

1.0.0

H1: Werken met Visual Studio

Introductie tot C#

Introductie tot programmeren met C-Sharp

Het algoritme

De essentie van een computerprogramma is het zogenaamde **algoritme** (het "recept" zeg maar). Dit is de reeks instructies die het programma moet uitvoeren telkens het wordt opgestart. Het algoritme van een programma moet je zelf verzinnen. De volgorde waarin de instructies worden uitgevoerd zijn uiteraard zeer belangrijk. Dit is exact hetzelfde als in het echte leven: een algoritme om je fiets op te pompen kan zijn:

1. Haal dop van het ventiel
2. Plaats pomp op ventiel
3. Begin te pompen

Eender welke andere volgorde van bovenstaande algoritme zal vreemde (en soms fatale) fouten geven.

C-Sharp

Om een algoritme te schrijven dat onze computer begrijpt dienen we een programmeertaal te gebruiken. Net zoals er ontelbare spreektaal in de wereld zijn, zijn er ook vele programmeertalen. **C#** (spreek uit 'siesjarp') is er een van de vele. In tegenstelling tot onze spreektaal moet een computertaal 'exact' zijn en moet het op ondubbelzinnige manier door de computer verstaan worden. C# is een taal die deel uitmaakt van de .NET (spreek uit 'dotnet') omgeving die meer dan 15 jaar geleden door Microsoft werd ontwikkeld (juli 2000).

Deze cursus werd geschreven in [Markdown](#). Helaas ondersteunt deze geen # in titels. Daardoor heet dit hoofdstuk "C-Sharp" en niet "C#". Niets aan te doen.

De geschiedenis en de hele .NET-wereld vertellen zou een cursus op zich betekenen en gaan we hier niet doen. Het is nuttig om weten dat er een gigantische bron aan informatie over .NET en C# online te vinden is, beginnende met docs.microsoft.com.

Met .NET Core kun je zowel webapplicaties, desktopapplicaties, microservices en Internet of Things toepassingen ontwikkelen.

Visual Studio en .NET Core installeren

We hebben twee zaken nodig om C# programma's te schrijven.

Met een editor, zoals Visual Studio, krijgen we ondersteuning bij het schrijven van de code die we later zullen uitvoeren. Een editor vergemakkelijkt dus het schrijven van een specifiek soort tekst.

De "SDK" (software development toolkit) bevat alles wat je nodig hebt om de code daadwerkelijk uitvoerbaar te maken.

Vooraleer je begint

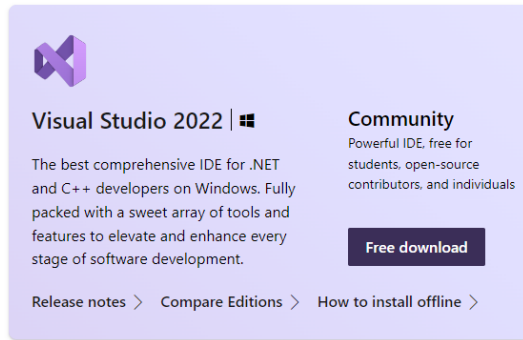
1. Voor deze opdrachten gebruiken we .NET Core 6.0, maar we gebruiken de (explicietere) template van .NET Core 5.0. Je moet dus beide SDK's installeren.
2. Sommige afbeeldingen hieronder dateren van de tijd van .NET Core 2.0 en Visual Studio 2017. Soms kunnen zaken er op jouw scherm dus wat anders uitzien dan in deze cursus.

Installatie

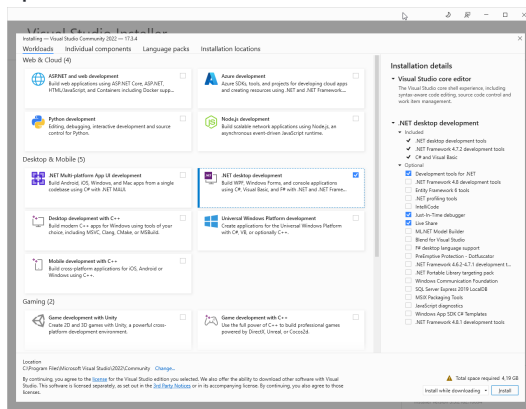
Visual Studio 2022 Community

1. <https://visualstudio.microsoft.com/downloads/>

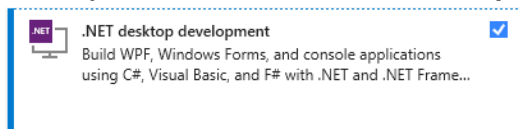
2. Kies voor de "community" versie



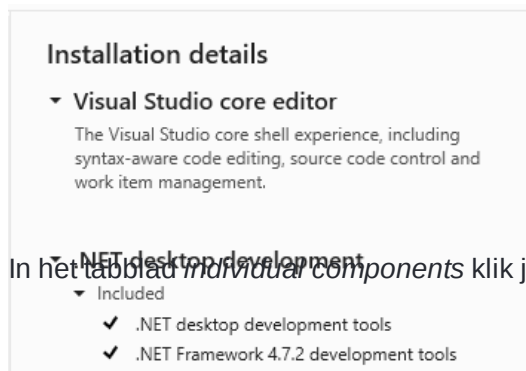
3. Het installatiescherm ziet er als volgt uit, in de volgende punten wordt beschreven welke elementen je specifiek moet aanduiden.



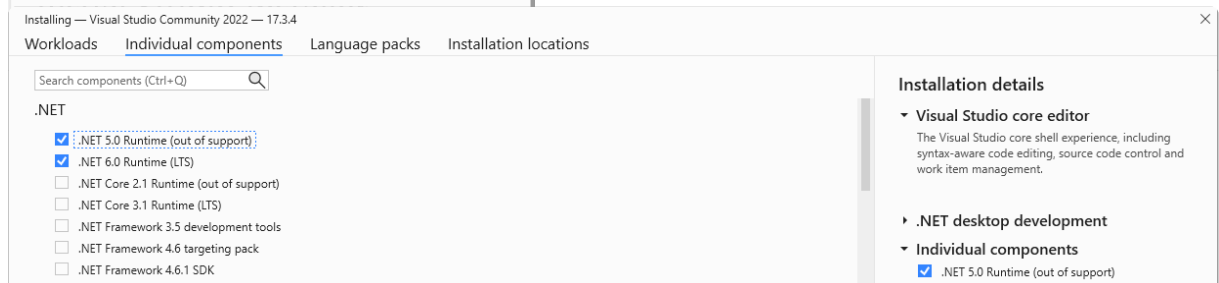
- Kies bij de installatie voor **.NET desktop development**



- Bij de *installation details* kies je voor volgende elementen

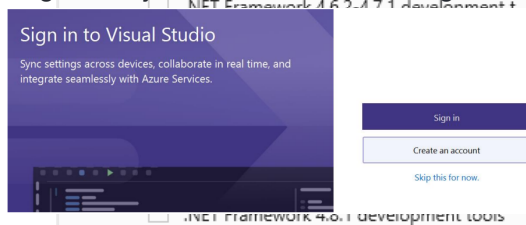


- In het tabblad **individual components** klik je **.NET 5.0 Runtime (out of support)** nog extra aan.



4. Vervolgens kies je rechts onderaan voor de knop **Install**

5. Log in met jouw account van de hogeschool, nl. **s-nummer@ap.be**



Opmerking

.NET 5.0 Runtime kan je ook afzonderlijk installeren via <https://dotnet.microsoft.com/en-us/download/dotnet/5.0>

Flowgorithm

Flowgorithm is een visuele programmeertaal. We zullen deze gebruiken om bepaalde concepten duidelijk zichtbaar te maken. Je moet de Windows Installer van [deze pagina](#) halen. Tijdens de installatie klik je gewoonweg elke keer op "Next".

