

C# Essentials - Introductie

Inhoudsopgave

1. Over deze cursus	5
1.1. Wat is C# Essentials?	5
1.2. Competenties	5
1.2.1. Programmeur	5
1.2.2. Tester	5
1.3. Eindresultaat	5
1.4. Inhoud & planning	6
1.4.1. Permanente Evaluaties	6
1.5. Contactrijk onderwijs	7
1.5.1. Tijdens de lessen	7
1.5.2. Na de schooluren	7
2. Installatie	8
2.1. Download	8
2.2. Installatie	8
2.3. Aanmelden	9
3. Analytisch denken	12
3.1. Analytisch denken	12
3.1.1. Waarom?	12
3.1.2. Hoe?	12
3.2. Structuur	12
3.2.1. Probleemanalyse	12
3.2.2. Oplossingsstrategie	12
3.2.3. Algoritme/Actieplan	13
3.2.4. Uitvoeren/Implementeren (=programmeren)	13
3.2.5. Testen	13
3.2.6. Documenteren	13
3.3. Voorbeeld	13
3.3.1. Probleemanalyse	13
3.3.2. Oplossingsstrategie	13
3.3.3. Algoritme/Actieplan	13
3.3.4. Uitvoeren/Implementeren (=programmeren)	14
3.3.5. Testen	14
3.3.6. Documenteren	14
3.4. Oefening: Maak een algoritme voor...	14
3.4.1. Grootste getal	14

3.4.2. Ongeldige code's	15
3.4.3. Volgende vrije plaats	16
3.4.4. Dichtst bij nul	16
3.4.5. Eerste fout in een reeks	17
3.4.6. Langste herhaling	18
3.4.7. Eerste unieke waarde	18
3.4.8. Minimale wissels	19
3.4.9. Correct afgesloten haakjes	20
3.4.10. Slimme lichtschakelaar	20
4. .NET Framework & C#	22
4.1. Wat is een framework?	22
4.2. .NET Framework	22
4.2.1. Korte geschiedenis van .NET Framework	22
4.2.2. LTS	23
4.3. C#	23
4.3.1. Wat is C#?	23
4.3.2. Belangrijke kenmerken:	23
4.3.3. Rol van C# binnen .NET	24
5. Naming Conventions	26
5.1. Wat zijn naming conventions	26
5.2. Verschillende schrijfwijzen (casings)	26
5.2.1. PascalCase	26
5.2.2. camelCase	27
5.2.3. Overige	27
5.3. PXL Coding Conventions	27
6. Werken met Visual Studio	29
6.1. Wat is Visual Studio	29
6.2. Wat is een IDE?	29
6.2.1. Solution Explorer	29
6.2.2. Editor	30
6.2.3. Programma uitvoeren	30
6.2.4. Opslaan	30
6.3. Instellingen	31
6.3.1. IntelliSense	31
6.3.2. CodeLens	32
6.4. Extensions	33
6.4.1. Rainbow Braces	33
6.4.2. IndentRainbow	34

7. Consoleapplicaties	35
7.1. Wat is een consoleapplicatie	35
7.2. Een consoleapplicatie maken	35
7.2.1. Project aanmaken	35
7.2.2. "top-level statements"?	38
7.2.3. Uitvoeren	39
7.3. Projectstructuur	39
7.3.1. Program.cs	39
7.3.2. FirstConsoleApp.csproj	40
7.3.3. obj/ en bin/	40
7.3.4. Extra bestanden	40
7.4. Interactie met de gebruiker	40
7.4.1. Tekst tonen	41
7.4.2. Tekst lezen	42
7.5. Gegevens omzetten	42
7.6. Kleur aanpassen in de console	42
7.7. Demo	43
7.8. Build Errors	44
7.8.1. Instelling aanpassen	44
8. Oefeningen	45
8.1. Tutorial	45
8.2. Favoriete kleur	45
8.3. Leeftijd	46
9. Source Code Management	47
9.1. Versiebeheer	47
9.2. Git en GitHub	47
9.2.1. Git	47
9.2.2. GitHub	49
9.3. Workflow	50
9.3.1. Vier zone's	50
9.3.2. Basiscommando's	50
10. Git in Visual Studio	53
10.1. Basic Workflow in Visual Studio 2022	53
10.1.1. Een nieuwe repository maken	53
10.1.2. Een bestaande repository <i>clonen</i>	53
10.2. Tutorial	53
10.2.1. Remote repository	53
10.2.2. Clone met Visual Studio	54

10.2.3. Kennisclip	55
--------------------------	----

1. Over deze cursus

1.1. Wat is C# Essentials?

C# Essentials is een opleidingsonderdeel in het eerste semester van het graduaat programmeren. Het vak vormt een eerste kennismaking met programmeren en begeleidt je stap voor stap in het ontwikkelen van correcte en werkende desktopapplicaties. We gebruiken hiervoor de programmeertaal **C#**, ontwikkeld door Microsoft, en het geïntegreerde ontwikkelplatform **Visual Studio**.

