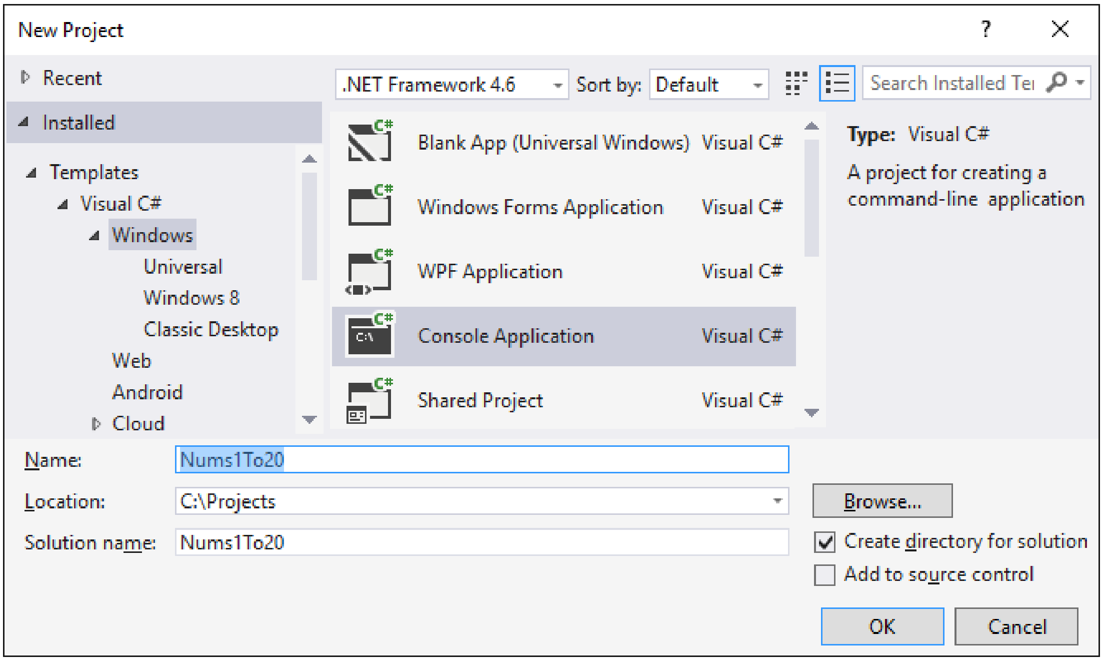
Write a C# console program that **prints the numbers from 1 to 20** on separate lines on the console.

Video: Problem "Numbers from 1 to 20"

Watch a video lesson to learn how to solve the "Numbers from 1 to 20" problem step by step: https://youtu.be/8Qne7CBM2SQ.

Hints and Guidelines

Create **a C# console application** with name "`Nums1To20`":


New Project

.NET Framework 4.6 | Sort by: Default | Search Installed Templates

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows
        - Universal
        - Windows 8
        - Classic Desktop
      - Web
      - Android
      - Cloud

Blank App (Universal Windows)	Visual C#
Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Shared Project	Visual C#

Type: Visual C#

A project for creating a command-line application

Name: Nums1To20

Location: C:\Projects

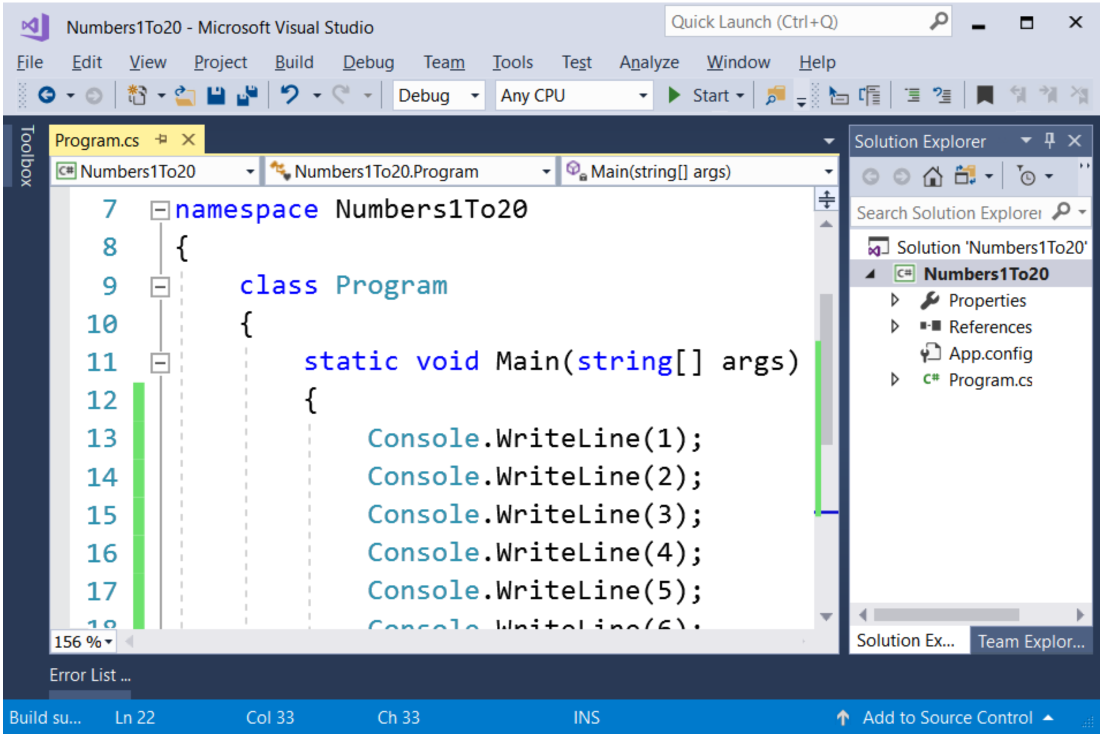
Solution name: Nums1To20

[✓] Create directory for solution

[ ] Add to source control

Browse... OK Cancel

Inside the `static void Main()` method write 20 commands `Console.WriteLine(…)`, each on a separate line, in order to print the numbers from 1 to 20 one after another. Some of you may be wondering if there is a smarter way. Relax, there is, but we will mention it later on.


Numbers1To20 - Microsoft Visual Studio

Program.cs | Numbers1To20 | Numbers1To20.Program | Main(string[] args)

```
namespace Numbers1To20
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(1);
            Console.WriteLine(2);
            Console.WriteLine(3);
            Console.WriteLine(4);
            Console.WriteLine(5);
            Console.WriteLine(6);
```

Solution Explorer

- Solution 'Numbers1To20'
  - Numbers1To20
    - Properties
    - References
    - App.config
    - Program.cs

Build su... Ln 22 Col 33 Ch 33 INS Add to Source Control

Now **we start the program** and we check if the result is what it is supposed to be:

```
1
2
…
20
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/503#2.

Now think whether we can write the program **a smarter way**, so we don't repeat the same command 20 times. Seek out information on the Internet about "for loop C#".

Problem: Triangle of 55 Stars

Write a C# console program that **prints a triangle made of 55 stars** on 10 lines:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Video: Problem "Triangle of 55 Stars"

Watch a video lesson to learn how to solve the "Triangle of 55 Stars" problem step by step: https://youtu.be/BfITRoOQYLA.

Hints and Guidelines

Create **a new console C# application** with name “`TriangleOf55Stars`”. Inside it, write code that prints the triangle of stars, for example through 10 commands, as the ones pointed out below:

```
Console.WriteLine("*");
Console.WriteLine("**");
…
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/503#3.