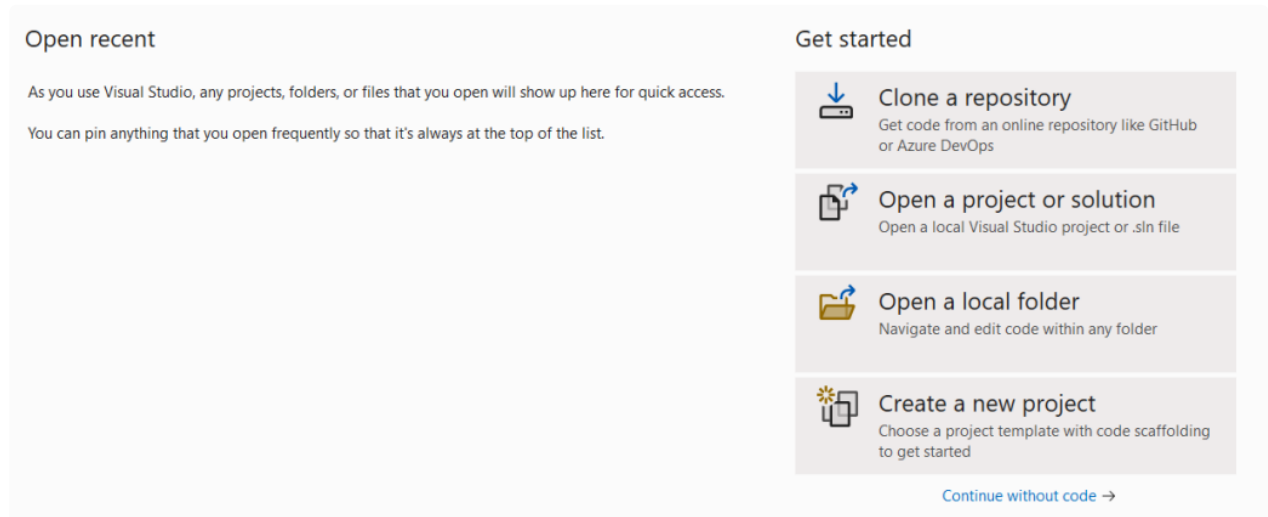
Gebruikers van Mac OS installeren dit best via [Wine](#) of in een virtuele machine. Als dit echt niet lukt, kunnen ze tijdelijk [FlowRun](#) gebruiken, maar deze software werkt wat anders en ondersteunt geen arrays.

Een voorbeeld van hoe je begint te werken met Flowgorithm kan je hier bekijken.

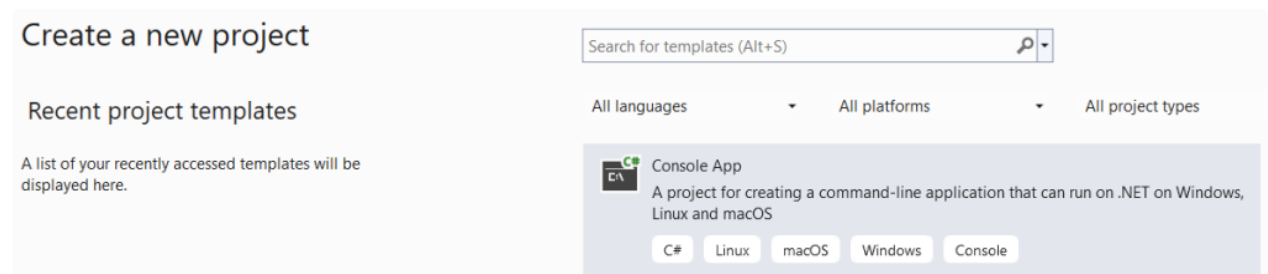
<https://www.youtube.com/watch?v=7SuYTvYCqOw>

Eerste gebruik Visual Studio

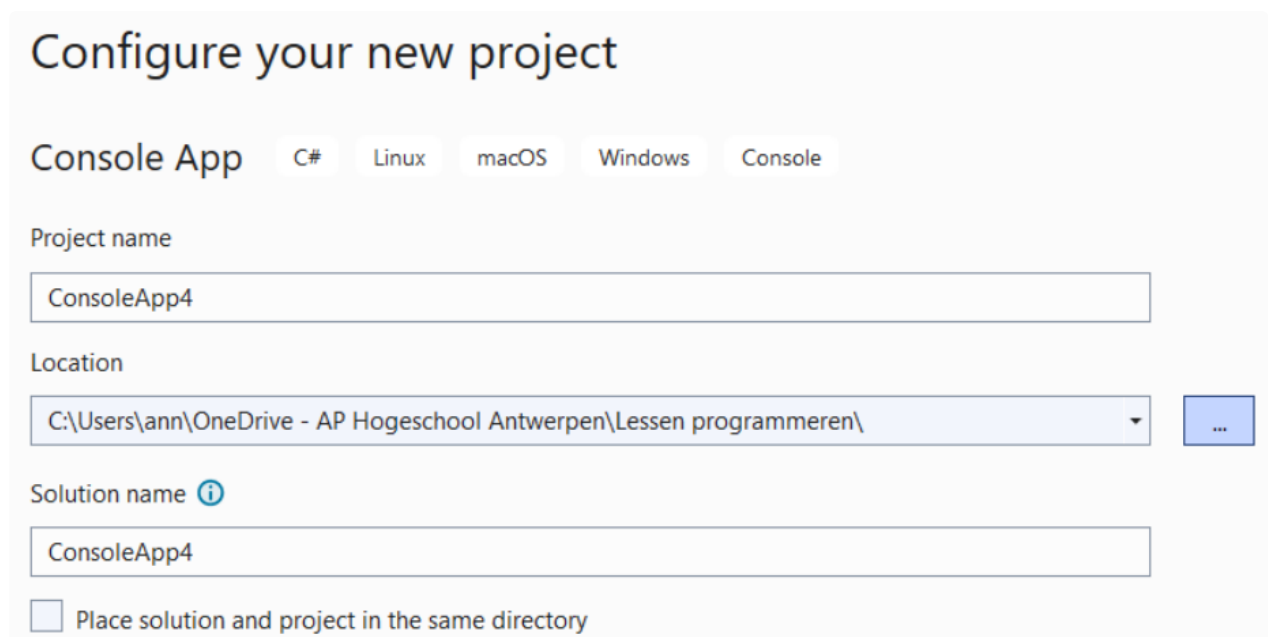
1. Kies voor **Create New Project**



2. Vervolgens selecteer je **Console App**



3. Je geeft je project een goede naam, hieronder dien je de **Project name** dan aan te passen en je zal zien dat de **Solution name** automatisch mee wordt gewijzigd.
Denk er ook aan om je project op de juiste plaats op te slaan door de **Location** te wijzigen.



4. In de volgende stap dien je als **Framework** .NET 6.0 te kiezen en de optie "Do not use top-level statements" **aan** te vinken.

Additional information

Additional information

Console App C# Linux macOS Windows Console

Framework ⓘ

.NET 6.0 (Long-term support)

☒ Do not use top-level statements ⓘ

Je settings moeten er zo uitzien.

Nadat je je project hebt aangemaakt, rechterklik je op je project in de Solution Explorer -> Properties -> Build -> General en schakel je "Implicit global usings" uit.

Search properties

Application

Build

General

Errors and warnings

Output

Events

Strong naming

Advanced

Package

Code Analysis

Debug

Resources

Platform target ⓘ

Specifies the processor to be targeted by the output file. Choose 'Any CPU' to specify that any processor is acceptable, allowing the application to run on the broadest range of hardware.

Any CPU

Nullable ⓘ

Specifies the project-wide C# nullable context. Only available for projects that use C# 8.0 or later.

Enable

Implicit global usings

☐ Enable implicit global usings to be declared by the project SDK.

Unsafe code ⓘ

☐ Allow code that uses the 'unsafe' keyword to compile.

Optimize code ⓘ

Enable compiler optimizations for smaller, faster, and more efficient output.

☐ Release

Je settings moeten er zo uitzien.

Code Snippet

Hierbij ga je ervoor zorgen dat je met afkortingen kan coderen die dan automatisch in het goede formaat worden omgezet.

