

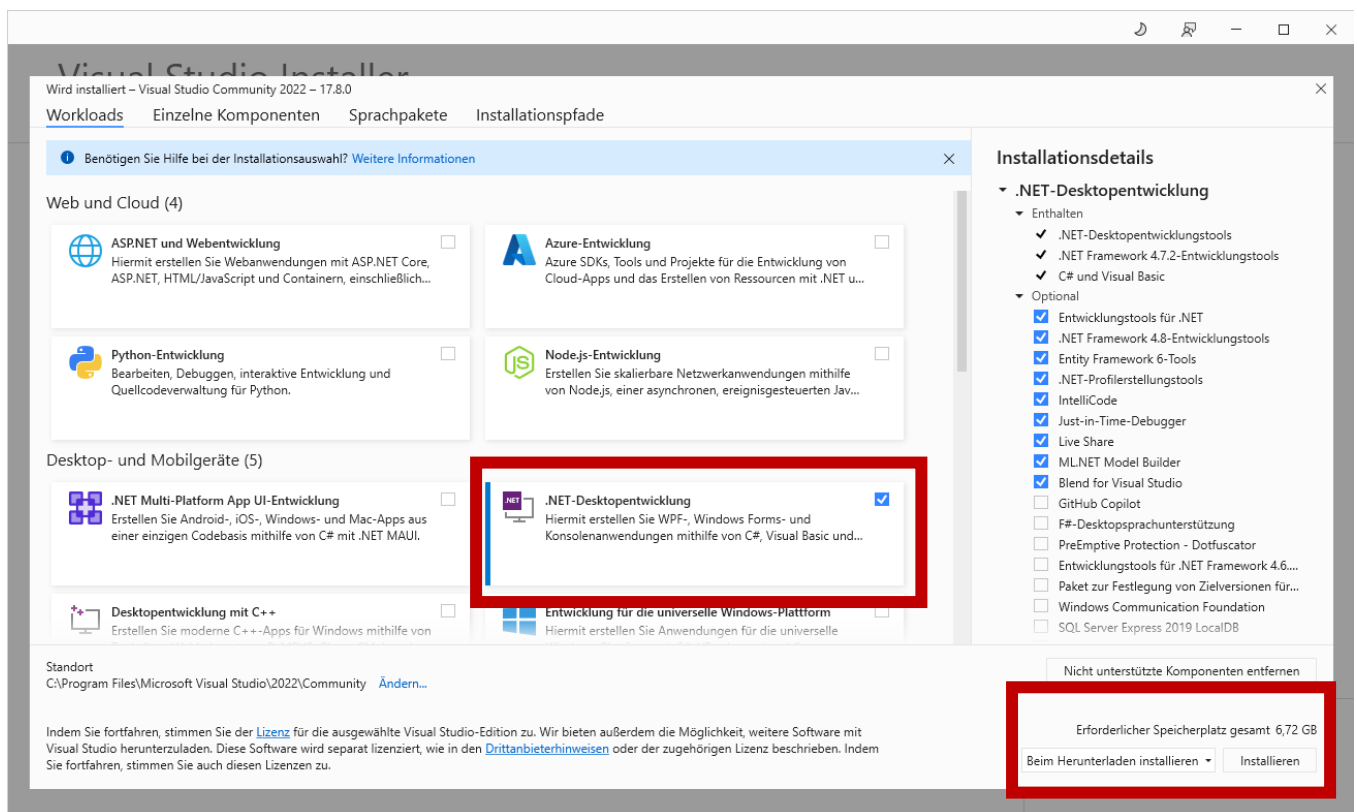
## 1.1.0 Visual Studio Community Edition installieren

### Herunterladen und Ausführen:

Im Teams Ordner finden Sie die Installationsdatei `vs_community_versionsnummer_.exe`. Laden Sie die Datei herunter und führen Sie diese aus.

### Auswählen von Workloads und Installation

Im Installer können Sie die Workloads/Features auswählen, die installiert werden sollen. Für den AWP-Unterricht benötigen Sie das im Screenshot abgebildete Paket. Setzen Sie den Haken bei *.NET-Desktopentwicklung* und starten die Installation.



Alternativ auch als Video: ( <https://www.youtube.com/watch?app=desktop&v=kP71SOQtbGA> )

## 1.1 Situationsbeschreibung „Programmierumgebung“

Sie haben kürzlich Ihre Ausbildung im IT Unternehmen **DataSol** begonnen und sollen sich in eine Ihnen nicht bekannte Programmiersprache einarbeiten.

Testen Sie die Software, in dem Sie wie in den Kapiteln 1.1.1 – 1.1.2 beschrieben ein neues Projekt mit einem Hello-World-Programm anlegen.

### 1.1.1 Ein Projekt in Visual Studio anlegen

Online erklärt am Beispiel von VS2022:

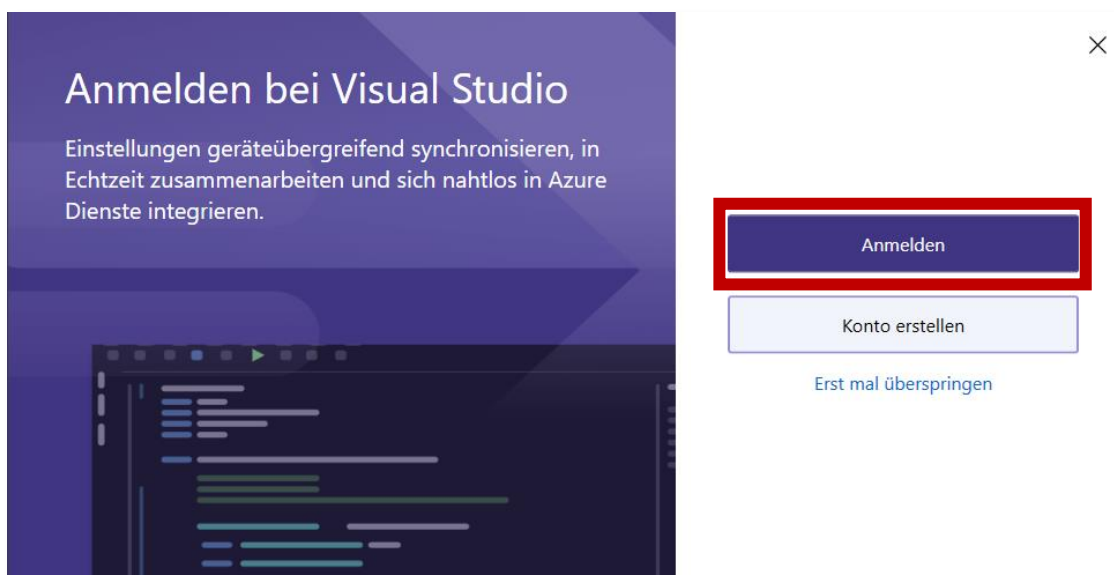
<https://learn.microsoft.com/de-de/dotnet/core/tutorials/with-visual-studio?pivots=dotnet-7-0>

Bitte **lesen** Sie diesen Abschnitt **sehr sorgfältig** durch und führen Sie **die im Text beschriebenen Aktionen** aus. Sämtliche Schritte sind die Grundlage **für alle Programme**, die Sie zukünftig im AWP-Unterricht mit C# **anlegen** und programmieren werden.

Eine umfassende Darstellung der IDE<sup>1</sup> und damit nahezu aller Möglichkeiten die Visual Studio 2022 (VS2022) bietet würde den Rahmen dieser Einführung bei weitem sprengen und ist für das Erste auch nicht erforderlich.