Try to **improve your solution**, so that it doesn't have many repeating commands. Could it be done with a `for` loop? Did you find a smart solution (for example with a loop) of the previous task? With this task you can also use something similar, but a bit more complex (two loops, one inside the other). If you don't succeed, there is no problem, we will be learning soon and you will be reminded of this task then.

Problem: Calculate Rectangle Area

Write a C# program that **reads** from the console **two numbers, a and b**, **calculates** and **prints** the area of a rectangle with sides **a** and **b**.

Sample Input and Output

a	b	area
2	7	14
7	8	56
12	5	60

Video: Problem "Rectangle Area"

Watch a video lesson to learn how to solve the "Rectangle Area" problem step by step: https://youtu.be/6fwNJ5k9zTE.

Hints and Guidelines

Create a new **console C# program**. To **read both of numbers**, use the following commands:

```
static void Main(string[] args)
{
    var a = decimal.Parse(Console.ReadLine());
    var b = decimal.Parse(Console.ReadLine());
    // TODO: calculate the area and print it at the console
}
```

What remains is to finish the program above, to calculate the area of the rectangle and to print it. Use the command that is already known to us `Console.WriteLine()` and put inside its brackets the multiplication of the numbers **a** and **b**. In programming, multiplication is done using the operator `*`.

Test Your Solution

Test your solution with a few examples. You have to get an output, similar to this one (we enter 2 and 7 as input and the program prints result 14 – their multiplication):

```
2
7
14
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/503#4.

* Problem: A Square Made of Stars

Write a C# console program that **reads** from the console **an integer N** and **prints** on the console **a square made out of N stars**, like in the examples below.

Sample Input and Output

Input	Output	Input	Output	Input	Output

3	`***`
	`* *`
	`***`
		4	`****`
			`*  *`
			`*  *`
			`****`
				5	`*****`
					`*   *`
					`*   *`
					`*   *`
					`*****`

Video: Problem "Square of Stars"

Watch a video lesson to learn how to solve the "Square of Stars" problem step by step: https://youtu.be/zaj-DRbaHal.

Hints and Guidelines

Create a new **console C# program**. To read the number N (2 ≤ N ≤100), use the following code:

```
static void Main(string[] args)
{
    var n = int.Parse(Console.ReadLine());
    // TODO: Print the rectangle
}
```

Finish the program above, so that it prints a square, made out of stars. It might be necessary to use `for` loops. Look for information on the Internet.

Attention: this task is harder than the rest and is given now on purpose, and it's marked with a star, in order to provoke you to look for information on the Internet. This is one of the most important skills that you have to develop while you're learning programming: **looking for information on the Internet**. This is what you're going to do every day, if you work as a developer, so don't be scared, try it out. If you have any difficulties, you can also ask for help in the SoftUni's discussion forum: http://forum.softuni.org.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/503#5.

Lab: Graphical and Web Applications

Now we are about to build one simple **web application** and one simple **graphical application**, in order to take a look at what we will be able to create when we progress with programming and software development. We are not going to look through the details about the used techniques and constructions fundamentally. Rather than that, we are just going to take a look at the arrangement and functionality of our creation. After we progress with our knowledge, we will be able to do bigger and more complex software applications and systems. We hope that the examples given below will **straighten your interest**, rather than make you give up.

Console, Graphical and Web Applications

With console applications, as you can figure out yourselves, all operations for reading input and printing output are done through the console. The input data is inserted in the console, which is then read by the application, also in it, and the output data is printed on the console after or during the runtime of the program.

While a console application uses the text console, web applications use web-based user interface. To execute them, two things are needed – a web server and a web browser, as the browser plays the main role in the visualization of the data and the interaction with the user. Web applications are much more pleasant for the user, they visually look better, and a mouse and touch screen can be used (for tablets and smartphones), but programming stands behind all of that. And this is why we have to learn to program and we have already made our first very little steps towards that.