Bv. `cr` (gevolgd door twee TABs) wordt dan `Console.ReadLine()`

Je kan volgende code kopiëren in een tekstbestand met als extensie `.snippet` of je kan het hieronder downloaden.



programmeren.snippet 697B
Binary


```

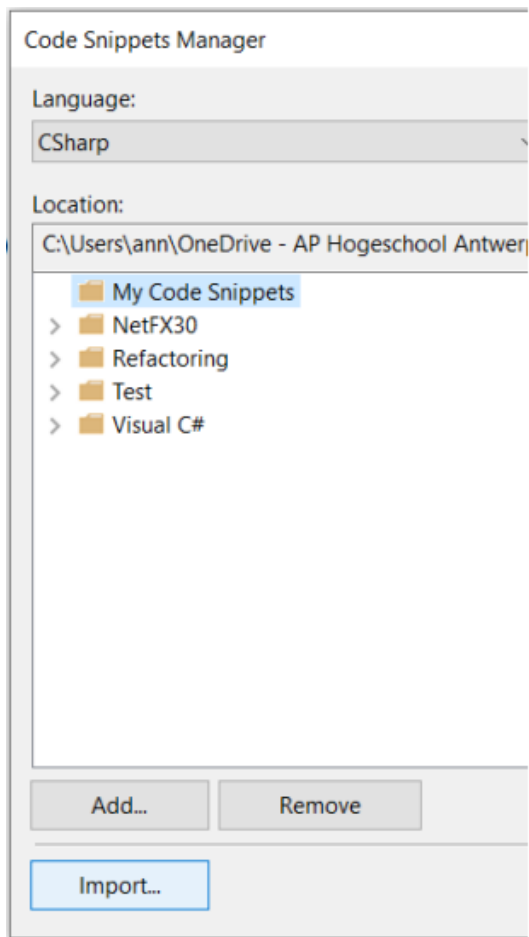
<?xml version="1.0" encoding="utf-8" ?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>cr</Title>
      <Shortcut>cr</Shortcut>
      <Description>Code snippet for Console.ReadLine</Description>
      <Author>Community</Author>

      <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
      </SnippetTypes>
    </Header>
    <Snippet>
      <Declarations>
        <Literal Editable="false">
          <ID>SystemConsole</ID>
          <Function>SimpleTypeName(global::System.Console)</Function>
        </Literal>
      </Declarations>
      <Code Language="csharp">
        <![CDATA[$SystemConsole$.ReadLine();$end$]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>

```

Vervolgens dien je binnen Visual Studio in het menu Tools naar Code Snippets Manager gaan.

Je kiest voor Language CSharp en klikt op de knop Import.



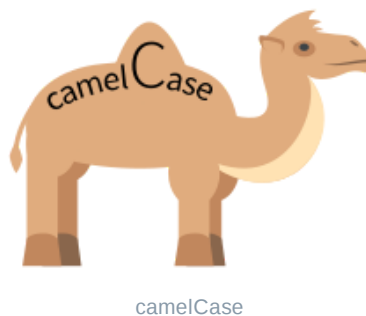
Je kiest het snippet bestand en voegt dit toe.

Een C# project maken in Visual Studio

We gebruiken Visual Studio om een C# programma te ontwikkelen. In dit onderdeel leren we hoe je een C# project opstart.

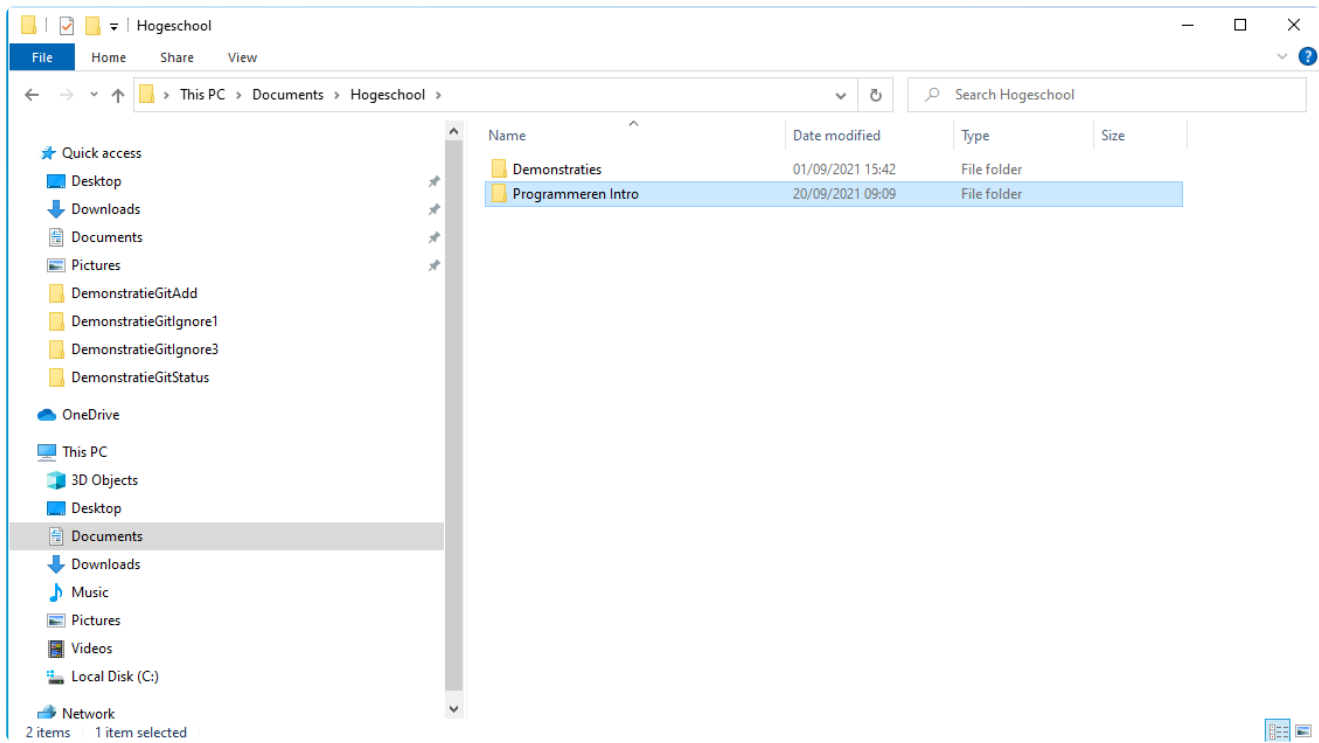
Het belang van het maken van afspraken voor het geven namen

1. We kunnen niet genoeg benadrukken hoe belangrijk het is om afspraken te maken en te volgen bij het schrijven van programma's. Goede afspraken zorgen ervoor dat je:
 1. code gemakkelijk kan delen met andere programmeurs; zonder afspraken kan je niet in een team werken;
 2. veel sneller fouten in je programma zal vinden;
 3. code leesbaar blijft, ook als je maanden later opnieuw bekijkt;
 4. sneller en beter gaan leren programmeren;
2. We vinden het warm water niet opnieuw uit en volgen de richtlijnen van Microsoft:
 1. een overzicht van alle afspraken vind je op [Naming Guidelines](#)
 2. een overzicht van algemene afspraken, wat hier van toepassing is, vind je op [General Naming Conventions](#)
3. De basis voor de naamgevingsafspraken zijn:
 1. **PascalNotatie**: geen spaties tussen de woorden en alle woorden beginnen met een hoofdletter: `EenProgrammaOmJeGoedBijTeVoelen`
 2. **camelCasenotatie**: geen spaties tussen de woorden en alle woorden, behalve het eerste, beginnen met een hoofdletter, bijvoorbeeld: `eenProgrammaOmJeGoedBijTeVoelen`



Rootmap

Ik raad jullie aan een rootmap te maken voor alle vakken die je op AP volgt. Je geeft die bijvoorbeeld de naam "AP" of "Hogeschool". Daarin maak je een map met de naam Programmeren Intro. In deze map zullen we alle projecten plaatsen die we in dit vak zullen maken.