Het vak is praktijkgericht: je leert door te doen. We starten met kleine consoletoepassingen, waarin je de basisprincipes van programmeren ontdekt. Van daaruit bouwen we verder richting grafische toepassingen met **WPF** (Windows Presentation Foundation). Onderweg maak je kennis met concepten zoals variabelen, methodes, foutopsporing en objectgeoriënteerd programmeren.

De belangrijkste doelstelling van dit vak is dat je leert denken en werken als een programmeur. Je leert niet alleen code schrijven, maar ook testen, verbeteren en logisch redeneren bij het oplossen van problemen.

1.2. Competenties

Aan het einde van dit opleidingsonderdeel beschik je over de basiscompetenties van een programmeur en een tester. Dat betekent dat je niet alleen leert **software schrijven**, maar ook leert **testen en verbeteren** zodat de kwaliteit van je code verzekerd blijft.

1.2.1. Programmeur

Je leert softwaretoepassingen ontwikkelen volgens de standaarden en afspraken binnen de organisatie. Concreet betekent dit dat je:

- programmeerproblemen leert analyseren en vertalen naar werkende oplossingen
- de programmeertaal C# correct gebruikt om logische en overzichtelijke code te schrijven
- eenvoudige desktopapplicaties bouwt die de gebruiker kan bedienen.

1.2.2. Tester

Je leert je eigen software kritisch bekijken en verbeteren. Concreet betekent dit dat je:

- testscenario's opstelt en uitvoert,
- fouten opspoort en oplost met behulp van de debugtools in Visual Studio,
- feedback verwerkt om je toepassingen steeds te verbeteren.

1.3. Eindresultaat

Wanneer je dit opleidingsonderdeel met succes hebt afgerond:

- kan je zelfstandig kleine C#-toepassingen programmeren
- begrijp je de werking van de belangrijkste programmeerconstructies (variabelen, condities, lussen, methodes, klassen, ...)

- kan je foutopsporing en eenvoudige tests uitvoeren
- heb je een stevige basis om in latere vakken complexere toepassingen en professionele software te ontwikkelen

1.4. Inhoud & planning

Alle leerstof vind je terug op **Blackboard**. Daar wordt het cursusmateriaal hoofdstuk per hoofdstuk opgebouwd in duidelijke modules:

Week	Module
1	<ul style="list-style-type: none"> • Introductie • Variabelen en bewerkingen
2-3	<ul style="list-style-type: none"> • Controlestructuren • Veelgebruikte Functies
4	<ul style="list-style-type: none"> • Iteraties
5	<ul style="list-style-type: none"> • Methodes & functies
6	<ul style="list-style-type: none"> • Klassen
7	<ul style="list-style-type: none"> • Error & foutopsporing
8-9	<ul style="list-style-type: none"> • Verzamelingen
10-14	<ul style="list-style-type: none"> • WPF

-  • Elke module bestaat uit korte hoofdstukken met uitleg, codevoorbeelden en oefeningen. Daarnaast voorzien we ook opdrachten die je zelfstandig of in groep uitwerkt.
- **De volgorde van de modules ligt vast.** Je legt de cursus volgens een strikte volgorde af waarbij je per module een controletoets krijgt. Enkel wanneer je **60% of meer** behaalt op deze controletoets kan je starten aan de volgende module.
- Je hoeft geen extra materiaal aan te kopen. Alles wat je nodig hebt om te slagen voor dit vak wordt voorzien via Blackboard.

1.4.1. Permanente Evaluaties

Week	Evaluatie
6	Consoleapplicatie met variabelen, controle- en herhalingsstructuren, veelgebruikte functies en methodes

Week	Evaluatie
12	Werkende GUI maken met WPF

1.5. Contactrijk onderwijs

Leren programmeren doe je niet door alleen te lezen, maar vooral door **veel te oefenen en feedback te krijgen**. Daarom staat interactie centraal in dit vak.