Graphical (GUI) applications have a visual user interface, directly into your computer or mobile device, without a web browser. Graphical applications (also known as desktop applications) contain one or more graphical windows, in which certain controllers are located (text fields, buttons, pictures, tables and others), serving for dialog with the user in a more intuitive way. Similar to them are the mobile applications in your telephone or your tablet: we use forms, text fields, buttons and other controls and we control them by programming code. This is why we are learning how to write code now: the code is everywhere in software development.

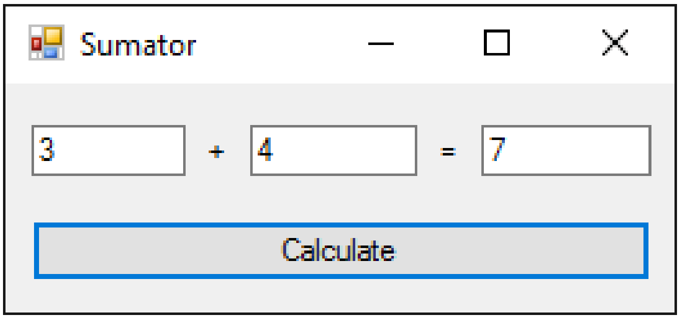
Exercises: GUI and Web Applications

In the next exercises we will create a GUI and a Web application:

- Graphical Application "Sumator" (Calculator)
- Web Application "Sumator" (Calculator)

Lab: Graphical Application "Sumator" (Calculator)

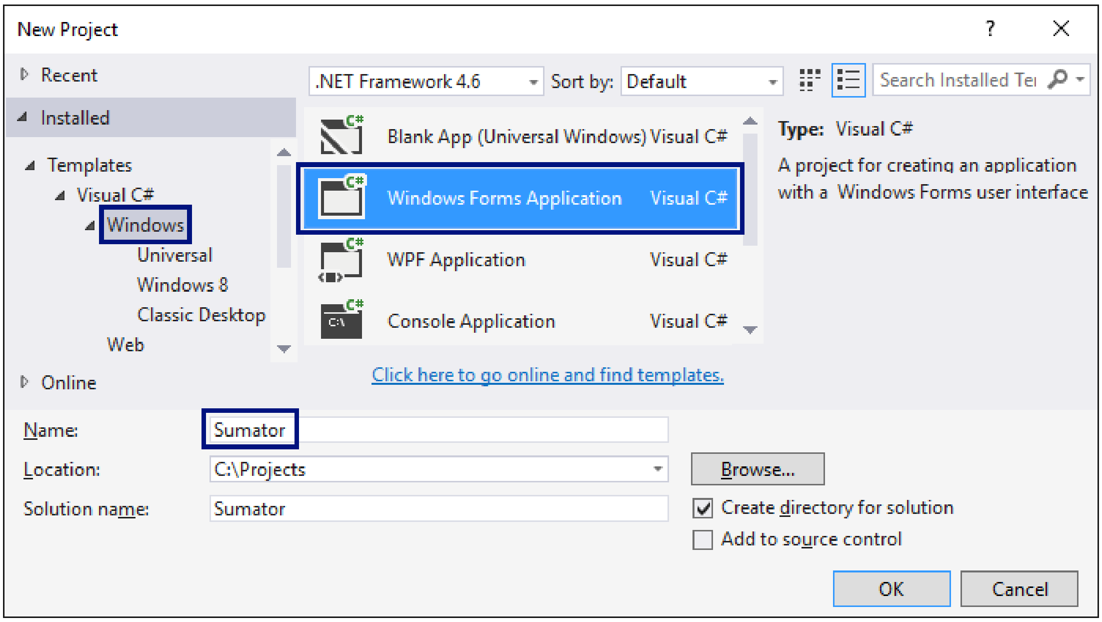
Write a graphical (GUI) application, which calculates the sum of two numbers:



By entering two numbers in the first two fields and pressing the button [Calculate] their sum is being calculated and the result is shown in the third text field. For our application we will use the Windows Forms technology, which allows the creation of graphical applications for Windows, in the development environment Visual Studio and with programming language C#.

Creating a New C# Project

In Visual Studio we create a new C# project of type “Windows Forms Application”:


New Project
?
Recent
Installed
Templates
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
Online
Search Installed Te
Blank App (Universal Windows) Visual C#
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

.NET Framework 4.6

Sort by: Default

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: Sumator

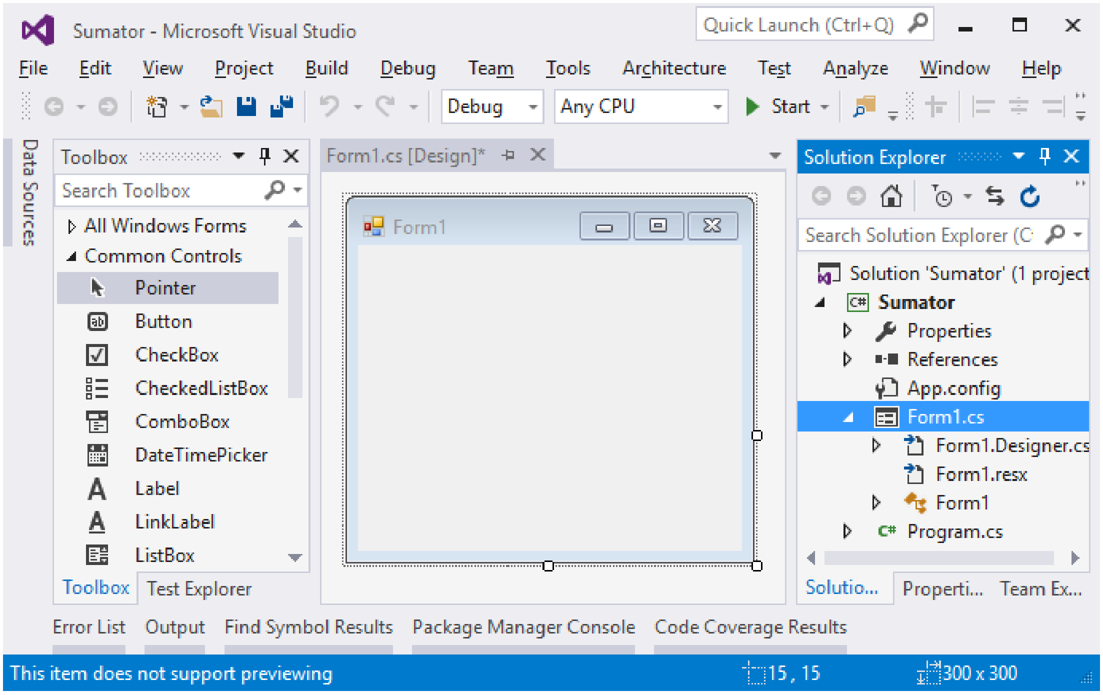
Location: C:\Projects

Solution name: Sumator

[✓] Create directory for solution

[ ] Add to source control

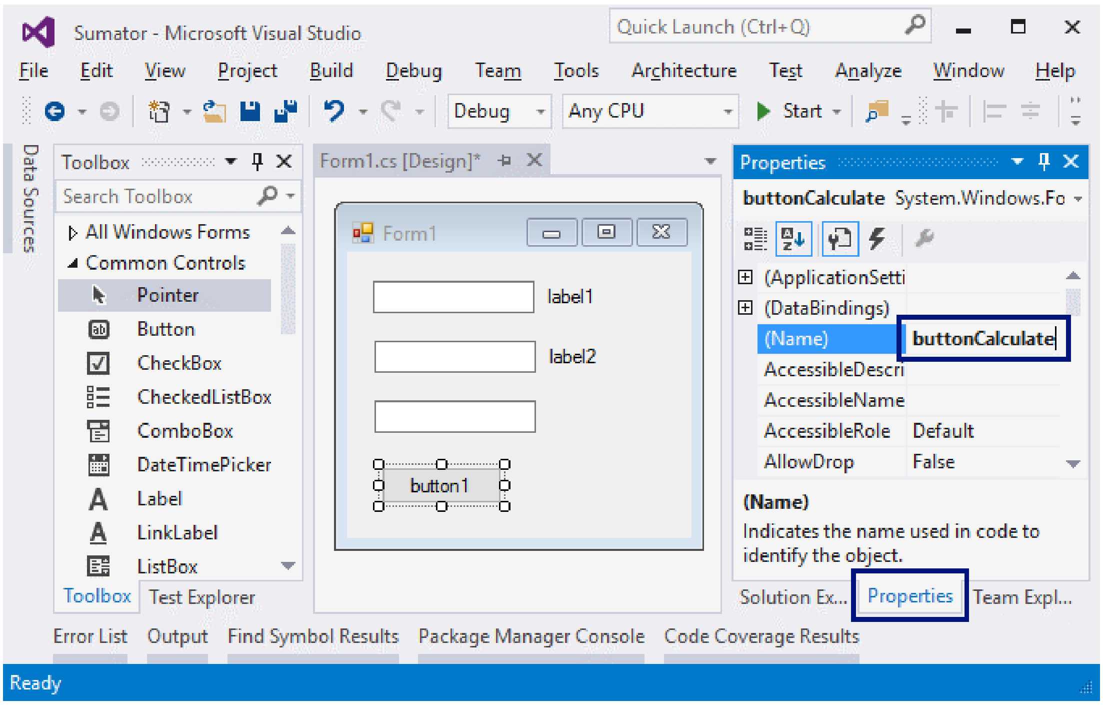