Voor vakken waarin met Git gewerkt wordt, vermijden we mappen die automatisch gesynchroniseerd worden. Zet je rootmap dus niet in OneDrive of Dropbox of Google Drive. Dat kan voor technische problemen zorgen. Meer info vind je in [de cursus Git](#).

Fouten in je code

✓ Kennisclip voor deze inhoud



⚠ Je code kan niet gecompileerd en uitgevoerd worden zolang er fouten in je code staan.

Opzettelijk een fout maken

We willen zien wat er gebeurt als er een fout in het programma staat. We verwijderen het puntkomma aan het einde van het statement in de Main functie. We vervangen

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

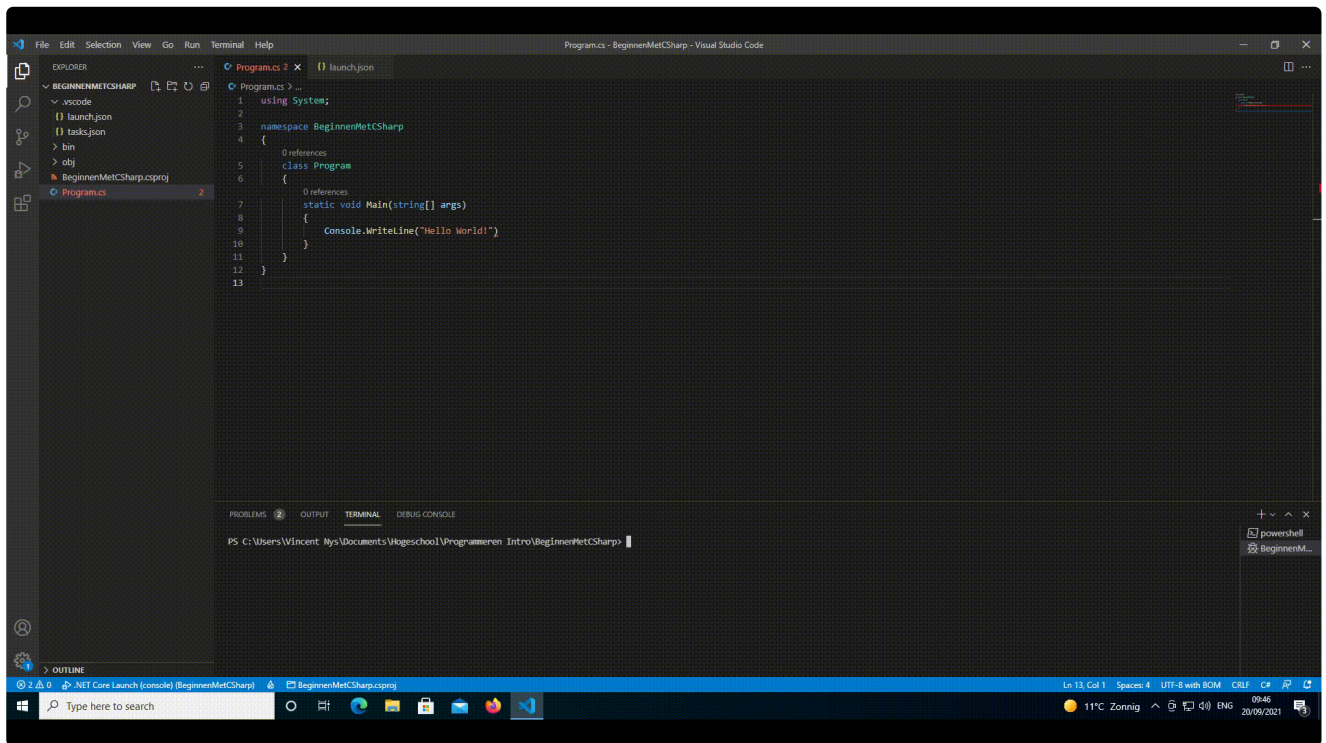
door

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

Het enige dat we hebben gewijzigd is de puntkomma op het einde.

Opzettelijke fout testen

Probeer het programma opnieuw uit te voeren. Je krijgt een melding dat er iets mis is. Vink eerst aan dat er onthouden moet worden wat je nu doet en geef dan aan dat je de fouten wil tonen via "Show Errors". Dit geeft je een overzicht, waarin je opmerkt dat er iets niet klopt op de regel en in de kolom waar de puntkomma ontbreekt. In de screenshot staat [9,46] en dat wijst op regel 9 van de code, positie 46. Waar de puntkomma moet staan, dus.



Je ziet ook dat het haakje net na "World!" rood onderlijnd is. Dat wijst op de positie van de fout. Let op: ook hier weet de ontwikkelomgeving niet precies wat je uiteindelijk wil, dus zie dit eerder als een hint.

Fout corrigeren

Plaats de puntkomma opnieuw op het einde van het statement op regel 9.

Opnieuw testen

Voer het programma opnieuw uit.

Meest voorkomende fouten

De meest voorkomende fouten in deze eerste weken zullen zijn:

- **Puntkomma** vergeten.
- **Schrijffouten** in je code RaedLine i.p.v. ReadLine.
- Geen rekening gehouden met **hoofdletter gevoeligheid** Readline i.p.v. ReadLine (zie volgende hoofdstuk).
- Per ongeluk **accolades verwijderd**.
- Code geschreven op plekken waar dat niet mag.

Je eerste stappen in C#

In dit onderdeel analyseren we het standaard programma dat Visual Studio voor ons in het bestand Program.cs gemaakt heeft. Verder zetten we de eerste stappen in het leren programmeren.



Kennisclip voor deze inhoud

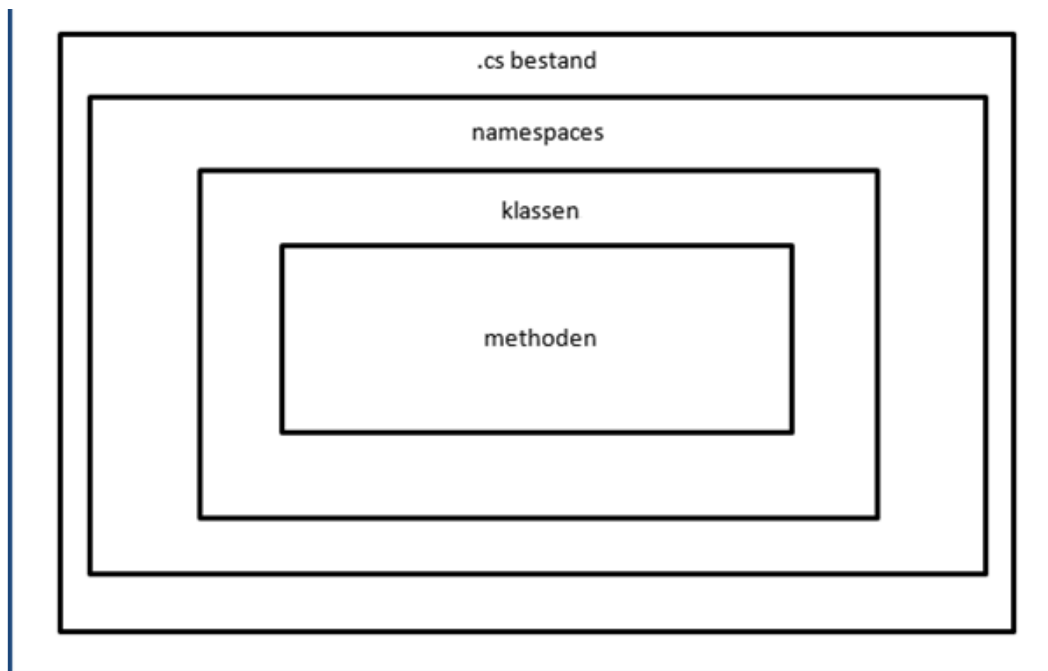
Doelstelling

1. weet je hoe de basisstructuur van een C# programma in elkaar steekt
2. ken je de basiselementen van het programmeren:
 1. **declaratie**: gegevens en variabelen
 2. **statement**: één instructie
 3. **functie** of **methode**: een reeks instructies om één specifiek doel te realiseren, bijvoorbeeld voor het bereken van de oppervlakte van een cirkel
In Object Oriented jargon spreekt men van een methode en vermits C# een OO taal is, zullen we in deze cursus verder het woord methode gebruiken. Als jullie JavaScript leren ga je het niet over een methode hebben maar over een functie omdat JavaScript geen strikte OO taal is.

