C#

1. Instalation:

Następujące elementy zostały zainstalowane w: „C:\Program Files\dotnet”

• Zestaw .NET Core SDK 3.1.201

• Środowisko uruchomieniowe platformy .NET Core 3.1.3

• Środowisko uruchomieniowe platformy ASP.NET Core 3.1.3

• Środowisko uruchomieniowe platformy .NET Core systemu Windows Desktop 3.1.3

Ten produkt gromadzi dane dotyczące użycia

• Więcej informacji i rezygnacja: https://aka.ms/dotnet-cli-telemetry

Zasoby

• Dokumentacja platformy .NET Core: https://aka.ms/dotnet-docs

• Dokumentacja zestawu SDK: https://aka.ms/dotnet-sdk-docs

• Informacje o wersji: https://aka.ms/netcore3releasenotes

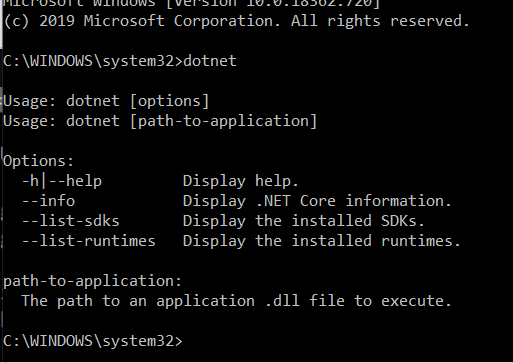
• Samouczki: https://aka.ms/dotnet-tutorials

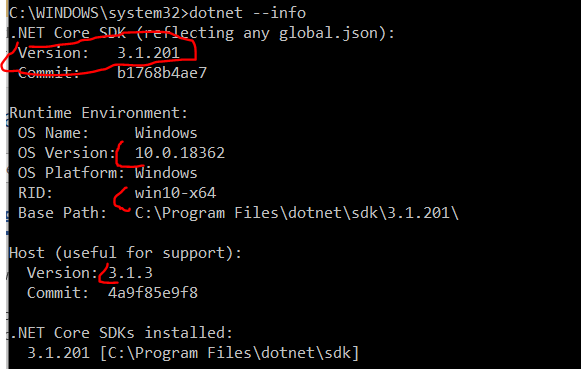
# **TWO .NET FRAMEWORKS:**

.NET:

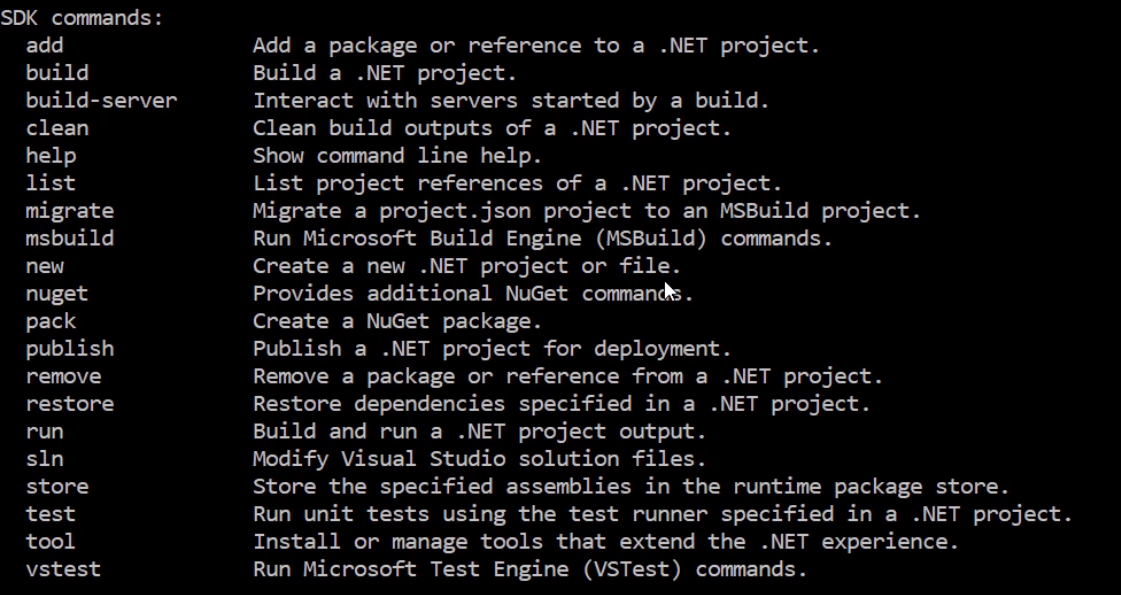
1. .NET Framework (Windows only)
2. .NET Core - completely open-source for Windows, Linux, Mac, ARM
3. **CLR** - Common Language Runtime
4. **FCL** - Framework Class Library