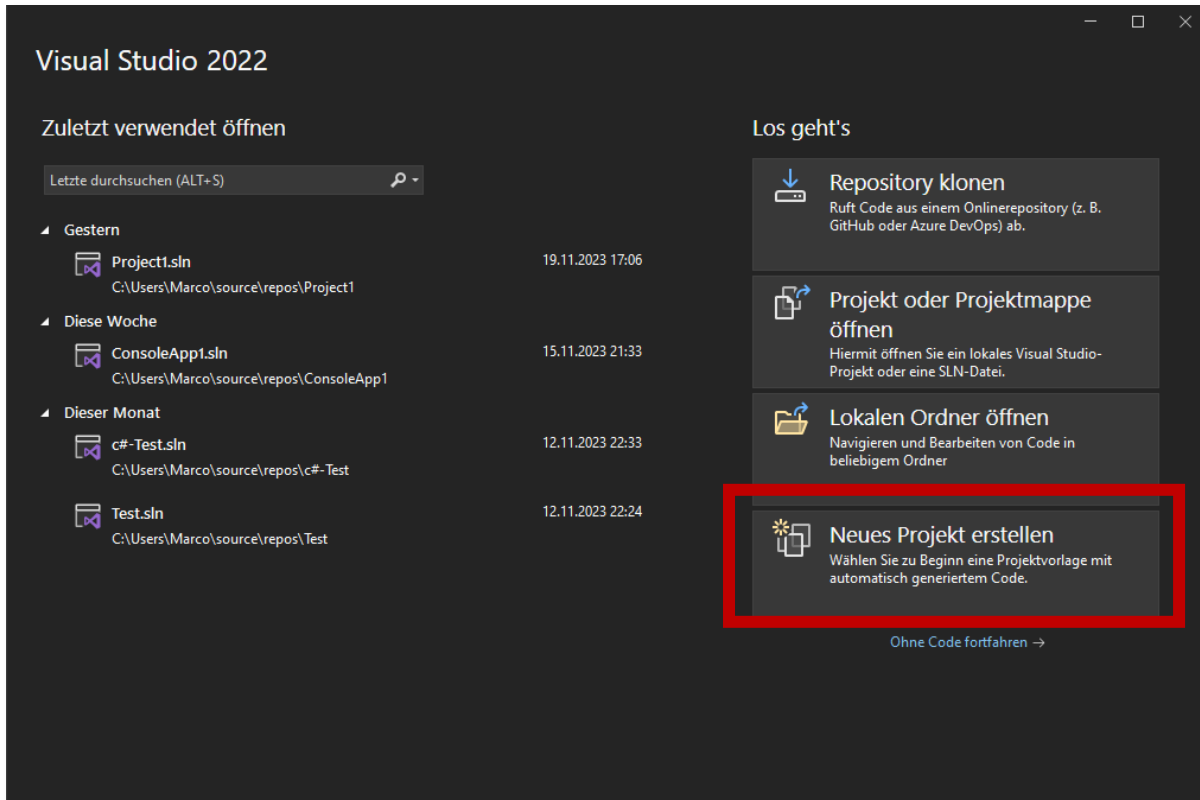
Nach der erfolgreichen Installation von VS2022 ist im angebotenen Auswahlmenü die Option neues Projekt zu wählen.

Starten Sie Visual Studio und melden Sie sich mit ihrem Microsoft-Account (Anmeldedaten wie in MS Teams) an.

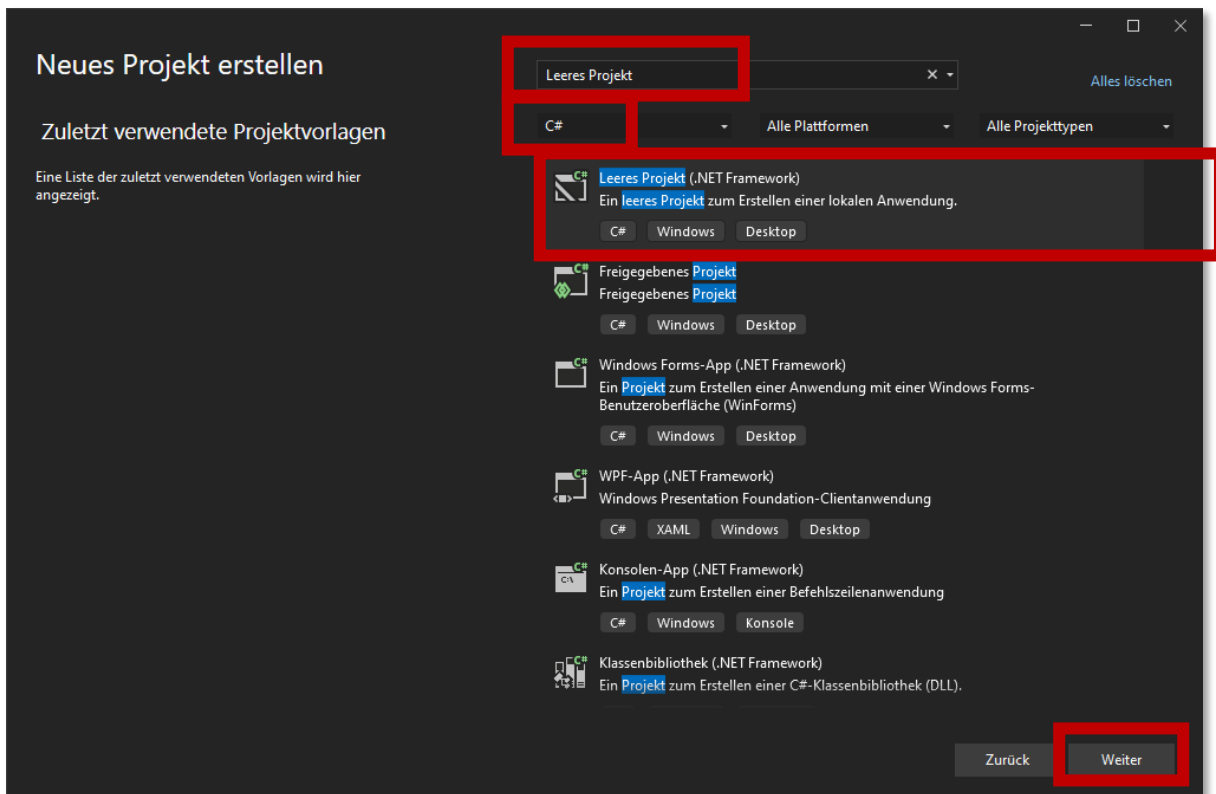


<sup>1</sup> IDE = Integrated Development Environment = Integrierte Entwicklungsumgebung

Erstellen Sie nun ein neues Projekt.



Geben Sie **Leeres Projekt** in die Suchleiste ein und wählen Sie **C#** als Programmiersprache aus. Wählen Sie **Leeres Projekt** und klicken Sie auf weiter.



**Benennen** Sie Ihr Projekt eindeutig (bspw. HelloWorld). Als Speicherort verwenden Sie bitte einen passenden Ordernamen auf dem Desktop des PCs. Bitte legen Sie Ihr Projekt stets in **einem neuen Ordner auf dem Desktop**<sup>2</sup> ab und kopieren/sichern es dann erst am Ende der Stunde auf Ihren USB-Stick oder einen Cloud-Speicher. Alle Daten, die auf dem Schuldesktop abgelegt werden, **werden nach jedem Login automatisch gelöscht**.