De basisstructuur van een C# programma

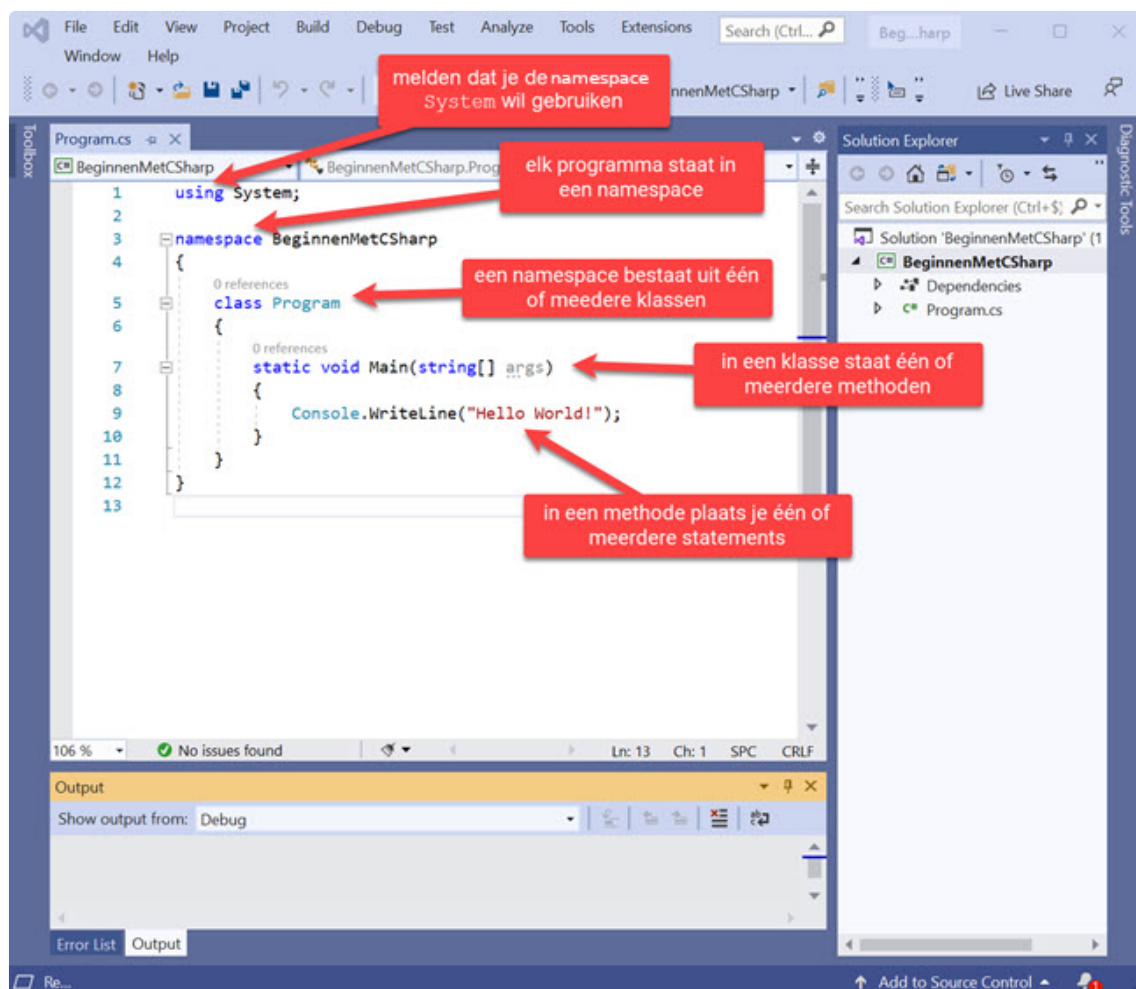
Schema





orde in .NET

Detail



Beschrijving van bovenstaande basisstructuur

1. **sleutelwoorden** (keywords)

In de editor zie je dat sommige woorden in blauw worden weergegeven. Visual Studio geeft alle sleutelwoorden in het blauw weergegeven. Sleutelwoorden zijn vooraf gedefinieerde, gereserveerde woorden die een speciale betekenis hebben voor de C# compiler. Je kan ze niet als namen (identifiers) in je programma gebruiken. Een lijst van alle gereserveerde sleutelwoorden vind je op [C# Keywords](#).

2. **namespace**

Letterlijk vertaald is dat een naamruimte. Deze structuur biedt de mogelijkheid om namen van o.a. klassen en methode - in programmeerjargon spreekt men van identifiers - in een namespace te groeperen. Je moet er dan enkel voor zorgen dat deze namen uniek zijn binnen de namespace. De regels en afspraken voor het geven van namen aan namespaces vind je op [Names of Namespaces](#).

Je kan een namespace vergelijken met een familienaam. Binnen de 'naamruimte' van een familienaam moeten de voornamen uniek zijn. Maar dezelfde voornamen kunnen in verschillende familienamen gebruikt worden: De namespace in het voorbeeld hierboven is

`BeginnenMetCSharp`. Visual Studio gebruikt de naam van het project voor het benoemen van deze namespace.

3. **klasse**

Gegevens (data) en algoritmes (programma's) worden ondergebracht in klassen. Dit maakt een programma overzichtelijker. In het tweede semester gaan we dieper in op klassen en objecten ('exemplaren' van een klasse). In het eerste semester gebruiken we klassen alleen om code te ordenen.

De klassenaam in het voorbeeld hierboven is `Program`. Deze klassenaam wordt door Visual Studio standaard gebruikt wanneer je een nieuw project aanmaakt.

De regels en afspraken voor het geven van namen aan klassen vind je op [Names of Classes, Structs, and Interfaces](#).

4. **methode**

In C# staat code altijd in een methode. Je kan geen statement schrijven zonder eerst een methode te hebben gemaakt. Hier moeten enkele zaken worden opgemerkt:

1. **naamgeving**

De namen van methoden worden in pascalnotatie geschreven, meer info op [Names of Methods](#).

2. **static**

Het sleutelwoord static is een *modifier*. Met een modifier geven we aan hoe we de methode willen gebruiken. Vermits we in het eerste semester klassen enkel en alleen gaan gebruiken om code te ordenen en niet om objecten te maken en te gebruiken, gaan we meestal de modifier static vóór de methode naam plaatsen, zoals in het voorbeeld hierboven. Met static bedoelen we dan deze methode tot de klasse behoort en niet tot de objecten die op basis van die klasse

worden gemaakt.

3. void

In C# spreken we van methoden. Maar een methode is eigenlijk een functie. Een functie retourneert meestal een waarde. Is dat niet zo moet je aangegeven dat de methode geen waarde retourneert door er void vóór de plaatsen zoals de Main methode in het voorbeeld hierboven.

4. Main

Dit is de naam van de methode. De moeder aller methoden in een C# programma draagt de naam Main methode. Elk programma dat je ooit gaat schrijven in C# moet een Main methode bevatten. Het is bovendien de enige methode die zal worden uitgevoerd als je het programma runt. Alle andere methode worden niet automatisch uitgevoerd. Wil je een bepaalde methode laten uitvoeren moet je die in de Main methode oproepen.

5. (string[] args)

Na de naam staan twee ronde haakjes. Tussen de ronde haken kan je *parameters* plaatsen. In deze parameters worden de argumenten die bij het oproepen van de methode worden meegegeven opgevangen. In het voorbeeld hierboven is dat string[] args. Dat wil zeggen dat je de Main methode kan oproepen en een reeks van argumenten kan meegeven. We komen hier later op terug. Je mag deze parameter gerust verwijderen vermits we pas later leren werken array's.

De Main methode hierboven bevat slechts 1 statement.