We geloven sterk in **contactrijk onderwijs**: veel oefenen in de les, onmiddellijk feedback krijgen en samen leren van elkaar's vragen. Dit betekent dat jouw aanwezigheid, inzet en interactie een groot verschil maken in je leerproces en je slaagkansen!

1.5.1. Tijdens de lessen

- Je krijgt volop de kans om vragen te stellen en samen te zoeken naar oplossingen.
- Fouten maken hoort erbij: we analyseren die samen en bekijken hoe je ze kan vermijden of oplossen.
- Er is ruimte om in kleine groepjes te overleggen en ideeën uit te wisselen.

1.5.2. Na de schooluren

Heb je buiten de les nog vragen? Dan kan je de lector aanspreken via **e-mail**. Je lector zal mogelijk je vraag via e-mail beantwoorden maar indien mogelijk kan deze ook tijdens één van de volgende lessen behandeld worden.

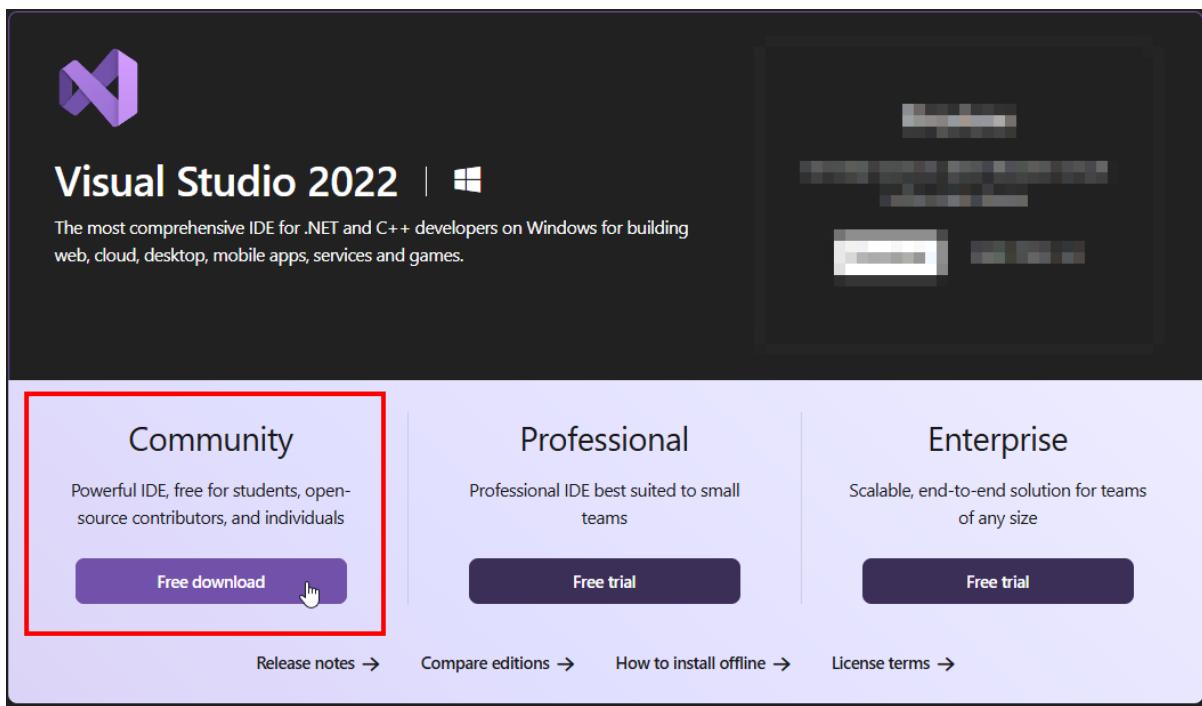


Wacht niet tot het einde van het semester om hulp te vragen. Programmeerproblemen los je sneller en beter op als je er **tijdig** mee aan de slag gaat.