Neues Projekt konfigurieren

Leeres Projekt (.NET Framework) C# Windows Desktop

Projektname  
HelloWorld

Ort  
C:\Users\schueler\Desktop\

Name der Projektmappe ⓘ  
HelloWorld

☐ Platzieren Sie die Projektmappe und das Projekt im selben Verzeichnis.

Framework  
.NET Framework 4.7.2

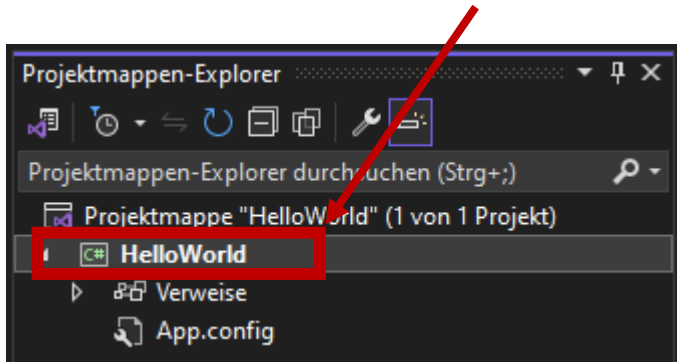
Projekt wird in „C:\Users\schueler\Desktop\HelloWorld\HelloWorld\“ erstellt

Zurück Erstellen

**Sie sind dafür verantwortlich Ihre Projekte, Dateien  
und sonstige Unterlagen sauber abzuspeichern.**

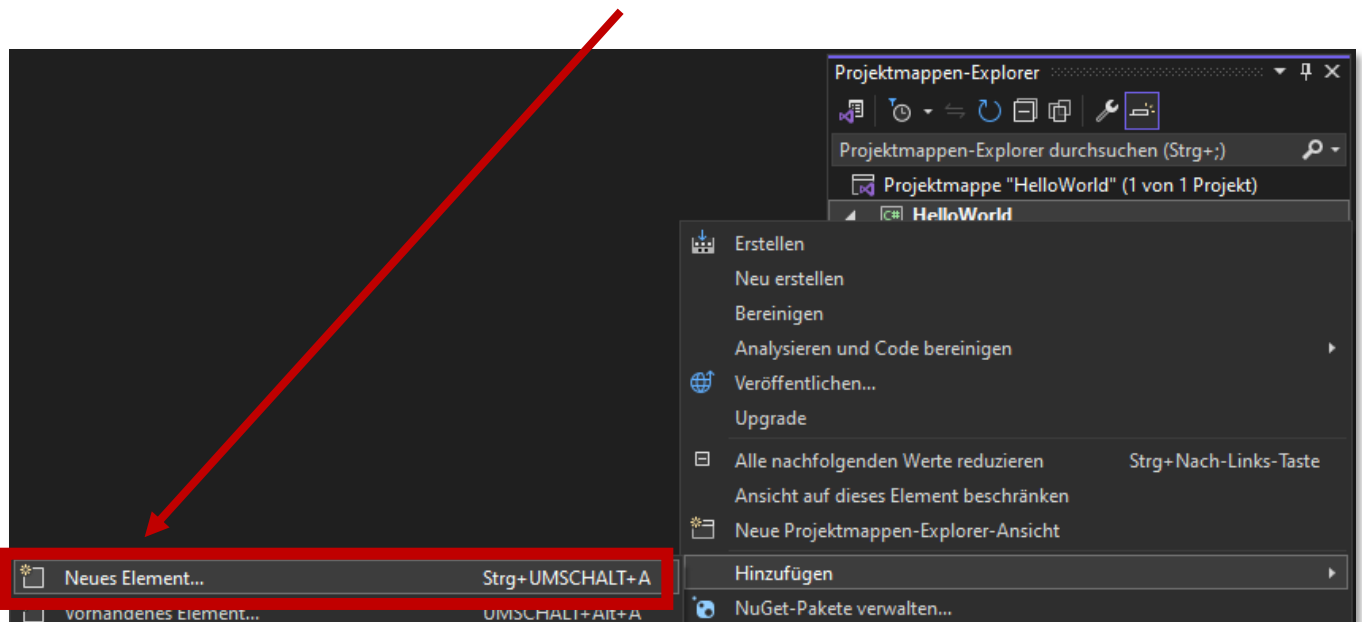
<sup>2</sup> Die PCs verfügen alle über Solid State Disks, welche den Kompiliervorgang im Vergleich zu einem USB-Stick erheblich beschleunigen. Dadurch können Sie viel schneller Ihre Programme testen und nutzen die Lernzeit effizient.

Legen Sie jetzt im Projektmappen-Explorer eine neue cs-Datei für Ihren Programmcode an. Klicken Sie dazu mit der rechten Maustaste auf **Ihren Projektnamen** ...

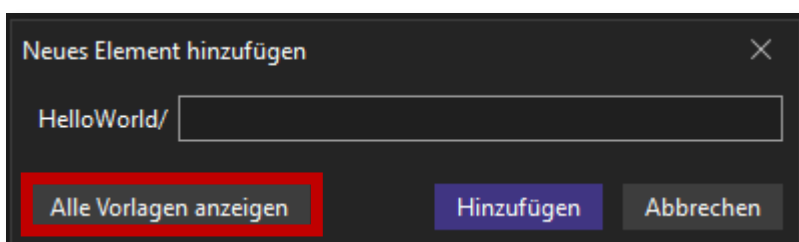


(Falls Sie den Projektmappen-Explorer nicht sehen, können Sie diesen über den Menüpunkt **Ansicht** → **Projektmappen-Explorer** öffnen)