When creating a Windows Forms application **an editor for user interface** will be shown, in which **different visual elements** could be put (for example text boxes and buttons):



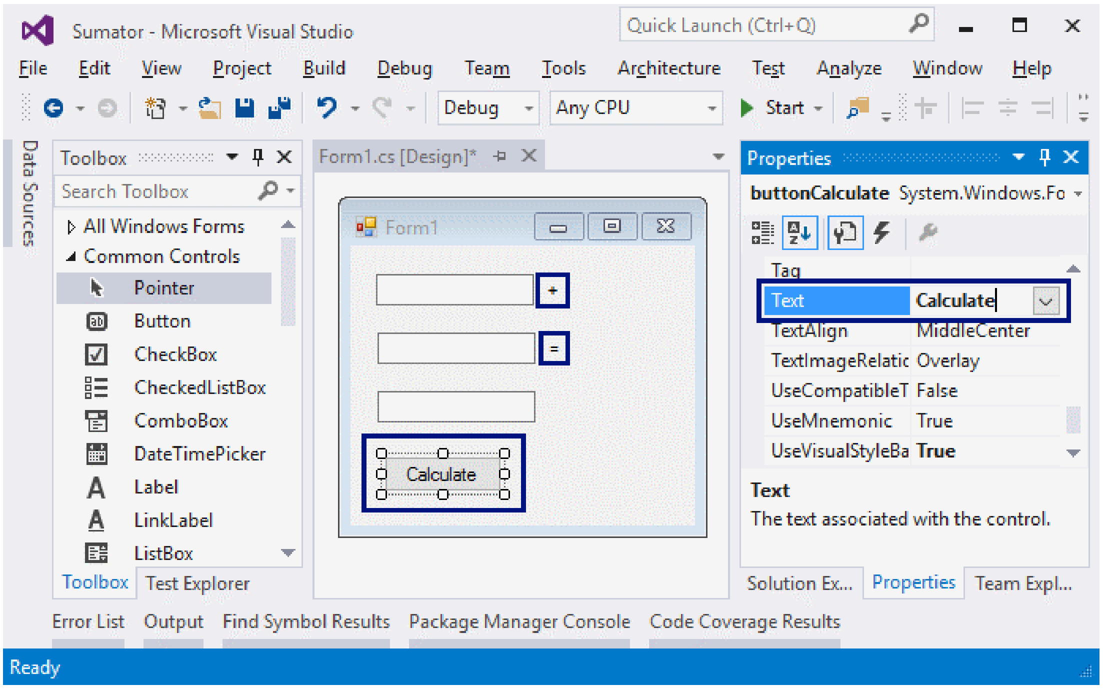
This item does not support previewing

Adding Text Fields and a Button

We download from the bar on the left (Toolbox) **three text boxes** (`TextBox`), **two labels** (`Label`) and **a button** (`Button`), afterwards we arrange them in the window of the application. Then we **change the names of each of the controls**. This is done from **the window "Properties"** on the right, by changing the field (`Name`):



- Names of the text boxes: `textBox1`, `textBox2`, `textBoxSum`
- Name of the button: `buttonCalculate`
- Name of the form: `FormCalculate`

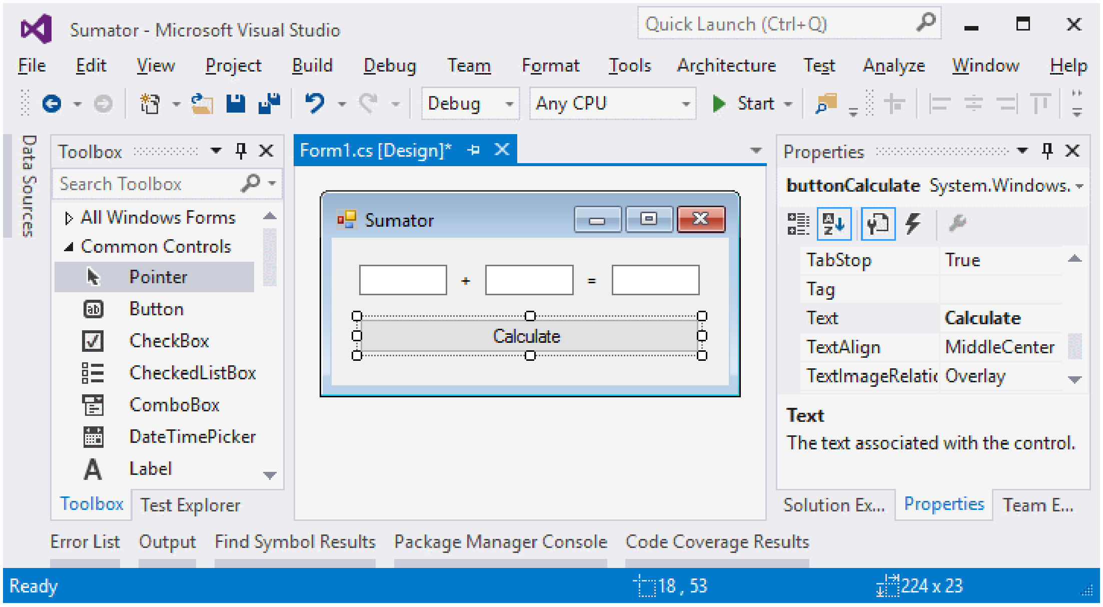

Sumator - Microsoft Visual Studio
Toolbox
Form1.cs [Design]*
Form1
+
=
Calculate
Properties
buttonCalculate System.Windows.Fo
Text Calculate
TextAlign MiddleCenter
Text
The text associated with the control.
Ready

We change the headings (the `Text` property) of the controls:

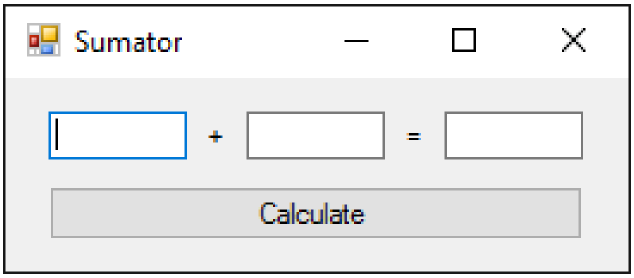
- `buttonCalculate` -> `Calculate`
- `label1` -> `+`
- `label2` -> `=`
- `Form1` -> `Sumator`

Resizing the Controls and Starting the Application

We resize and arrange the controls, to make them look better:

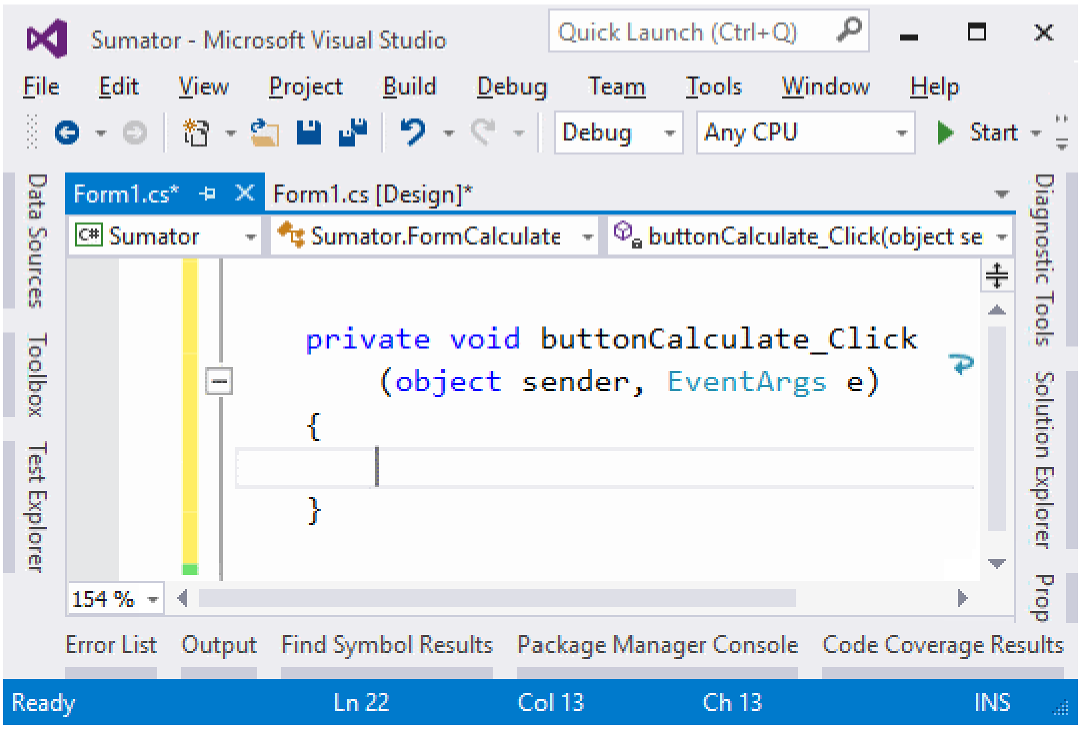


We try to run the application [**Ctrl+F5**]. It should start, but it should **not function completely**, because we haven't written what happens when we click the button yet.



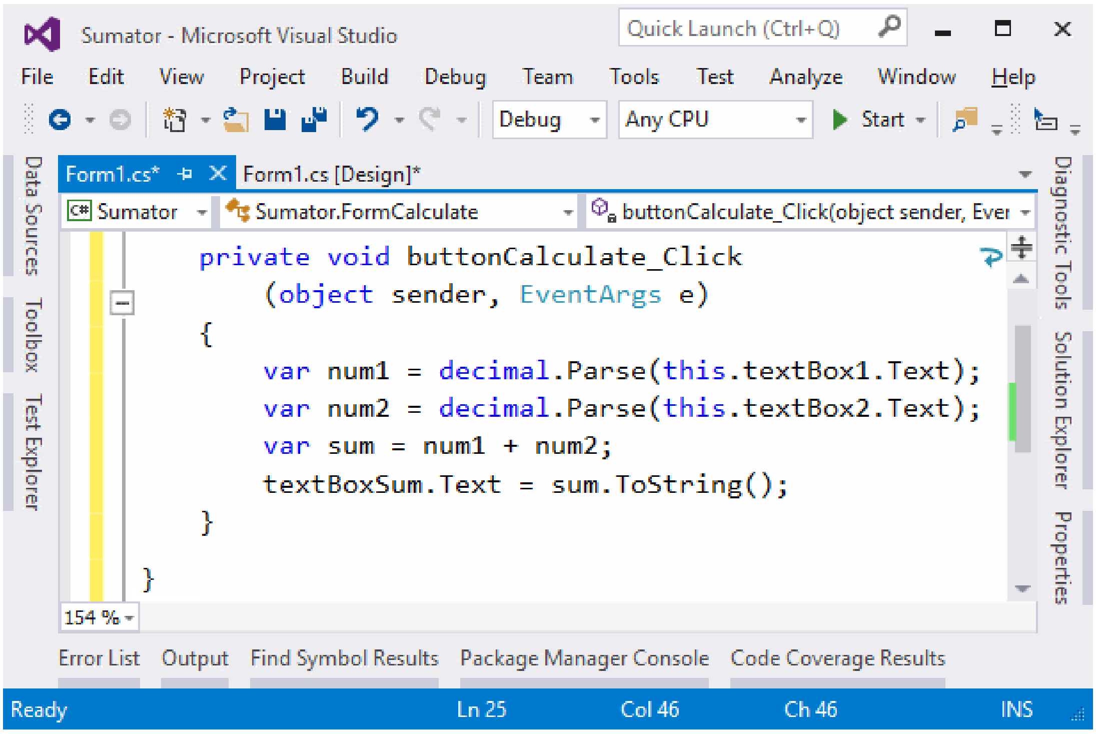
Writing the Program Code

Now it is time to write the code, which **sums the numbers** from the first two fields and **shows the result** in the third field. For this purpose, we double click **the [Calculate] button**. The place, in which we write what is going to happen by clicking the button will be shown:


Sumator - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Window Help
Debug
Any CPU
Start
Form1.cs*
Form1.cs [Design]*
Sumator
Sumator.FormCalculate
buttonCalculate_Click(object se
Data Sources
Toolbox
Test Explorer
Diagnostic Tools
Solution Explorer
Prop
154 %
Error List Output Find Symbol Results Package Manager Console Code Coverage Results
Ready Ln 22 Col 13 Ch 13 INS