CMD: type dotnet





Version of SDK and of the runtime will be different



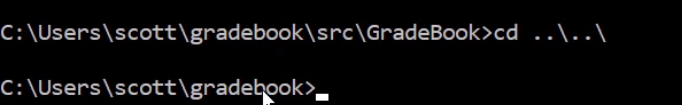
dotnet new console 🡪 console application

cls - clear and create a new application

extenstion .csproj - C# project

dotnet new console - command for new console app

dotnet run - prints Hello World



code . - launch VS Code

# **Steps while creating a project in CMD:**

(user location) mkdir gradebook

cd gradebook

dir

mkdir src

mkdir test

cd src

mkdir GradeBook

cd Gradebook

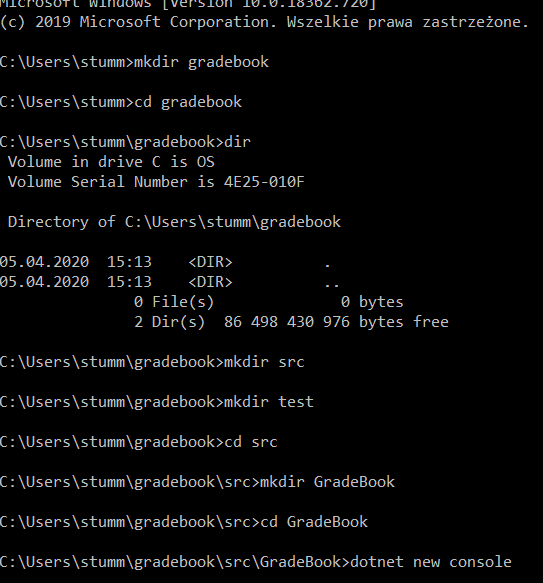
dotnet new console *(info displayed)*

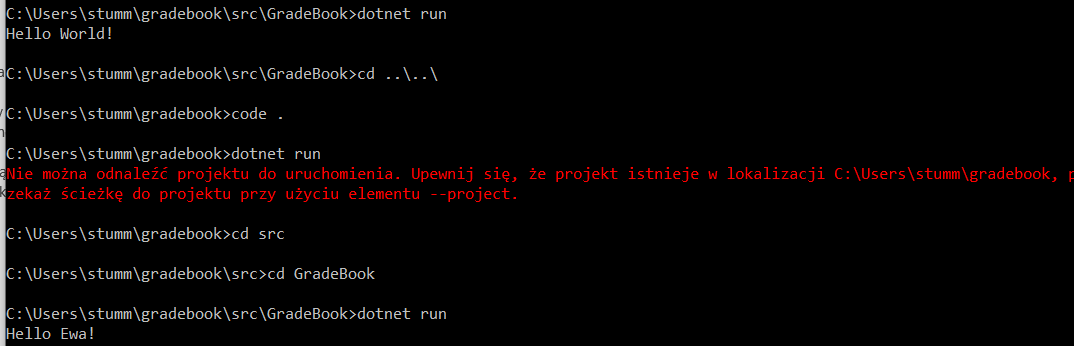
dotnet run

cd ..\..\

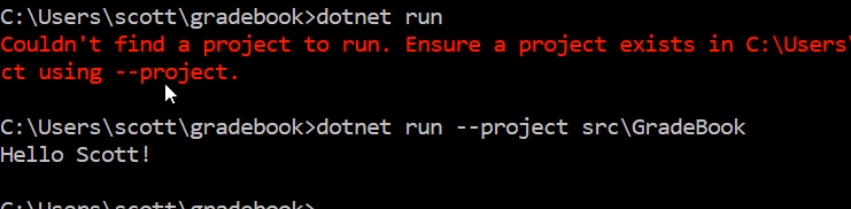
code .