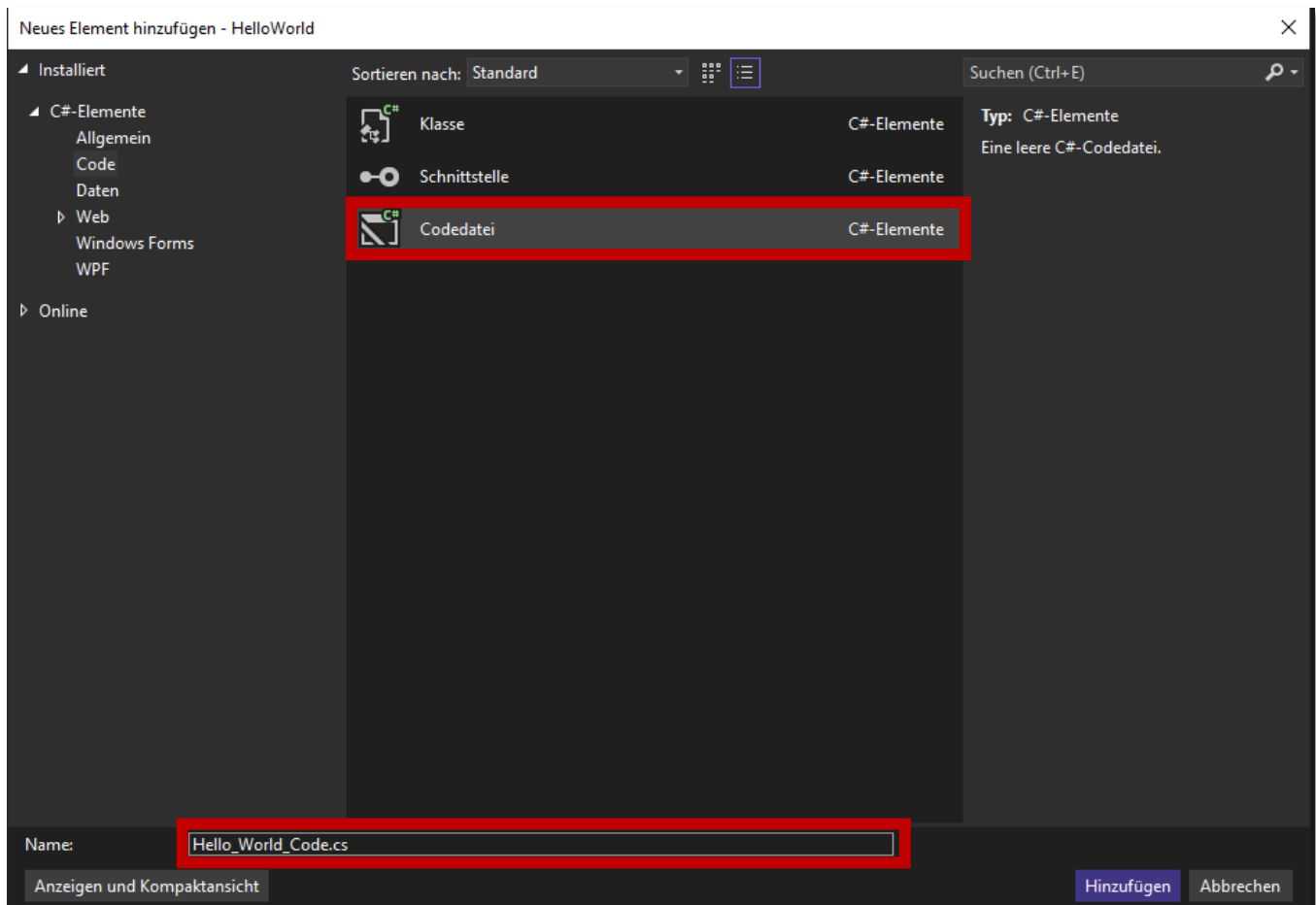
.... und wählen Sie anschließend **Hinzufügen** → **Neues Element** aus



Klicken Sie auf **Alle Vorlagen anzeigen** ...



... und fügen Sie eine **C#-Cododatei** (**.cs**) hinzu. (Die Dateinamen der .cs-Dateien sollten am Schluss immer noch **\_Code** stehen haben, damit ganz klar ist, in welcher Datei des Projekts der Programmcode gespeichert ist. Also bspw. **Hello\_World\_Code.cs**)



Schreiben Sie nun das Programmgerüst und Ihr „Hello-World“-Programm, so wie **auf der folgenden Seite beschrieben**.

Wenn Sie fertig sind, **kompilieren Sie** das Programm durch Drücken der Funktionstaste **F5** oder durch Klick auf **Starten**.



### 1.1.2 Das Programmgerüst in C#

Für das **Hello World**-Programm muss man sich zunächst ein sogenanntes C# -Programmgerüst zurechtlegen. Damit ist die Abfolge von Befehlen gemeint, die vom Prozessor des Computers nacheinander ausgeführt werden, damit „Hello World“ auf dem Bildschirm in der Konsole erscheint.

Der folgende Quellcode zeigt den Ablauf des „Hello World“-Programms. Informieren Sie sich kurz über den Aufbau der einzelnen Programmabschnitte (Codezeilen) und klären Sie deren Funktion in Zusammenarbeit mit Ihrem Partner!

```
1  using System;
2
3  namespace HelloWorld
4  {
5
6      class Hello_World_Code
7      {
8
9          static void Main()
10         {
11
12             Console.WriteLine("Hello World!");
13
14             Console.ReadKey();
15
16         }
17
18     }
19
20 }
21
22 }
```

Einbinden der Bibliothek System, die "Console" ermöglicht.

Festlegen eines Namensbereichs (Namensraum)

Anlegen der Klasse *Hello\_World\_Code* als Behälter für unsere Variablen und Methoden.

Hauptmethode/Einstiegspunkt eines jeden C#-Projekts.

Ein Methodenaufruf, der Hello World! ausgibt.

Warten auf eine Tastatureingabe, sonst würde das Konsolenfenster sofort verschwinden.

Dieses Programmgerüst dient als Vorlage **für alle kommenden Programme**, die Sie in C# im AWP-Unterricht schreiben werden.

### Erklärung zu 1.1.2 Hello World Befehle

`using System;`      **Warum steht da**      `using System; ?`

Das `using System;` Statement in C# ermöglicht den Zugriff auf Typen und Funktionen, die in der System-Namespace enthalten sind. In diesem Fall wird die Console-Klasse für die Ausgabe verwendet, und sie gehört zum System-Namespace. Durch das Hinzufügen des `using System;` Statements signalisierst du dem Compiler, dass er nach den benötigten Klassen im System-Namespace suchen soll, ohne jedes Mal den vollständigen Namespace-Pfad anzugeben.

Ohne dieses Statement müsstest du den vollen Pfad zur Console-Klasse verwenden, was so aussehen könnte: `System.Console.WriteLine("Hello, World!");`

Das `using System;` macht den Code kürzer und lesbarer, indem es dem Compiler mitteilt, dass die Console-Klasse im System-Namespace zu finden ist.

`namespace HelloWorld`      **Warum steht da**      `namespace HelloWorld; ?`

In C# dient der Namespace dazu, den Bereich von Identifiern zu organisieren und zu strukturieren. Der Begriff "HelloWorld" in diesem Zusammenhang könnte ein Namespace sein, der deine Klassen und Typen organisiert.

Der Namespace HelloWorld ist in diesem Fall optional, besonders für kleine Programme. Bei größeren Projekten ist die Verwendung von Namespaces hilfreich, um den Code zu organisieren, zu gruppieren und mögliche Namenskonflikte zu vermeiden, wenn verschiedene Teile des Codes unterschiedliche Funktionen haben.

In diesem Beispiel ist der Namespace also nicht unbedingt erforderlich, aber es ist eine bewährte Praxis, um den Code besser zu strukturieren und zu organisieren, insbesondere wenn du mit umfangreicheren Projekten arbeitest.

`class Hello_World_Code`      **Warum steht da**      `class Hello_World_Code ?`

In C# wird der Code in Klassen organisiert, und jede ausführbare Anwendung benötigt eine Klasse mit einer Methode namens Main. Die Main-Methode ist der Startpunkt des Programms, von dem aus die Ausführung beginnt.

Die Zeile `class Hello_World_Code` definiert eine Klasse mit dem Namen Program. Innerhalb dieser Klasse befindet sich die Main-Methode, die den Einstiegspunkt für die Ausführung des Programms darstellt. Der Begriff "Hello\_World\_Code" ist in diesem Kontext ein Bezeichner und könnte auch anders benannt werden, solange er den Regeln für Bezeichner in C# folgt.



In C# wird die Klasse, die die Main-Methode enthält, auch oft einfach nur `Program` genannt. Dies erleichtert das Verständnis und die Lesbarkeit des Codes für andere Entwickler.

### `static void Main()` Warum steht da `static void Main()` ?

In C# ist die Main-Methode der Einstiegspunkt für die Ausführung eines Programms. Hier ist eine kurze Erklärung der Teile von `static void Main()`:

**static:** Das Schlüsselwort "static" gibt an, dass die Methode auf Klassenebene ist und nicht auf Instanzebene. Die Main-Methode muss statisch sein, damit sie ohne eine Instanz der Klasse aufgerufen werden kann.