```
private void buttonCalculate_Click
    (object sender, EventArgs e)
{

}
```

We write the following C# code between the opening and the closing brackets `{ }`, where the cursor is:

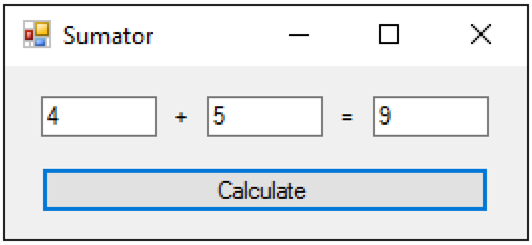

Sumator - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Form1.cs*
Form1.cs [Design]*
Sumator
Sumator.FormCalculate
buttonCalculate_Click(object sender, Ever
Data Sources
Toolbox
Test Explorer
Diagnostic Tools
Solution Explorer
Properties
154 %
Error List Output Find Symbol Results Package Manager Console Code Coverage Results
Ready Ln 25 Col 46 Ch 46 INS

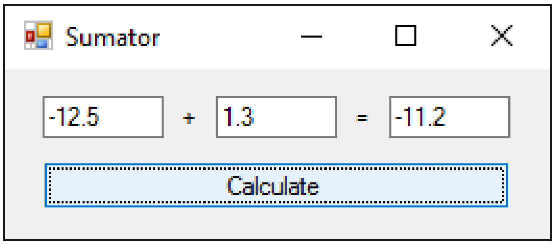
```
private void buttonCalculate_Click
    (object sender, EventArgs e)
{
    var num1 = decimal.Parse(this.textBox1.Text);
    var num2 = decimal.Parse(this.textBox2.Text);
    var sum = num1 + num2;
    textBoxSum.Text = sum.ToString();
}
}
```

This code **takes the first number** from the field `textBox1` and keeps it **in the variable `num1`**, keeps **the second number** from the field `textBox2` in **the variable `num2`**, afterwards it **sums `num1` and** `num2` in the variable `sum` and in the end takes the text value of the variable `sum` in the field `textBoxSum`.

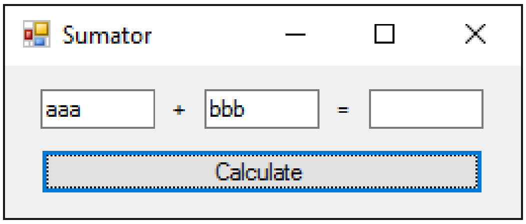
Testing the Application

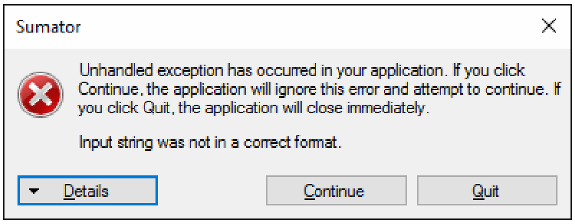
We start the program again with [Ctrl+F5] and we check whether it works correctly. We try to calculate 4 + 5, and afterwards -12.5 + 1.3:





We try with invalid numbers, for example: "aaa" and "bbb". It seems there is a problem:




Sumator

Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

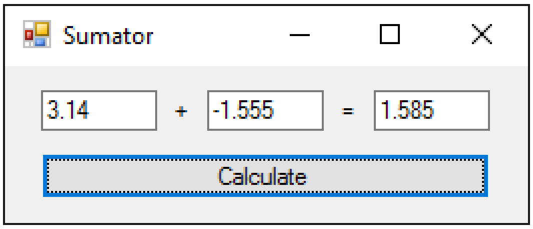
Input string was not in a correct format.

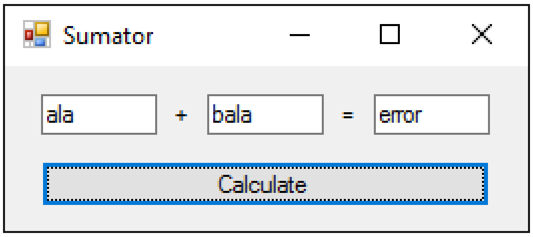
Fixing the Bug and Retesting the Application

The problem comes from the conversion of the text field into a number. If the value inside the field is not a number, the program throws an exception. We can rewrite the code in order to fix this problem:

```
private void buttonCalculate_Click
    (object sender, EventArgs e)
{
    try
    {
        var num1 = decimal.Parse(this.textBox1.Text);
        var num2 = decimal.Parse(this.textBox2.Text);
        var sum = num1 + num2;
        textBoxSum.Text = sum.ToString();
    }
    catch (Exception)
    {
        textBoxSum.Text = "error";
    }
}
```

The code above **catches the errors when working with numbers** (it catches exceptions) and in case of an error **it gives a value `error`** in the field with the result. We start the program again with [Ctrl+F5] and try if it works. This time **by entering a wrong number the result is `error`** and the program doesn't break:





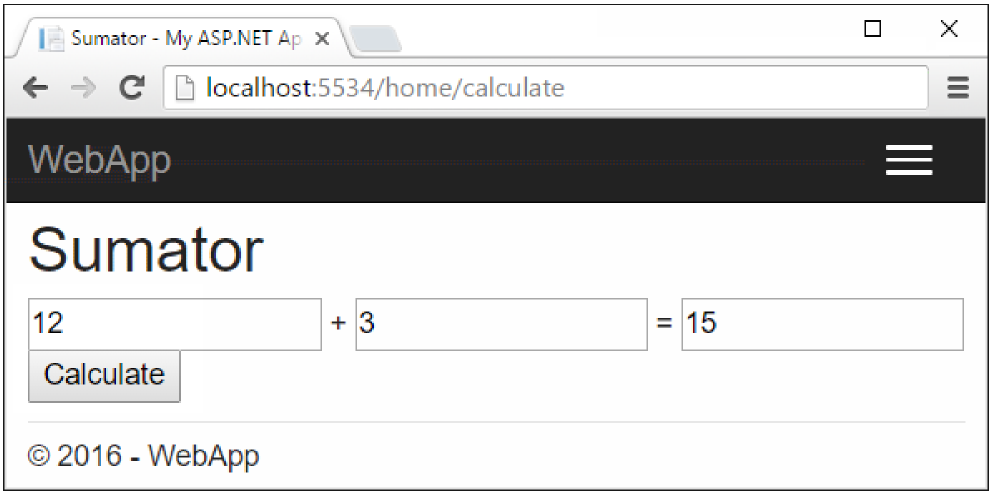
Is it complicated? It is normal to seem complex, of course. We are just beginning to get into programming. The example above requires much more knowledge and skills, which we are going to develop through this book and even afterwards. Just allow yourself to have some fun with desktop programming. If something doesn't work, watch **the video in the beginning of this publication** or ask in **the SoftUni forum**: http://forum.softuni.org. A time will come when it is going to be easy for you, but you really have to put **an effort and be persistent**. Learning programming is a slow process with lots and lots of practice.

Lab: Web Application "Sumator" (Calculator)

Now we are going to create something even more complex, but also more interesting: Web application that **calculates the sum of two numbers**. By **entering two numbers** in the first two text fields and pressing the [**Calculate**] button, **their sum is being calculated** and the result is shown in the third text field.

Pay attention that we are creating **a Web-based application**. This is an application that is available through a web browser, just like your favorite email or news website. The web application is going to have a server side (back-end), which is written in the C# language with the ASP.NET MVC technology, and a client side (front-end), which is written in the HTML language (this is a language for visualization of information in a web browser).

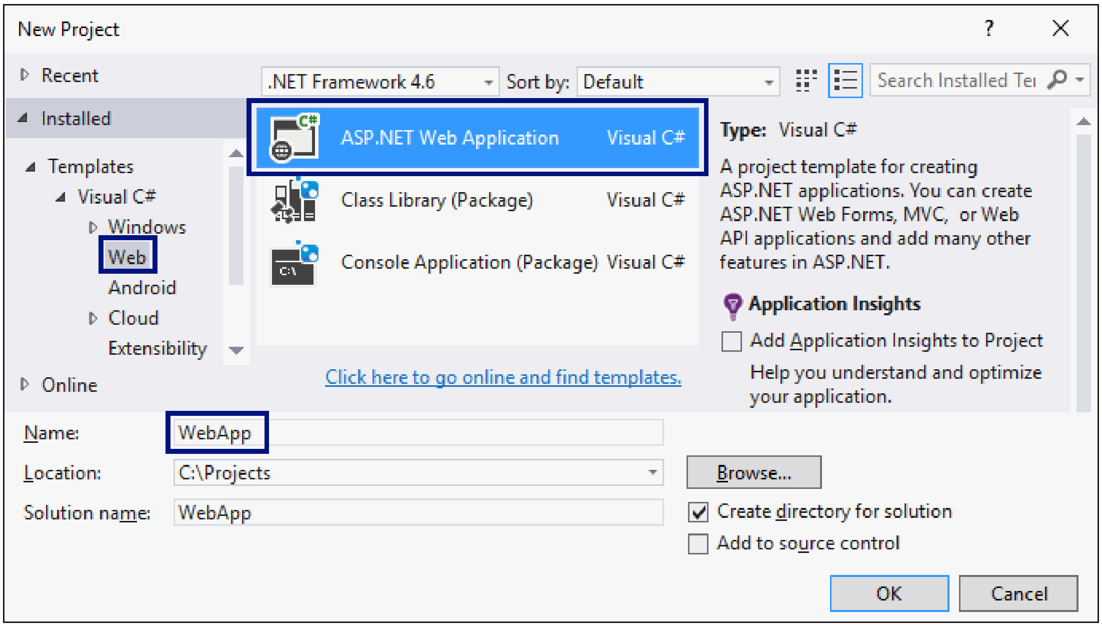
The **Web application** is expected to look similarly to the following:



As a difference compared to console applications, which read and write the data in the form of a text on the console, Web applications have **a Web-based user interface**. Web applications **are being loaded from some Internet address** (URL) through a standard web browser. Users write input data in a page, visualized from the web browser, the data is processed on a web server and the results are shown again in a page of the web browser. For our web application we are going to use **the ASP.NET MVC technology**, which allows creating of **web applications with the programming language C#** in the development environment **Visual Studio**.

Creating a New ASP.NET MVC Project

In VS we create **a new C# project of type "ASP.NET Web Application"** with name **WebApp**:


New Project

Installed > Templates > Visual C# > Web

.NET Framework 4.6 | Sort by: Default

ASP.NET Web Application — Visual C#
Class Library (Package) — Visual C#
Console Application (Package) — Visual C#

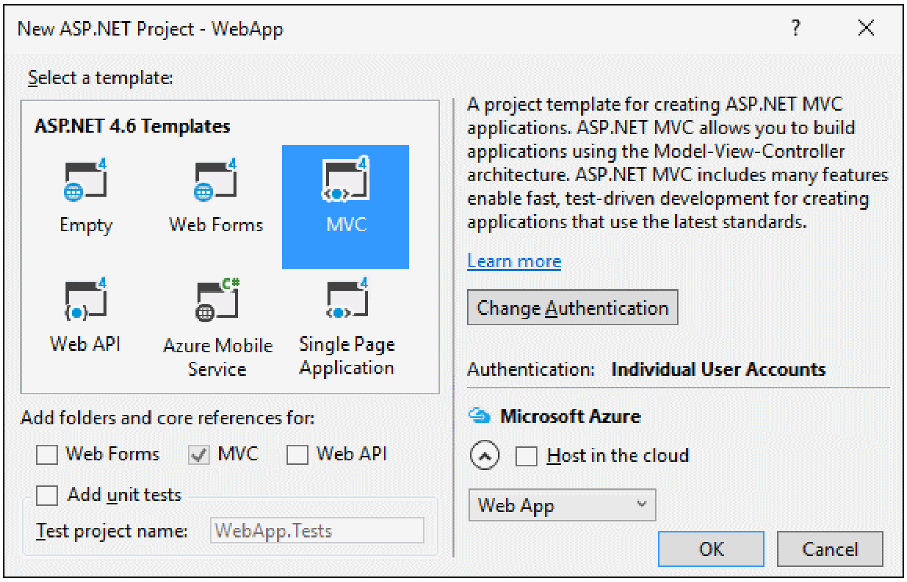
Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights
☐ Add Application Insights to Project
Help you understand and optimize your application.

Name: WebApp
Location: C:\Projects
Solution name: WebApp
☑ Create directory for solution
☐ Add to source control

We choose as **type** of the application: "**MVC**":


New ASP.NET Project - WebApp

Select a template:

ASP.NET 4.6 Templates: Empty, Web Forms, MVC, Web API, Azure Mobile Service, Single Page Application

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features enable fast, test-driven development for creating applications that use the latest standards.

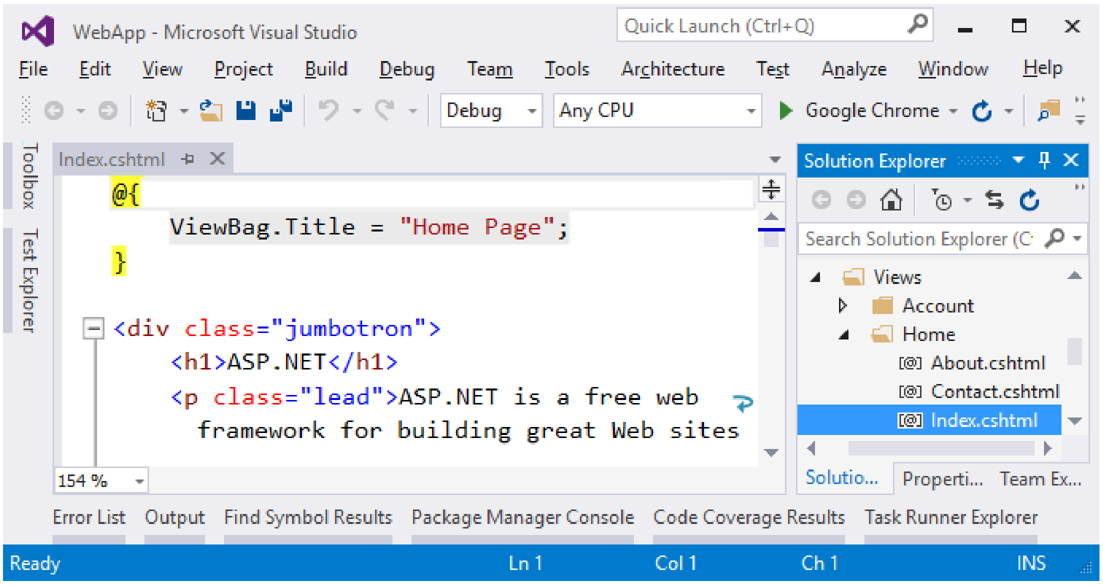
Authentication: Individual User Accounts

Add folders and core references for:
☐ Web Forms ☑ MVC ☐ Web API
☐ Add unit tests
Test project name: WebApp.Tests

Microsoft Azure
☐ Host in the cloud
Web App

Creating a View (Web Form)

We find the file `Views\Home\Index.cshtml`. **The view of the home page** of our web application is inside it:


WebApp - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Debug
Any CPU
Google Chrome
Index.cshtml
Toolbox
Test Explorer
Solution Explorer
Search Solution Explorer (C
Views
Account
Home
About.cshtml
Contact.cshtml
Index.cshtml
Solutio... Properti... Team Ex...
154 %
Error List Output Find Symbol Results Package Manager Console Code Coverage Results Task Runner Explorer
Ready Ln 1 Col 1 Ch 1 INS

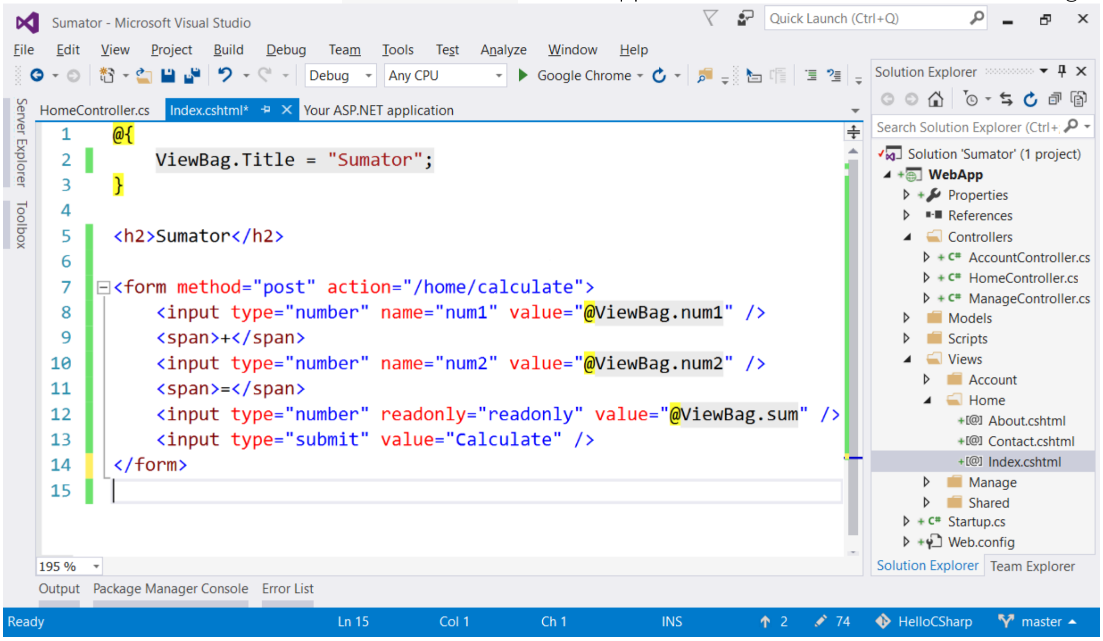
```
@{
    ViewBag.Title = "Home Page";
}