DOTNET RUN IS HERE 🡪 **C:\Users\stumm\gradebook\src\GradeBook>dotnet run**



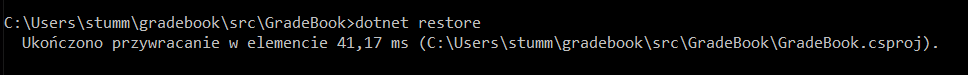


OR:

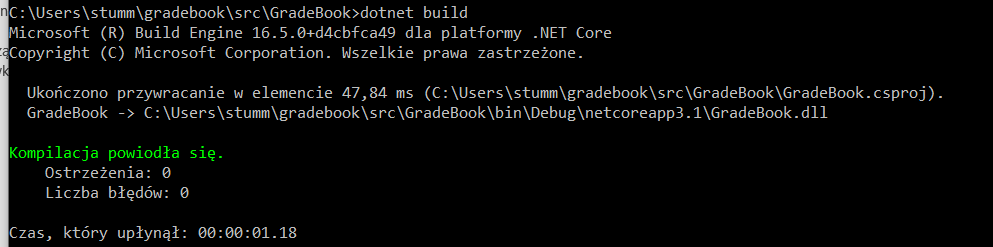


next 🡪 restore

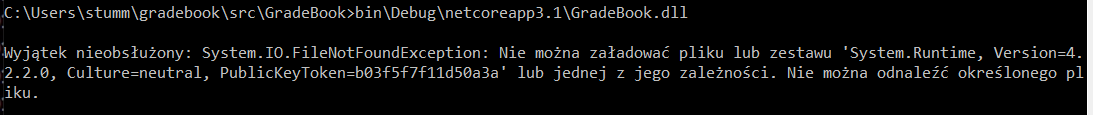
dotnet restore



# **dotnet build 🡪 compilation**

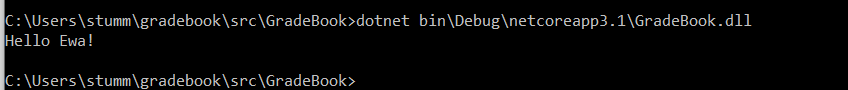


bin 🡪 binary: debug folder



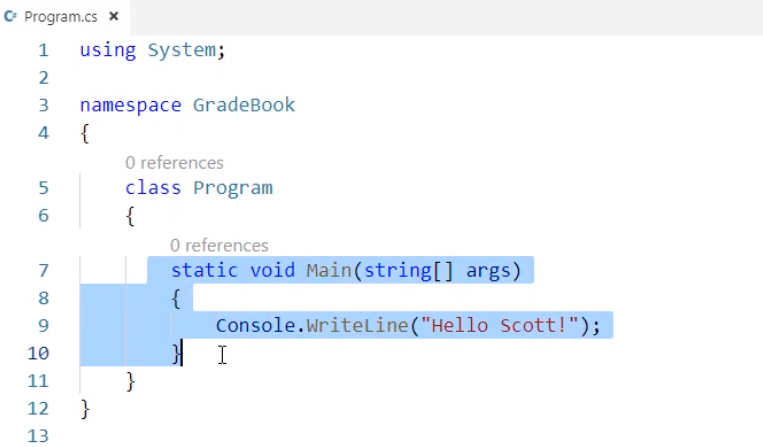
Ways to overcome:

1. dotnet run OR
2. dotnet bin as below:



# **CODE DESCRIPTION:**

Overview:

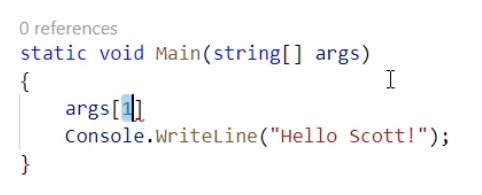


Selected static void (…) Console 🡪 method; method name is Main; executed code put in {}

string[] args 🡪 parameter and parameter name; args are arguments that are passed to the application, whereas type of this parameter is string