**void:** Das Schlüsselwort "void" zeigt an, dass die Main-Methode keinen Rückgabewert hat. Sie führt Anweisungen aus, gibt jedoch keinen Wert zurück.

**Main():** Dies ist der Name der Methode. In C# wird die Main-Methode als spezielle Methode betrachtet, da sie als Startpunkt für die Ausführung des Programms dient. Der Name "Main" ist konventionell, aber er könnte auch anders benannt werden, solange die Signatur (Rückgabotyp, Name, Parameter) korrekt ist.

Die vollständige Signatur `static void Main()` gibt an, dass das Programm beim Start die Main-Methode aufruft, um die Ausführung zu beginnen.

```
    Console.WriteLine("Hello World!");
```

### Warum steht da `Console.WriteLine("Hello World!");` ?

`Console.WriteLine` ist eine Methode in der `System`-Namespace der .NET-Bibliothek und wird in C# verwendet, um einen Text oder eine Zeichenkette auf der Konsole auszugeben, gefolgt von einem Zeilenumbruch. In diesem Beispiel wird der Text "Hello, World!" auf der Konsole ausgegeben, und anschließend wird automatisch in eine neue Zeile gewechselt. `Console.WriteLine` ist besonders nützlich, wenn du Informationen auf der Konsole anzeigen möchtest, sei es für Benutzereingaben, Debugging-Zwecke oder allgemeine Ausgaben während der Programmausführung.

```
    Console.ReadKey();
```

### Warum steht da `Console.ReadKey()` ?

Ohne `Console.ReadKey()` würde die Konsole nach dem Ausführen des Programms sofort geschlossen werden. Das liegt daran, dass die Ausführung des Programms zum Ende der Main-Methode gelangt und die Konsole automatisch geschlossen wird.

Durch das Einfügen von `Console.ReadKey()` wird das Programm angehalten und wartet darauf, dass der Benutzer eine Taste auf der Tastatur drückt, bevor es beendet wird. Dies bietet eine Möglichkeit, die Ausgabe zu betrachten, bevor das Konsolenfenster geschlossen wird. Es ist besonders hilfreich, wenn du das Programm außerhalb einer integrierten Entwicklungsumgebung (IDE) ausführst, wo das Konsolenfenster nach Abschluss des Programms standardmäßig geschlossen wird.

### 1.1.3 Das Programmgerüst in C#

#### Bonusaufgaben für Schnelldenker:

- 1) Was passiert in Ihrem Programm, wenn Sie den Befehl `using System;` weglassen?

kann nicht direkt auf Funktionen der Console-Klasse zugreifen

Dies bedeutet, dass Sie Funktionen wie `Console.WriteLine` nicht verwenden können, um Text auf der Konsole auszugeben.

- 2) Was ist der Unterschied zwischen:

```
Console.Write("Hello ");
```

Wird auf eine Zeile geschrieben, also:

```
Console.Write("World");
```

Hello World

und

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

Wird auf der Zeile der Console geschrieben, also:

```
Console.WriteLine("World");
```

Hello

World

- 3) Was ist der Unterschied zwischen einer Methode und Funktion? Tipp: Nutzen Sie Ihren Internetzugang.

### 1.1.4 Grundsätzliche Regeln beim Programmieren mit C# am Beispiel von „Hello World“

Befehle, welche anschließend einen **Anweisungsblock** {...} öffnen, wie `static void Main()`, benötigen kein Semikolon, da der Anweisungsblock zum Befehl gehört und dieser erst danach beendet ist.

Jede öffnende Klammer muss auch wieder

geschlossen werden.  
Nach einer schließenden Klammer ist eine Bedingung oder ein Anweisungsblock beendet.

Ein Semikolon ist hier nicht nötig.

Zur besseren Lesbarkeit empfiehlt es sich, öffnende geschweifte Klammern immer in einer neuen Zeile zu schreiben, da so die öffnende Klammer immer über der dazugehörigen schließenden Klammer steht. nicht so:

```
void Main(){  
...  
}
```

sondern so:

```
void Main()  
{  
...  
}
```

```
using System;  
  
namespace HelloWorld  
{  
    O references  
    class Hello_World_Code  
    {  
        O references  
        static void Main()  
        {  
            Console.WriteLine("Hello World!");  
            Console.ReadKey();  
        }  
    }  
}
```

In C# wird jeder Befehl mit einem Semikolon abgeschlossen. Der Compiler weiß damit, dass ein Befehl zu Ende ist.

Formatierungszeichen wie **Leerzeichen, Tabulatoren und Zeilenumbrüche** werden vom Compiler ignoriert,

wenn sie nicht mitten in einem Befehl, einer Zeichenkette oder ähnlichen Strukturen stehen.

```
void Main()  
{  
    Console.WriteLine("Hello");  
    Console.ReadKey();  
}
```

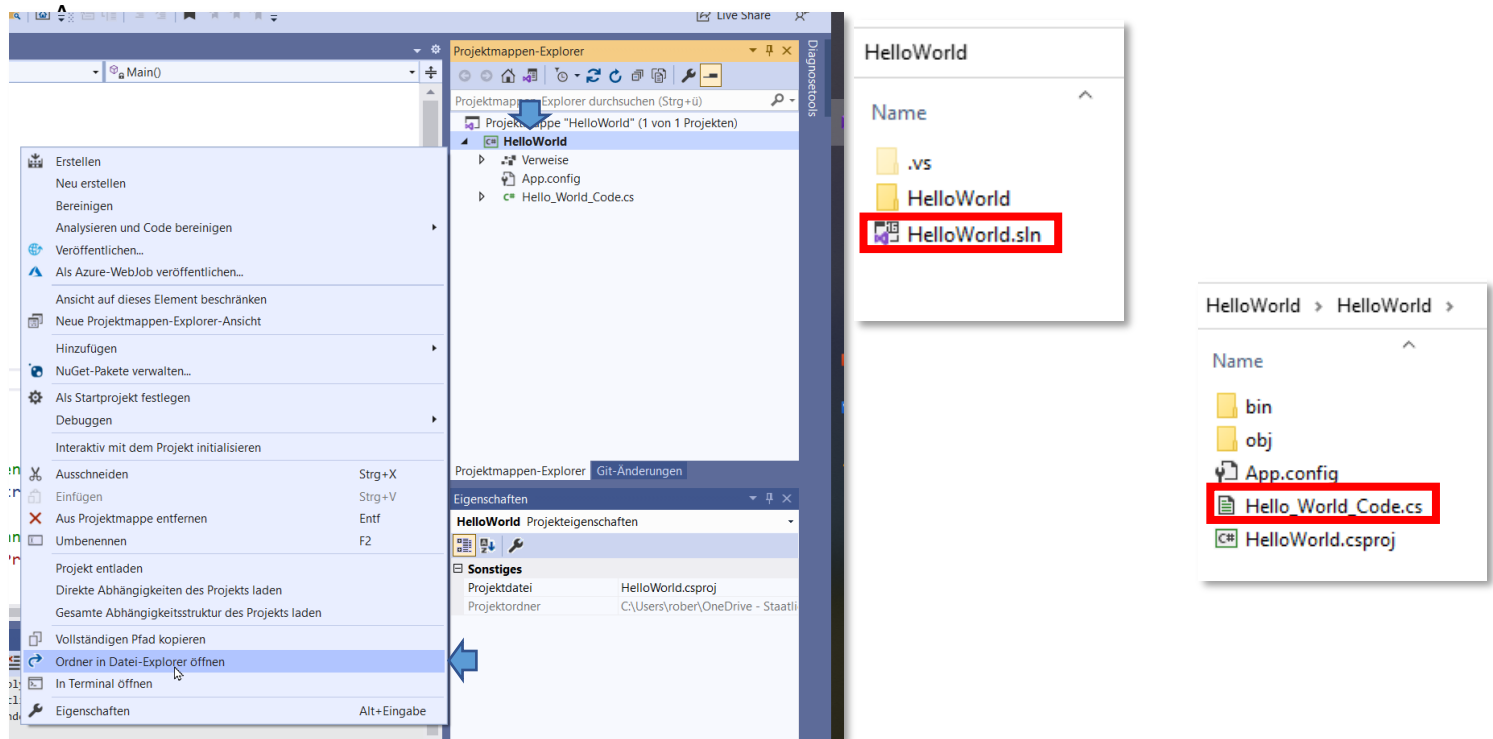
bewirkt also das gleiche wie

```
void Main()  
{  
    Console.WriteLine("Hello");  
  
    Console.ReadKey();  
}
```