5. statement

Er is maar 1 statement. In het statement roepen we de WriteLine methode op die in de klasse Console staat. De klasse Console staat in de namespace met de naam System. Bovenaan, op regel 1, geven we aan dat we die bibliotheek in ons programma zullen gebruiken met het statement using System;

Het statement ziet er zo uit:

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

Het eerste wat we opmerken is dat een **statement eindigt met een ;** (een puntkomma).

In dit statement roepen we de methode WriteLine op en geven als *argument* de tekst "Hello World" mee. De methode WriteLine beschikt over 1 parameter die deze tekst aanneemt en voor ons in de console zal tonen.



Let erop dat je iedere 'zin' eindigt met een puntkomma.

Input/Output: ReadLine/WriteLine

✓ Kennisclip voor deze inhoud

Via Console.WriteLine kan je iets op het scherm laten zien:

```
using System;

namespace Demo1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("Hoi, ik ben het");
            Console.WriteLine("Wie ben jij?!");
        }
    }
}
```

ReadLine: Input van de gebruiker verwerken

Met de console kan je met een handvol methoden reeds een aantal interessante dingen doen.

Zo kan je bijvoorbeeld input van de gebruiker inlezen en bewaren in een variabele als volgt:

```
string result;
result = Console.ReadLine();
```

Bespreking van deze code:

```
string result;
```

- Concreet zeggen we hiermee aan de compiler: maak in het geheugen een plekje vrij waar enkel data van het type string in mag bewaard worden;
- Noem deze geheugenplek `result` zodat we deze later makkelijk kunnen in en uitlezen.

```
result = Console.ReadLine();
```

- Vervolgens roepen we de `ReadLine` methode aan. Deze methode zal de invoer van de gebruiker uitlezen tot de gebruiker op enter drukt.
- Het resultaat van de ingevoerde tekst wordt bewaard in de variabele `result` (denk eraan dat de toekenning van rechts naar links gebeurt).
- Een stukje code van de vorm `naam = waarde` heet een **toekenning**. Dit is niet hetzelfde als een wiskundige gelijkheid. In een toekenning plaats je een waarde aan de rechterkant. Aan de linkerkant schrijf je een naam die je aan deze waarde geeft.
- Als we een naam voor de eerste keer in het programma koppelen aan een waarde, spreken we soms over een **initialisatie**. Dit is ook een soort toekenning.

Je programma zou nu moeten zijn:

```
namespace Demo1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hoi, ik ben het!");
            Console.WriteLine("Wie ben jij?!");
            string result;
            result = Console.ReadLine();
        }
    }
}
```

Start nogmaals je programma. Je zal merken dat je programma nu een cursor toont en wacht op invoer. Je kan nu eender wat intypen en van zodra je op enter duwt gaat het programma verder (in dit geval stopt het programma hierna dus).

Input gebruiker verwerken en gebruiken

We kunnen nu invoer van de gebruiker, die we hebben bewaard in de variabele `result` gebruiken en tonen op het scherm.

```
Console.WriteLine("Dag ");
Console.WriteLine(result);
Console.WriteLine(" hoe gaat het met je?");
```

Op de tweede lijn hier gebruiken we de variabele `result` (waar de invoer van de gebruiker in bewaard wordt) als parameter in de `WriteLine` -methode.

Met andere woorden: de `WriteLine` methode zal op het scherm tonen wat de gebruiker even daarvoor heeft ingevoerd.

Je volledige programma ziet er dus nu zo uit:

```

using System;

namespace Demo1
{
    class Program
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Hoi, ik ben het!");
            Console.WriteLine("Wie ben jij?!");

            string result;
            result = Console.ReadLine();

            Console.WriteLine("Dag");
            Console.WriteLine(result);
            Console.WriteLine("hoe gaat het met je?");

        }
    }
}

```

Test het programma en voer je naam in wanneer de cursor knippert.

Voorbeelduitvoer (lijn 3 is wat de gebruiker heeft ingetypt)

```

Hoi, ik ben het!
Wie ben jij?!
tim [enter]
Dag
tim
hoe gaat het met je?

```

Aanhalingsteken of niet?

Wanneer je de inhoud van een variabele wil gebruiken in een methode zoals `WriteLine()` dan plaats je deze zonder aanhalingsteken! Bekijk zelf eens wat het verschil wordt wanneer je volgende lijn code `Console.Write(result);` vervangt door `Console.Write("result");`.

De uitvoer wordt dan:

```

Hoi, ik ben het!
Wie ben jij?!
tim [enter]
Dag
result
hoe gaat het met je?

```

Write en WriteLine

De `WriteLine` -methode zal steeds een line break (een 'enter') aan het einde van de lijn zetten zodat de cursor naar de volgende lijn springt.

De `Write` -methode zal geen enter aan het einde van de lijn toevoegen. Als je dus vervolgens iets toevoegt (met een volgende `Write` of `WriteLine`) **dan zal dit aan dezelfde lijn toegevoegd worden.**

Vervang daarom eens de laatste 3 lijnen code in je project door:

```
Console.Write("Dag");  
Console.Write(result);  
Console.Write("hoe gaat het met je?");
```

Voer je programma uit en test het resultaat. Je krijgt nu:

```
Hoi, ik ben het!  
Wie ben jij?!  
tim [enter]  
Dagtimhoe gaat het met je?
```

Wat is er verkeerd gelopen? Al je tekst van de laatste lijn plakt zo dicht bij elkaar? Inderdaad, we zijn spaties vergeten toe te voegen! Spaties zijn ook tekens die op scherm moeten komen (ook al zien we ze niet) en je dient dus binnen de aanhalingstekens spaties toe te voegen. Namelijk:

```
Console.Write("Dag ");  
Console.Write(result);  
Console.Write(" hoe gaat het met je?");
```

Je uitvoer wordt nu:

```
Hoi, ik ben het!  
Wie ben jij?!  
tim [enter]  
Dag tim hoe gaat het met je?
```

Opletten met spaties

Spaties zijn ook tekens die op scherm moeten komen (ook al zien we ze niet) en je dient dus binnen de aanhalingstekens spaties toe te voegen. Indien je deze erbuiten plaats dan heeft dit geen effect (je wist al uit het eerste hoofdstuk dat C# alle witregels negeert die niet tussen aanhalingstekens staan). *In volgend voorbeeld zijn de spaties aangegeven als liggende streepjes (_).*

Fout (de code zal werken maar je spaties worden genegeerd):

```
Console.Write("Dag_");  
Console.Write(result_);  
  
Console.Write("hoe gaat het met je?");
```

Correct:

```
Console.Write("Dag_");  
Console.Write(result);  
Console.Write("_hoe gaat het met je?");
```

Zinnen aan elkaar plakken

We kunnen dit hele verhaal een pak korter tonen. De plus-operator (+) in C# kan je namelijk gebruiken om variabelen van het type string aan elkaar te plakken. De laatste 3 lijnen code kunnen korter geschreven worden als volgt:

```
Console.WriteLine("Dag " + result + " hoe gaat het met je?");
```

Merk op dat result dus NIET tussen aanhalingstekens staat, in tegenstelling de andere stukken zin. Waarom is dit? Aanhalingstekens in C# duiden aan dat een stuk tekst moet beschouwd worden als tekst van het type string. Als je geen aanhalingsteken gebruikt dan zal C# de tekst beschouwen als een variabele met die naam.