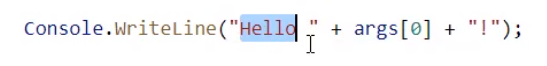
[] - an array

Add args[1] - because C# also has a zero-based indexing:



# **1st solution:**

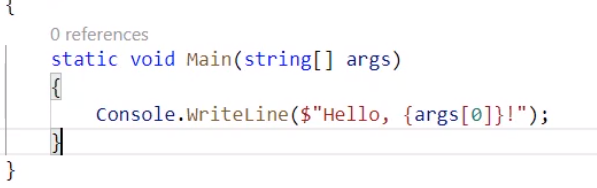
Purpose: to build a string that consists of this writing of ‘Hello…’ and provides the given argument



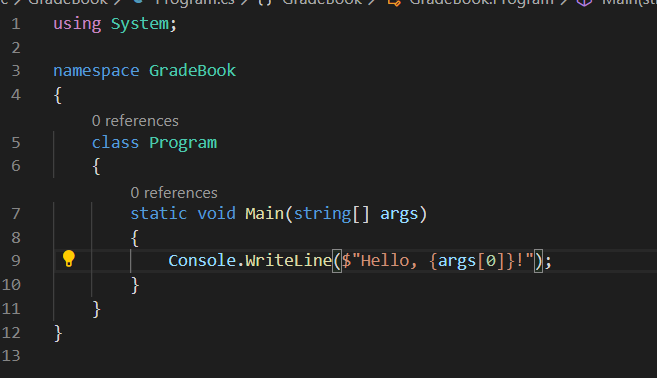
*Note: we are dealing with an array of strings!*

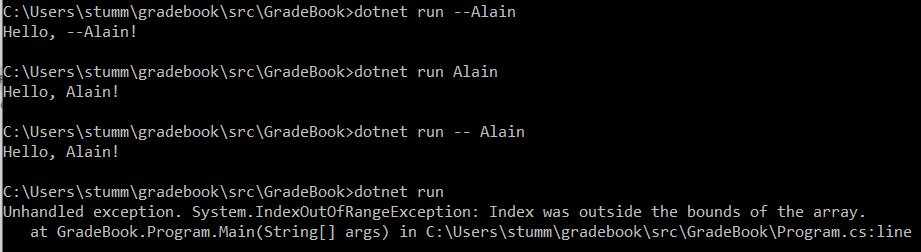
# **2nd solution 🡪 string interpolation:**

2nd solution: String interpolation



Result in my console:

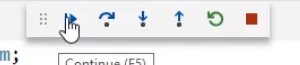




# **Debugging:**

Run 🡪 Start debugging, mark line 9 with red circle beforehand

Debugger tool bar:

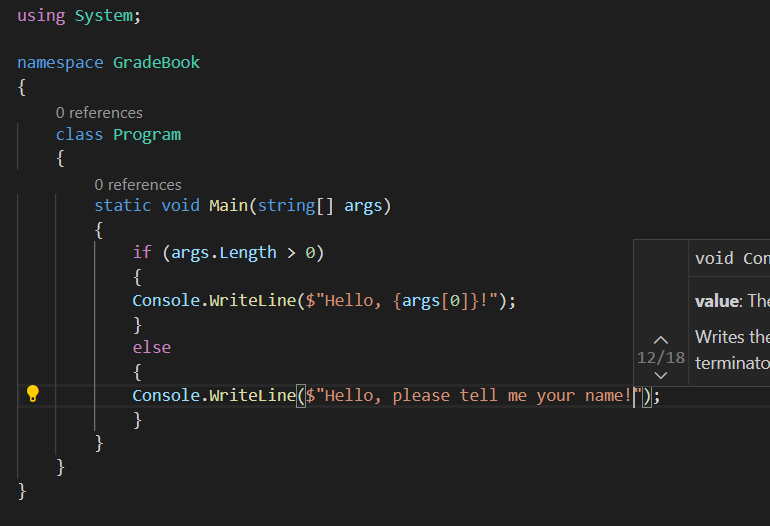


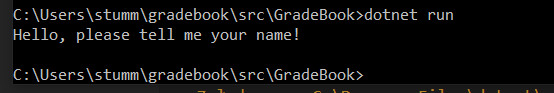
Someone has to pass an argument so that the algorithm is executed