### 1.1.5 Projektdateien

Um nachzuvollziehen, wie aus dem geschriebenen Code eine ausführbare Datei wird, öffnen Sie bitte Ihren Projektordner zum „Hello World“-Programm und informieren Sie sich über die einzelnen Dateien und deren Endungen – dafür dürfen Dateierweiterungen nicht ausgeblendet sein.

Dateien mit den Endungen \*.sln, \*.vcxproj und \*.vcxproj.filters dienen beispielsweise der Projektorganisation unter Visual Studio. **Öffnen Sie Ihre Projektmappe stets über die Solution-Datei (\*.sln).**



1) Über welche Datei sollten Sie Ihr Programm in Visual Studio öffnen?

HelloWorld.sln: öffnet die Gesamte Lösung inklusive Programmcode und dazugehörigen Projekten.

2) In welcher Datei ist der Quelltext/Code gespeichert?

Hello\_World\_Code.cs

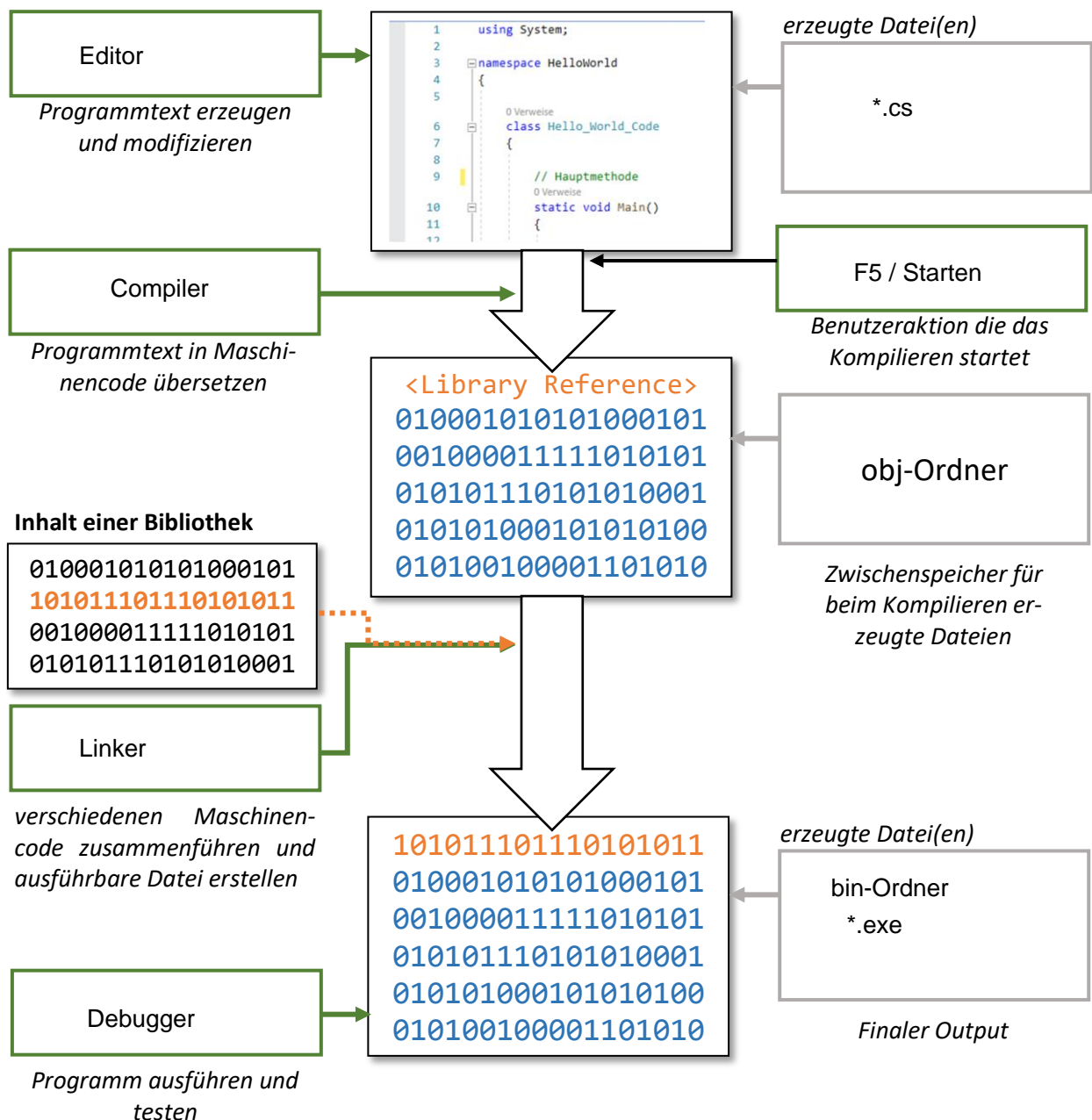
3) In welchem Verzeichnis innerhalb ihres Projektordners wird das fertig kompilierte Programm abgelegt?

HelloWorld > HelloWorld > bin > Debug/ Release

### 1.1.6 Komponenten der IDE (integrierte Entwicklungsumgebung)

Die Übersetzung Ihres „Hello World“-Quellcodes und die anschließende Ausführung des Programms in bspw. einem Konsolenfenster besteht dabei nicht nur aus einer einzigen Handlung, sondern ist eine schrittweise Durchführung verschiedener Arbeitsschritte, die in der Regel die IDE für Sie übernimmt.

Ordnen Sie die folgenden Begriffe zu: F5 / Starten, Editor, Compiler, Linker, Debugger, bin-Ordner, \*.exe, \*.cs



### 1.1.7 Kommentare in C#

In allen Programmiersprachen gibt es die Möglichkeit im Quelltext Notizen, bzw. Kommentare, zu verfassen. Diese sind Passagen innerhalb des Programms die vom Compiler ignoriert werden. Hauptsächlich kommentiert man seinen Code für andere – aber eben auch für sich selbst. Denn nach ein paar Wochen werden Sie möglicherweise Ihren eigenen Quelltext nicht mehr ohne weiteres verstehen. Kommentare sind also Hinweise, die Ihnen und anderen helfen besser und vor allem **schneller** zu verstehen, was das Programm tut.

*Ein Programm ist meistens nur so gut, wie seine Kommentare.*

Wie zu kommentieren ist, sehen Sie anhand der folgenden Beispiele. Dieses Vorgehen gehört zu den **Programmierrichtlinien** im AEuP-Unterricht hier an der Berufsschule - dazu zählen auch Benennung von Projekt- und Code-Dateien, Quelltext-Einrückung usw.