Bekijk zelf eens wat het verschil wanneer je volgende lijn code vervangt door de lijn er onder:

```
Console.WriteLine("Dag "+ result + " hoe gaat het met je?");  
Console.Write("Dag "+ "result" + " hoe gaat het met je?");
```

Meer input vragen

Als je meerdere inputs van de gebruiker tegelijkertijd wenst te bewaren dan zal je meerdere geheugenplekken nodig hebben om de invoer te bewaren. Bijvoorbeeld:

```
Console.WriteLine("Geef leeftijd");  
string leeftijd; //eerste geheugenplekje aanmaken  
leeftijd = Console.ReadLine();  
Console.WriteLine("Geef adres");  
string adres; //tweede geheugenplekje aanmaken  
adres = Console.ReadLine();
```

Je mag echter ook de geheugenplekken al vroeger maken. In C# zet men de geheugenplek creatie zo dicht mogelijk bij de code waar je die plek gebruikt (zoals vorig voorbeeld), maar dat is geen verplichting. Dit mag dus ook:

```
string leeftijd; //eerste geheugenplekje aanmaken

string adres; //tweede geheugenplekje aanmaken
Console.WriteLine("Geef leeftijd");
leeftijd = Console.ReadLine();
Console.WriteLine("Geef adres");
adres = Console.ReadLine();
```


Kleuren in Console

✓ Kennisclip voor deze inhoud

Kleuren in console

Je kan in console-applicaties zelf bepalen in welke kleur nieuwe tekst op het scherm verschijnt. Je kan zowel de kleur van het lettertype instellen (via `ForegroundColor`) als de achtergrondkleur (`BackgroundColor`).

Je kan met de volgende expressies de console-kleur veranderen, bijvoorbeeld de achtergrond in blauw en de letters in groen:

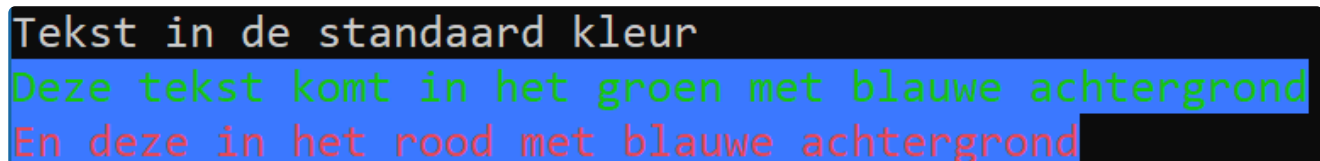
```
Console.BackgroundColor = ConsoleColor.Blue;  
Console.ForegroundColor = ConsoleColor.Green;
```

Vanaf dan zal alle tekst die je na deze 2 expressies via `WriteLine` naar het scherm stuurt met deze kleuren werken.

Een voorbeeld:

```
Console.WriteLine("Tekst in de standaard kleur");  
Console.BackgroundColor = ConsoleColor.Blue;  
Console.ForegroundColor = ConsoleColor.Green;  
Console.WriteLine("Deze tekst komt in het groen met blauwe achtergrond");  
Console.ForegroundColor = ConsoleColor.Red;  
Console.WriteLine("En deze in het rood met blauwe achtergrond");
```

Als je deze code uitvoert krijg je als resultaat:



```
Tekst in de standaard kleur  
Deze tekst komt in het groen met blauwe achtergrond  
En deze in het rood met blauwe achtergrond
```

Resultaat voorgaande code

i Kleur in console gebruiken is nuttig om je gebruikers een minder eentonig en meer informatieve applicatie aan te bieden. Je zou bijvoorbeeld alle foutmeldingen in het rood kunnen laten verschijnen.

Kleur resetten

Soms wil je terug de originele applicatie-kleuren hebben. Je zou manueel dit kunnen instellen, maar wat als de gebruiker slecht ziend is en in z'n besturingssysteem andere kleuren als standaard heeft ingesteld?!

De veiligste manier is daarom de kleuren te resetten door de `Console.ResetColor()` methode aan te roepen zoals volgend voorbeeld toont:

```
Console.ForegroundColor = ConsoleColor.Red;  
Console.WriteLine("Error!!!! Contacteer de helpdesk");  
Console.ResetColor();  
Console.WriteLine("Het programma sluit nu af");
```

Mogelijke kleuren

Alle kleuren die beschikbaar zijn zijn beschreven in `ConsoleColor` deze zijn:

- `ConsoleColor.Black`
- `ConsoleColor.DarkBlue`
- `ConsoleColor.DarkGreen`
- `ConsoleColor.DarkCyan`
- `ConsoleColor.DarkRed`
- `ConsoleColor.DarkMagenta`
- `ConsoleColor.DarkYellow`
- `ConsoleColor.Gray`
- `ConsoleColor.DarkGray`
- `ConsoleColor.Blue`
- `ConsoleColor.Green`
- `ConsoleColor.Cyan`
- `ConsoleColor.Red`
- `ConsoleColor.Magenta`
- `ConsoleColor.Yellow`

```
C:\Program Files\dotnet\dotnet.exe

Foreground color set to DarkBlue
Foreground color set to DarkGreen
Foreground color set to DarkCyan
Foreground color set to DarkRed
Foreground color set to DarkMagenta
Foreground color set to DarkYellow
Foreground color set to Gray
Foreground color set to DarkGray
Foreground color set to Blue
Foreground color set to Green
Foreground color set to Cyan
Foreground color set to Red
Foreground color set to Magenta
Foreground color set to Yellow
Foreground color set to White
=====
Background color set to Black
Background color set to DarkBlue
Background color set to DarkGreen
Background color set to DarkCyan
Background color set to DarkRed
Background color set to DarkMagenta
Background color set to DarkYellow
Background color set to Gray
Background color set to DarkGray
Background color set to Blue
Background color set to Green
Background color set to Cyan
Background color set to Red
```

Bron afbeelding : <https://www.c-sharpcorner.com/article/change-console-foreground-and-background-color-in-c-sharp/>

Oefeningen

Structuur van je oefeningen

Om je oefeningen ordelijk bij te houden, gaan we al een aantal zaken gebruiken die je pas verder in de cursus in detail leert. Hieronder krijg je een korte inleiding.

Stap 1: een project LaboOefeningen maken

Volg hiervoor de instructies in het document op DigitAP.

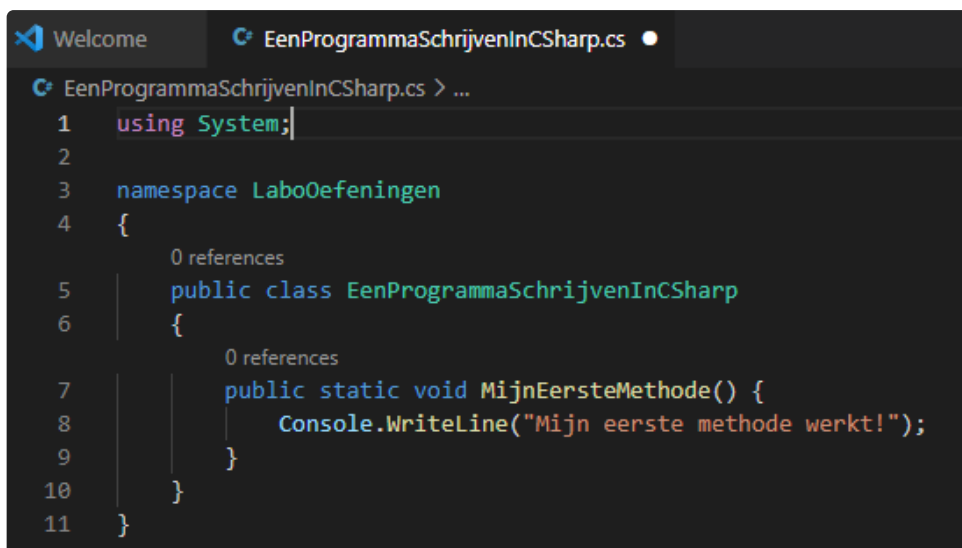
Stap 2: een klasse maken `EenProgrammaSchrijvenInCSharp`

Dit is nieuw. Klassen zijn een extra organisatie-eenheid van code. Ze spelen een grote rol in objectgeoriënteerd programmeren, maar kunnen ook gebruikt worden om stukken code verder te groeperen. Dit is voorlopig het enige dat wij er mee zullen doen. Via "File" maak je een nieuw bestand aan. Je kiest ervoor een nieuwe C#-klasse te maken met de naam `EenProgrammaSchrijvenInCSharp`.