Amend the code and add the condition which returns text even if argument is not provided

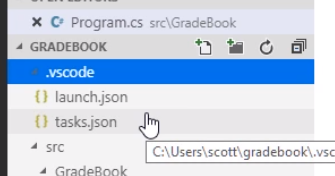


Mine:

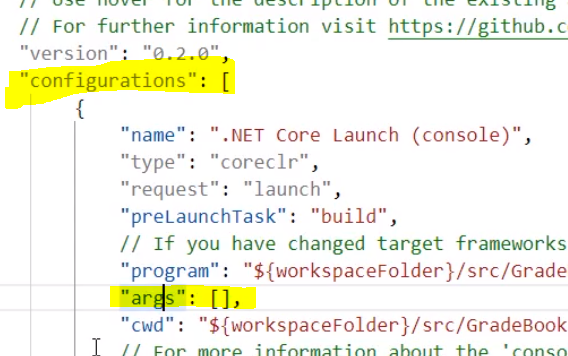




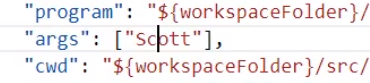
Coming back to files - choose launch.json



Important piece of code for us:



Type arg Scott:



Ctrl+F5 to run the debugger

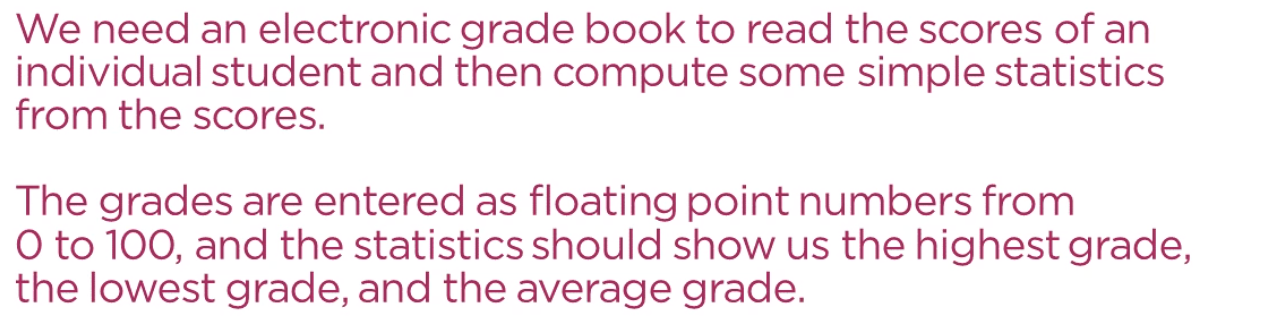
# **Summary**

1. Installation of .NET Core and VS Code
2. Finding first bug in the command line
3. Recognition that we need to provide a parameter in the command line
4. We need to provide condition in case if an argument is not provided

# **MODULE II**

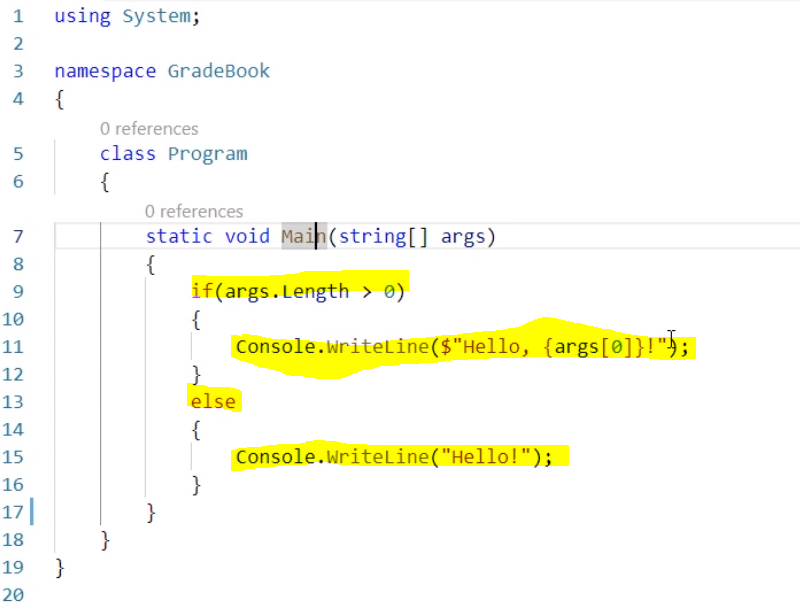
Learning the C# Syntax

Assignment review:



When we execute/build/run an application, we will execute the lines of code which appear inside of this Main method (an entry point of an application).