<div class="jumbotron">
    <h1>ASP.NET</h1>
    <p class="lead">ASP.NET is a free web
      framework for building great Web sites
```

We delete the old code from **the file `Index.cshtml`** and write the following code:

```
@{
    ViewBag.Title = "Sumator";
}

<h2>Sumator</h2>

<form method="post" action="/home/calculate">
    <input type="number" name="num1" value="@ViewBag.num1" />
    <span>+</span>
    <input type="number" name="num2" value="@ViewBag.num2" />
    <span>=</span>
    <input type="number" readonly="readonly" value="@ViewBag.sum" />
    <input type="submit" value="Calculate" />
</form>
```

This code **creates a web form with three text boxes and a button in it**. Inside the fields, values are being loaded, which are calculated previously in the object **`ViewBag`**. The requirement says that with the click of the [**Calculate**] button the action **`/home/calculate`** (**action `calculate` from the `home` controller**) will be called.

Here is how the file `Index.cshtml` is supposed to look after the change:


Sumator - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug Any CPU Google Chrome
HomeController.cs Index.cshtml* Your ASP.NET application
Solution Explorer
Solution 'Sumator' (1 project)
WebApp
Properties
References
Controllers
AccountController.cs
HomeController.cs
ManageController.cs
Models
Scripts
Views
Account
Home
About.cshtml
Contact.cshtml
Index.cshtml
Manage
Shared
Startup.cs
Web.config
Output Package Manager Console Error List
Ready Ln 15 Col 1 Ch 1 INS 2 74 HelloCSharp master

```
@{
    ViewBag.Title = "Sumator";
}

<h2>Sumator</h2>

<form method="post" action="/home/calculate">
    <input type="number" name="num1" value="@ViewBag.num1" />
    <span>+</span>
    <input type="number" name="num2" value="@ViewBag.num2" />
    <span>=</span>
    <input type="number" readonly="readonly" value="@ViewBag.sum" />
    <input type="submit" value="Calculate" />
</form>

```

Writing the Program Code

What remains is to write the action that sums the numbers when clicking the button [Calculate]. We open the file `Controllers\HomeController.cs` and we add the following code into the body of `HomeController` class:

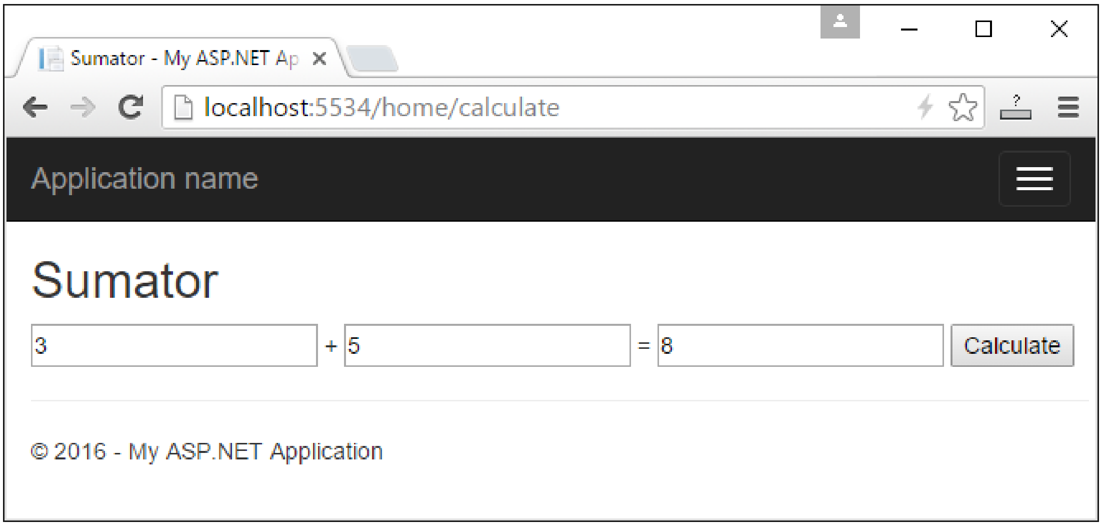
```
public ActionResult Calculate(int num1, int num2)
{
    this.ViewBag.num1 = num1;
    this.ViewBag.num2 = num2;
    this.ViewBag.sum = num1 + num2;
    return View("Index");
}
```

This code implements the "calculate" action. It takes two parameters `num1` and `num2` and records them in the objects `ViewBag`, after which it calculates and records their sum. The values recorded in `ViewBag` are then used from the view, in to be shown in the three text fields inside the form for summing numbers in the web page of the application.

Here is how the file `HomeController.cs` should look after the change:

Testing the Application

The application is ready. We can start it with [Ctrl+F5] and test whether it works:


Sumator - My ASP.NET Ap
localhost:5534/home/calculate
Application name
Sumator
3
+
5
=
8
Calculate
© 2016 - My ASP.NET Application

Does it look scary? Don't be afraid! We have a lot more to learn, to reach the level of knowledge and skills to write web-based applications freely like in the example above, as well as much bigger and much more complex ones. If you don't succeed, there is nothing to worry about, keep moving on. After some time, you will remember with a smile how incomprehensible and exiting your first collision with web programming was.

The purpose of both of the above examples (graphical desktop application and web application) is not to teach you, but to make you dive a little deeper into programming, to enhance your interest towards software development and to inspire you to learn hard. You have a lot to learn yet, but it's interesting, isn't it?

What's Next?

The "Programming Basics" series consists of publications for beginners in programming that covers the following topics:

1. First Steps in Programming – commands, programs, C#, Visual Studio,
2. Simple Calculations – variables, calculations, console input / output
3. Simple Conditions – conditional statements, the "if-else" construction
4. More Complex Conditions – nested if-else, logical "and", "or", "not"
5. Repetitions (Loops) – simple for-loops (repeat from 1 to n)
6. Nested Loops – nested loops and problem solving, drawing 2D figures
7. More Complex Loops – loops with a step, infinite loops with breaks
8. How to Become a Software Engineer?

The next topics are coming. Be patient!

Sign-Up to Study in SoftUni

The easiest way to become a software engineer is to go through the "Software University" training program at SoftUni. Signup now for the free Programming Basics training course:

https://softuni.org/apply

Join the SoftUni community and study programming for free in our interactive training platform. Get live support and mentoring from our trainers. Become a software developer now!