2. Installatie

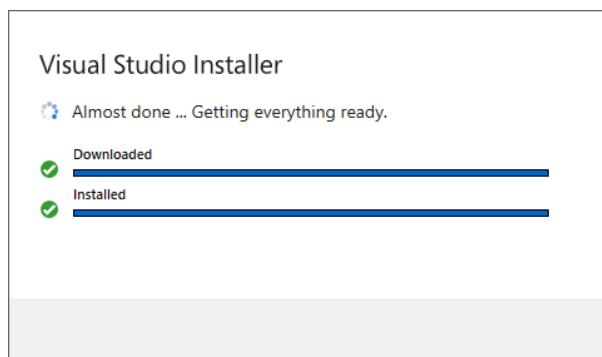
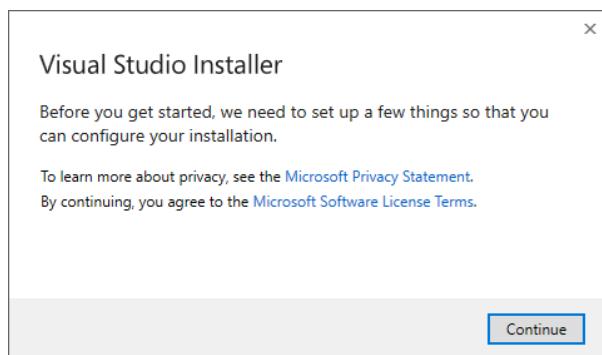
2.1. Download

Eerst ga je naar de [downloadpagina](#) van Visual Studio en download je **Visual Studio 2022 Community**.



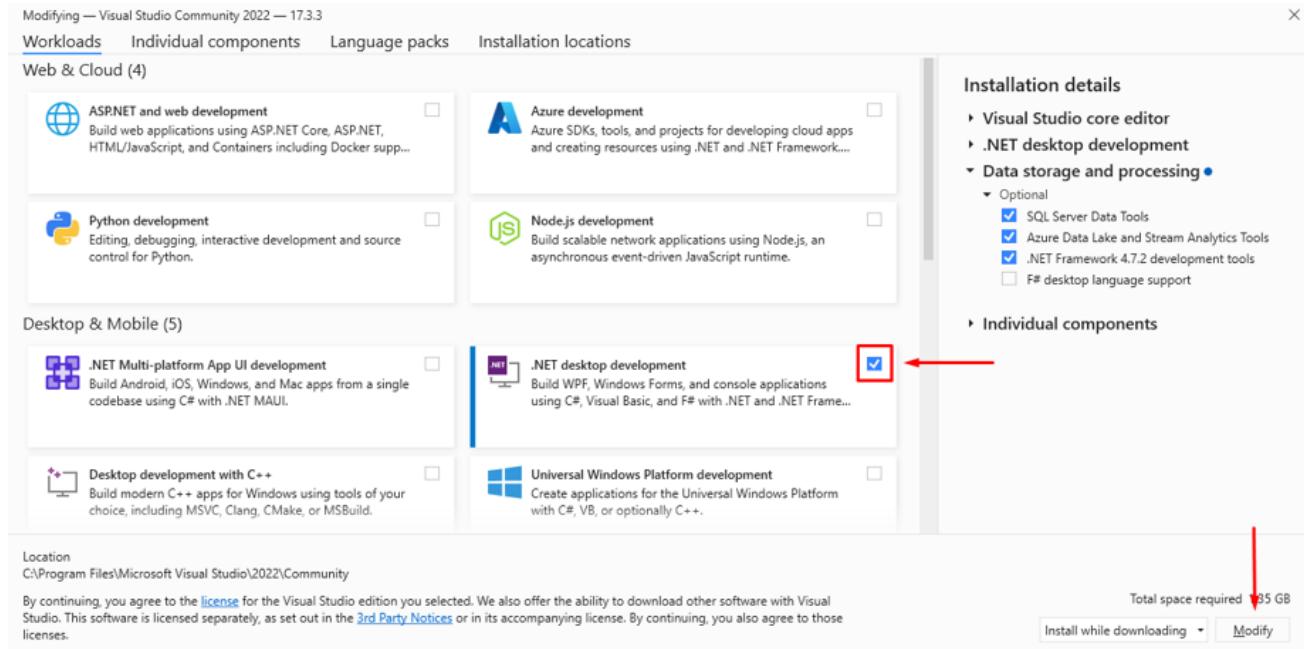
2.2. Installatie

1. Nu start je de installatie door het `VisualStudioSetup.exe` bestand te openen
2. Je klikt **Continue** om de installatie van Visual Studio Installer te starten