In Ihrem Unternehmen gibt es u. U. ebenso solche Programmierrichtlinien, die von denen hier gezeigten abweichen können - erkundigen Sie sich doch einfach einmal danach.

In C# stehen zwei verschiedene Arten an Kommentarsymbolen zur Verfügung:

- **Mehrzeilige Kommentare**

Der Mehrzeilen-Kommentar wird eingeleitet durch die beiden Zeichen `/*` und durch die Zeichen `*/` beendet. Zwischen diesen beiden Zeichen darf kein Leerzeichen stehen. Und alles was zwischen diesen Zeichenfolgen steht wird vom Compiler ignoriert.

Mithilfe dieses Kommentars ist es also möglich, mehrere Anweisungen z. B. für Testzwecke außer Kraft zu setzen. Dazu wird einfach vor den entsprechenden Anweisungen der Kommentar geöffnet und am Ende der Anweisungen der Kommentar wieder geschlossen

```
/* Hauptmethode */
0 references
static void Main()
{
    /*
    Console.WriteLine("Hello World");

    Console.ReadKey();
    */
}
```

- **Einzeilige Kommentare**

Die zweite Möglichkeit wird mit den Zeichen `//` eingeleitet. Alles was danach bis zum Zeilenende folgt, wird vom Compiler als Kommentar betrachtet. Auch hier darf zwischen den Zeichen `//` kein Leerzeichen stehen

Diese Variante ist die moderne Art des Kommentars. Sie ist in der Regel vorzuziehen, da sie einige Vorteile gegenüber der alten, noch aus C stammenden mehrzeiligen Variante hat.

```
// Hauptmethode
0 references
static void Main()
{
    // Console.WriteLine("Hello World");

    // Console.ReadKey();
}
```

**Vorteile von Kommentaren:**

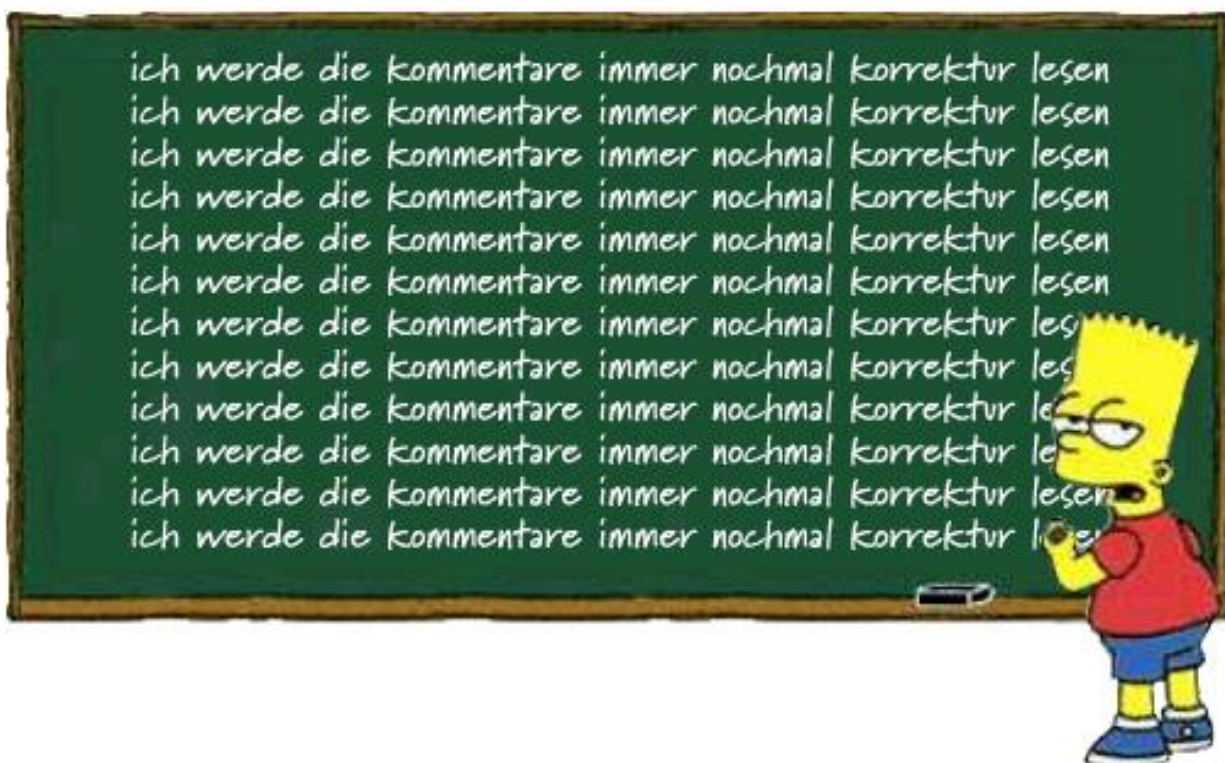
- Andere Programmierer und man selbst findet sich schneller im Programm zurecht. (Kommentierzeit 20sec vs. (Wieder-)Einarbeitungszeit 5min)
- Ein guter Kommentar erklärt Schritt für Schritt, was im Programm passiert.
- Fehler im Code können dadurch evtl. leichter gefunden werden.
- Reduziert Rückfragen zu Ihrem Code von Kollegen oder Kunden

**CleanCode vs. Kommentare**

Grundsätzlich sollte Code selbsterklärend sein. Hierzu ist es wichtig stets sprechende und wohlformulierte Variablen und Funktionsnamen zu wählen, welche die Inhalte bzw. die Übergabe und Rückgabewerte sehr gut beschreiben. Wenn der Code sich selbst erklärt, dann ist ein erklärender Kommentar eigentlich überflüssig (#CleanCode).

Aber es gibt ja nicht nur Kommentare, die den Code erklären (rechtliche Hinweise (//Copyright (c) 2021 DataSol GmbH. All rights reserved.), Warnungen (//Running this part of code will take extremely long! usw.) Insbesondere für Anfänger im Bereich der Programmierung ist es häufig sehr hilfreich, erklärende Kommentare aufzuschreiben.

In Wahrheit sind Kommentare auch nicht zwingend ein Zeichen für schlechten Code. An der richtigen Stelle, im richtigen Umfang und mit dem richtigen Ziel können Kommentare den Code verständlicher und lesbarer machen! Empfehlung für die 10te Klasse → **Lieber zu viel als zu wenig kommentieren!**





### 1.1.8 Ausgaben in C#

Damit die Anweisung `Console` verwendet werden kann, muss die entsprechende Bibliothek `System` eingebunden werden.

```
using System;
```

Durch Laden der System Bibliothek werden folgende Methoden bereitgestellt:

<https://docs.microsoft.com/de-de/dotnet/api/system?view=net-5.0>

#### Ausgabe von Text und Zeichen

Sollen Texte ausgegeben werden, so muss folgendermaßen vorgegangen werden:

- Nach `Console` folgt der Methodenaufruf von `Write`.
- Im Anschluss folgt dann der Text, der in Anführungszeichen/Gänsefüßchen `"..."` einzuschließen ist.
- Die Befehlszeile wird von einem `;` abgeschlossen.

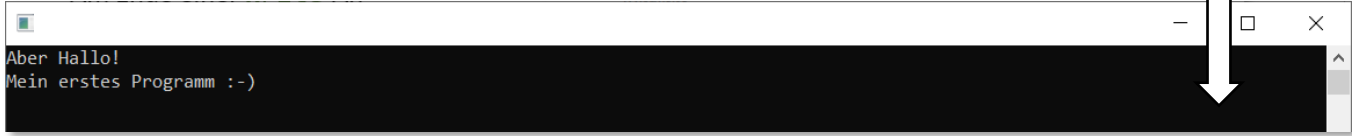
Mehrere Wörter können mit dem `+` Operator kombiniert werden.