Stap 3: een eigen methode maken

Een definitie van een methode bevat één taak die kan worden uitgevoerd. Onze eerste oefenprogramma's zullen telkens één methode zijn. De werking van methodes wordt verder in de cursus in detail omschreven. **Soms moet je eerst iets overnemen voor je de functie van elk onderdeelje kan begrijpen!**

Neem aandachtig volgende code over, zodat jouw bestand **exact** dit bevat:



```
1  using System;|
2
3  namespace LaboOefeningen
4  {
5      0 references
6      public class EenProgrammaSchrijvenInCSharp
7      {
8          0 references
9          public static void MijnEersteMethode() {
10             Console.WriteLine("Mijn eerste methode werkt!");
11         }
12     }
13 }
```

Belangrijke onderdelen:

- `using System;` maakt onderdelen uit de `System` namespace toegankelijk. Zonder dit kan je geen gebruik maken van `Console`.
- `namespace LaboOefeningen` geeft aan dat alles wat binnen de buitenste accolades (de symbolen op regels 4 en 11) staat behoort tot die namespace. Het is de "achternaam" van jouw code. Als er bijvoorbeeld twee stukken code `EenProgrammaSchrijvenInCSharp` zijn, kan je ze uit elkaar houden door te zeggen over welke namespace het gaat. Dit is hetzelfde als twee personen die dezelfde voornaam hebben, maar een andere achternaam.
- `public` : hier komen we pas een stuk later op terug. Voorlopig zetten we het standaard bij klassen en methoden.
- `class` : zie dit voorlopig als een verdere onderverdeling van je code. Eigenlijk zijn klassen veel meer dan dat, maar dat is voor later.
- `static` : dit is sowieso nodig wanneer we klassen enkel zien als een verdere onderverdeling van je code. We zetten het bij onze methoden.
- `void` : voorlopig zetten we dit altijd bij onze methoden. Het betekent ongeveer: "deze methode voert een commando uit, eerder dan een antwoord te geven op een vraag".
- `MijnEersteMethode` : dit is de naam van onze methode. We kiezen deze zelf.
- `()` : dit betekent dat de methode geen extra informatie nodig heeft om uit te voeren. Soms zetten we ook zaken tussen deze haakjes, de zogenaamde "parameters". Dit komt later aan bod.
- Alles tussen de accolades op regel 7 en 9: de "body" van de methode. Dit is eigenlijk wat er gebeurt als we de methode gebruiken.

Stap 4: je eigen methode oproepen

Standaard voert de code in de klasse `Program`, in de methode `Main` uit. Dat is het beginpunt van ons programma. We kunnen de body van `Main` aanpassen. We zullen hierin aangeven dat onze eigen methode moet worden opgeroepen als volgt:

```
static void Main(string[] args)
{
    EenProgrammaSchrijvenInCSharp.MijnEersteMethode();
}
```

We noemen de code in de body een "oproep" van de methode `MijnEersteMethode`. Hierin staat niet hoe die methode werkt. Er staat alleen dat we ze willen gebruiken.

Stap 5: opkuisen

Je mag nu zowel de definitie als de oproep van `MijnEersteMethode` wissen. In de oefeningen die volgen, maak je telkens nieuwe methodes. Test ze telkens uit door de **oproep** te vervangen. **Laat de definitie staan** wanneer je overgaat naar de volgende oefening. Later zullen we een keuzemenu maken dat ons toestaat makkelijk een oefening naar keuze te demonstreren.

Oefening: H1-MijnEersteProgramma

Leerdoelen

- een eigen programma kunnen uitvoeren
- input en output via `Console.ReadLine` en `Console.WriteLine`

Functionele analyse

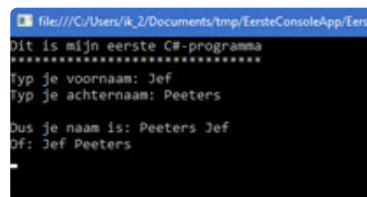
Binnen een zgn. dos-box wordt een titel weergegeven, nl. dit is mijn eerste c# programma.

Vervolgens wordt gevraagd je naam te noteren.

Wanneer je je naam hebt genoteerd en op enter hebt gedrukt, verschijnt de tekst “hallo [en je ingegeven naam]”.

Technische analyse

voorbeeldinteractie(s)



MijnEersteProgramma runt in de console

Technische hulp

maak een methode met de naam `MijnEersteProgramma`

Programmaverloop

Wat het lezen en schrijven van tekst betreft moet gebruik gemaakt worden `Console.WriteLine` en `Console.ReadLine`.

Testscenario's

- Probeer meer dan 200 tekens in te voeren
- Probeer geen tekst in te voeren

Oefening: H1-rommelzin

Leerdoelen

- een eigen programma kunnen uitvoeren
- input en output via `Console.ReadLine` en `Console.WriteLine`
- de computer leren zien als "domme verwerker"

Functionele analyse

Dit programma verwerkt tekst die door de gebruiker wordt ingetypt. Het print nieuwe berichten die deze tekst bevatten uit. Het print niet de berichten die je verwacht: het zal de antwoorden door elkaar halen en je favoriete kleur tonen wanneer het beweert je favoriete eten te tonen, enzovoort. De verbanden worden duidelijk uit de voorbeeldinteractie.

Technische analyse

Organisatie van de code

Schrijf dit programma als een methode met de naam `Rommelzin` binnen de klasse `EenProgrammaSchrijvenInCSharp`. Test uit door deze methode op te roepen binnen de `Main` methode.

voorbeeldinteractie(s)

```
Wat is je favoriete kleur?  
> blauw  
Wat is je favoriete eten?  
> spaghetti  
Wat is je favoriete auto?  
> Toyota Aygo  
Wat is je favoriete film?  
> Robocop 2  
Wat is je favoriete boek?  
> The Gone-Away World  
Je favoriete kleur is spaghetti. Je eet graag Toyota Aygo. Je lievelingsfilm is The Gone-Awa
```

Technische hulp

Programmaverloop

Per regel die getoond wordt op het scherm, maak je gebruik van `Console.WriteLine`. Per regel die je zelf intypt, maak je gebruik van `Console.ReadLine`. Zorg zelf voor de juiste ondersteunende code.

Testscenario's

- Test uit met een héél lang stuk tekst (meer dan 200 tekens) voor je favoriete kleur.
- Test uit met tekst met internationale karakters, bijvoorbeeld de ç.
- Ga na wat er gebeurt als je een lege regel invoert, dus als je meteen op ENTER duwt wanneer gevraagd wordt om invoer.

Oefening: H1-gekleurde-rommelzin

Leerdoelen

- de kleur van tekst in de console aanpassen
- herhaling van de leerdoelen uit H1-rommelzin

Functionele analyse

Dit programma werkt net als H1-rommelzin, maar elke regel die aan de gebruiker wordt getoond, krijgt een andere kleur. De namen van de kleuren die je gebruikt (in deze volgorde) zijn:

1. DarkGreen
2. DarkRed
3. DarkYellow
4. Blue
5. Cyan
6. Red

Technische analyse

Organisatie van de code

Schrijf deze oefening als een nieuwe methode met de naam `GekleurdeRommelzin` in de klasse `EenProgrammaSchrijvenInCSharp`. Test uit door deze methode op te roepen binnen de `Main` methode.

voorbeeldinteractie(s)

```
vincent@vincent-desktop:~/Projects/modeloplossingen-cursus-pro-oo/H0-gekleurde-rommelzin $ dotnet run
Wat is je favoriete kleur?
blauw
Wat is je favoriete eten?
spaghetti
Wat is je favoriete auto?
Toyota Aygo
Wat is je favoriete film?
Robocop 2
Wat is je favoriete boek?
The Gone-Away World
Je favoriete kleur is spaghetti. Je eet graag Toyota Aygo. Je lievelingsfilm is The Gone-Away World en je favoriete boek is Robocop 2.
```

De eerste regel behoort niet tot het programma. De rest moet er bij jou hetzelfde uitzien.

Technische hulp

Programmaverloop

Voor elke regel die in kleur getoond wordt, wissel je de voorgrondkleur. Op de juiste plaatsen in de code herstel je de oorspronkelijke kleuren van de terminal.

Testscenario's

- Test opnieuw uit met een kleur, maaltijd, auto, film en boek naar keuze.