In C#, we are also likely to use curly braces to denote the start and the end of different methods; start of a method begins with { and ends with } . What is inside, we call statements.



Line 9: an if statement

Line 11: a statement which invokes a WriteLine method

Some statements do not require a semi colon ( ; ) , e.g. if but in places then C# compiles the code into an efficient binary format already require it.

# **Variable declaration**

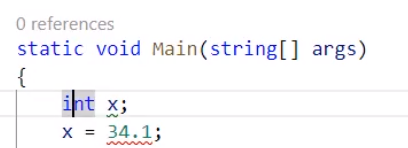
Another statement which we can have is called a variable declaration.

We will need to work with floating point numbers

There are some built-in data types in C# that can be used to hold floating point numbers in a variable. So in place when we declare the variable, we also declare a storage location to hold some value.

Every variable declaration consists of a name and type of a variable.

e.g float, double (double point number which is twice as precise as a float)



or

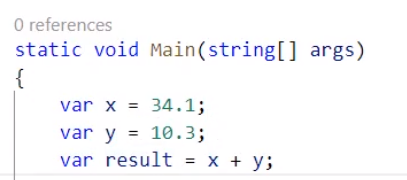


or



We can replace naming a variable type with “var”; however, we cannot store e.g. a string if a numeric value has been defined previously.

Also: we can state var result but the result must be stated



NOTE: in order to return the result, instead of “Console…” we can simply type **cw** and press tab:

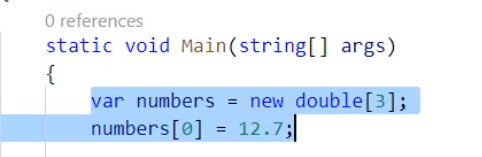
Hereafter:



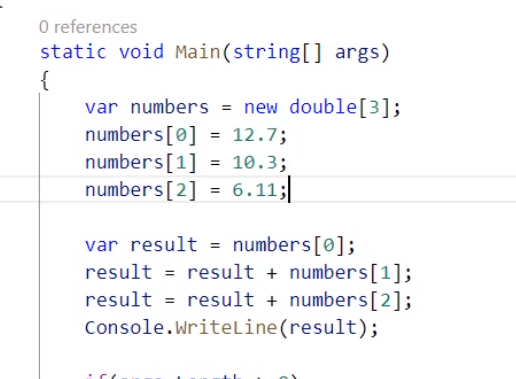
Full code: stored in .txt

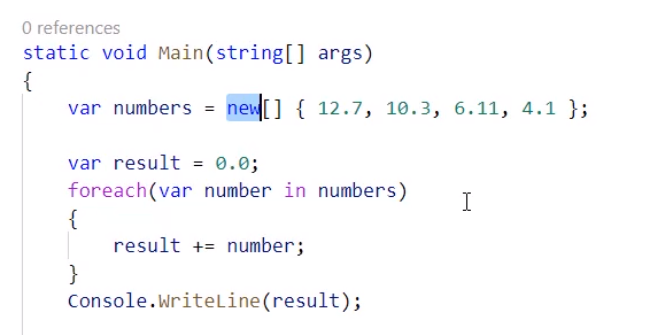
*In our project, we gonna have a collection of collection point numbers that we can manage and which can grow when we add more grades; we don’t want to do it by declaring each grade as we don’t know how many grades there would be so we define a collection as an array*

C# does not allow to use an unassigned local variable as it typically leads to an error in a program

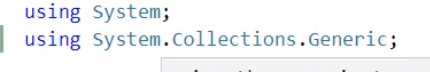


---Next sum example





System.Connections.Generic



Type argument - certain types and classes in .NET libraries that require us to give some additional information how we will use this particular type

Now: a list that stores only double-precision float numbers