Am Ende einer `Write`-Anweisung wird kein automatischer Zeilenumbruch eingefügt – dafür können Sie u. A. `System.Environment.NewLine` in den Ausgabe-stream einfügen.

```
using System;

namespace HelloWorld
{
    0 Verweise
    class Hello_World_Code
    {
        0 Verweise
        static void Main()
        {
            Console.WriteLine("Aber Hallo!" + System.Environment.NewLine);
            Console.WriteLine("Mein erstes Programm :-");
            Console.ReadKey();
        }
    }
}
```

*entsprechende Konsolenausgabe*



```
Aber Hallo!
Mein erstes Programm :-)
```

Einzelne Zeichen heißen auch „**Character**“ – kurz Char. Sie unterscheiden sich zu Texten dahingehend, dass ein Char eben nur ein einzelnes Zeichen und ein Text die Abfolge von mehreren Zeichen (Zeichenkette) darstellt. Eine solche Zeichenkette wird String genannt.

Chars müssen lediglich in Hochkommata (z. B. `'!'`) eingeschlossen werden. Werden Sie fälschlicherweise in Anführungszeichen (z. B. `"!"`) eingeschlossen, so werden sie als Strings (Zeichenkette) behandelt. Das beeinflusst u. A. den benötigten Speicherbedarf.



### 1.1.9 Übung zu Ausgaben in C#

Überprüfen Sie die nachfolgenden Verwendungen von `Write`. Welche Angaben stimmen und wie sieht die entsprechende Konsolenausgabe dazu aus? (Tipp: Copy&Paste)



```
int a = 5;
int b = 6;
```

Verwenden Sie eine Kopie Ihres Hello World-Programmes (Projektname: Ausgaben) und fügen Sie die beiden Definitionen der Variablen a und b (siehe links) in die ersten Zeilen Ihrer Main-Funktion ein.

Verwendungen	Richtigkeit und Ausgabe
<code>Console.Write("Hallo Welt!" + "Du bist schön!");</code>	Hallo Welt!Du bist schön! (mehrere Ausgabe-Objekt)
<code>Console.Write('Hallo Welt');</code>	Falsch (String kann nicht als Char ausgegeben werden)
<code>Console.Write("Das Ergebnis ist: " + a);</code>	Das Ergebnis ist: 5 (mehrere Ausgabe-Objekt unterschiedlichen Typs)
<code>Console.Write("Das Ergebnis ist: " + (a + b));</code>	Das Ergebnis ist: 11 (mehrere verschiedene Ausgabe-Objekte sowie Berechnungen in Ausgabe)
<code>Console.Write("Das Ergebnis ist: " + ('a' + 'b'));</code>	Das Ergebnis ist: 195 (sinnlos, da Addition der ASCII-Werte von a=97 und b=98)
<code>Console.Write("Das Ergebnis von: " + a + '+' + b + " lautet: " + (a + b));</code>	Das Ergebnis von: 5+6 lautet: 11 (Kombination aus oberen Varianten)
<code>Console.Write(\$"Das Ergebnis von: {a} + {b} lautet: " + (a + b));</code> String interpolation	Das Ergebnis von: 5 + 6 lautet: 11 (Kombination aus oberen Varianten)
<code>Console.Write("Test" + System.Environment.NewLine);</code>	Test (Zeilenumbruch NewLine)
<code>Console.Write("Test \n");</code> <code>Console.Write("Test" + "\n");</code>	Test Test (Zeilenumbruch durch Escape-Sequenz \n NewLine als String-Ausgabe) Escape-Sequenz werden durch einen Backslash("\") eingeleitet
<code>Console.Write("Test" + "\\n");</code>	Test\n

**Merke:**

- Variableninhalte werden ohne Hochkommata oder Anführungszeichen ausgegeben.
- Einzelne Zeichen (Chars) werden in Hochkommata gesetzt, Zeichenketten (Strings) in Gänsefüßchen.
- Hinter einzelnen Chars (genauer: Zeichenkonstanten) stehen Zahlenwerte (siehe ASCII-Tabelle).
- In einer Zeichenkette können Variableninhalte, Zeichen und Strings mit „+“ aneinandergereiht werden.



### Exkurs: Zum Unterscheid zwischen VS-Community Edition und VS-Code:

Visual Studio Code (VS Code) und Visual Studio Community Edition unterscheiden sich in der Art und Weise, wie sie Code darstellen und bearbeiten:



#### 1. Darstellung des Codes:

- **Visual Studio Community Edition:** Bietet eine sehr detaillierte und umfangreiche Darstellung des Codes. Es verfügt über fortgeschrittene Features wie IntelliSense (eine Art intelligente Autovervollständigung), die die Entwicklung mit C# und .NET besonders unterstützt. Die IDE **zeigt auch detaillierte Informationen über Klassen, Methoden, Eigenschaften und mehr an**, was besonders für umfangreiche und komplexe Projekte hilfreich ist.
- **Visual Studio Code:** Auch VS Code bietet eine gute Darstellung des Codes mit Funktionen wie Syntaxhervorhebung, einfacher Autovervollständigung und grundlegenden IntelliSense-Fähigkeiten. Die Darstellung ist jedoch **insgesamt weniger detailliert** als in Visual Studio Community. VS Code ist flexibler und leichter und legt mehr Wert auf eine schlanke, effiziente Benutzeroberfläche.

#### 2. Umfang der Codebearbeitungsfunktionen:

- **Visual Studio Community Edition:** Ist darauf ausgelegt, eine umfassende Entwicklungserfahrung für C# und .NET zu bieten. Es umfasst **leistungsstarke Debugging-Tools**, **grafische Benutzeroberflächendesigner** (für WPF, Windows Forms etc.), umfassende Refactoring-Tools und integrierte Datenbankverwaltungswerkzeuge.
- **Visual Studio Code:** Bietet zwar grundlegende Debugging-Tools und kann durch Erweiterungen erweitert werden, hat aber nicht die gleiche Tiefe an spezialisierten Tools für C# wie Visual Studio Community. Es ist mehr auf die Bearbeitung von Quellcode und weniger auf Design oder umfangreiches Projektmanagement ausgelegt.

#### Für Anfänger:

- **Visual Studio Community** könnte anfangs überwältigend sein, bietet aber langfristig mehr Unterstützung und Funktionen für komplexe C#-Projekte. Die Strukturen die bei der Programmierung wichtig sind, werden besser dargestellt.
- **Visual Studio Code** ist einfacher zu erlernen und ausreichend für die meisten grundlegenden und mittleren Entwicklungsanforderungen, besonders wenn man neben C# auch andere Sprachen erkunden möchte.

Zusammenfassend, während **Visual Studio Code** eine effiziente und benutzerfreundliche Umgebung für Codebearbeitung bietet, ist **Visual Studio Community Edition** umfangreicher und besser für komplexe C#-Projekte geeignet, besonders wenn man die volle Palette an Entwicklungswerkzeugen und -funktionen nutzen möchte. Letztendlich hängt die Wahl zwischen den beiden von den spezifischen Bedürfnissen, dem Projektfokus und persönlichen Vorlieben ab. Wir empfehlen zum besseren Verständnis **VS-Community Edition**.