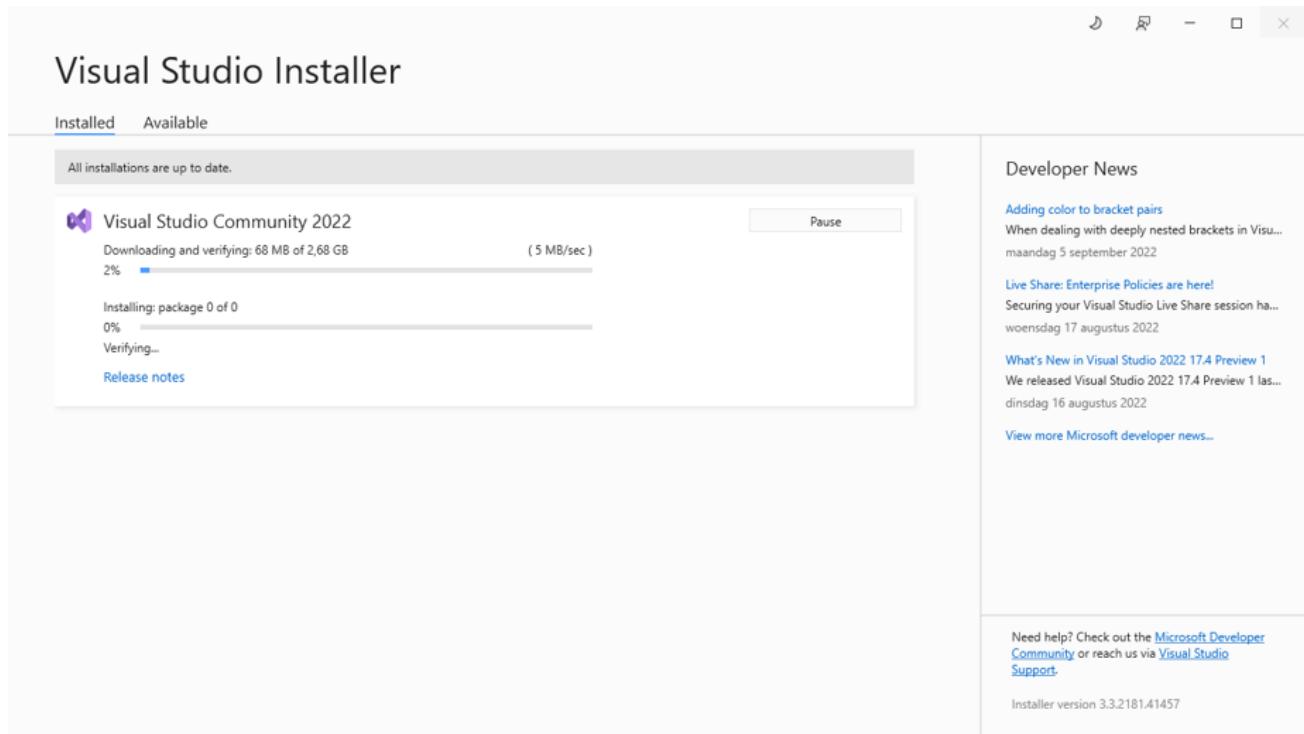


3. Tijdens de installatie vraagt Visual Studio welke onderdelen je wil installeren, kies hier voor **.NET**

desktop development



4. Als laatste klik je op **Modify** en wacht je tot de installatie voltooid is



⌚ Goed bezig, nu even geduld hebben!

2.3. Aanmelden

Om je instellingen te synchroniseren tussen verschillende installaties van Visual Studio (bv op je laptop en je desktop thuis) kan je ervoor kiezen om je aan te melden in Visual Studio:

1. Klik op de knop *Sign in*

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](#)

No account? [Create one!](#)

[Not now. maybe later.](#)

2. Selecteer Account voor werk of school

Een ander account gebruiken



Microsoft-account

E-mail, telefoon of Skype



Account voor werk of school

Toegewezen door uw organisatie

[Doorgaan](#)

3. Gebruik je PXL-account (voornaam.achternaam@student.pxl.be) om aan te melden

4. Ten slotte selecteer je de gewenste instellingen (deze kan je later eenvoudig aanpassen)

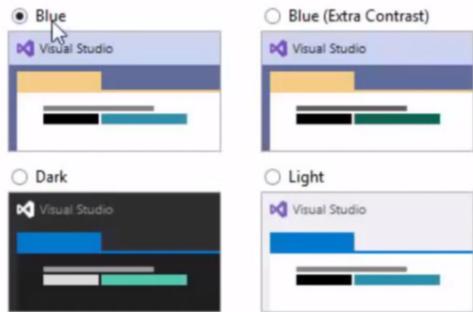
x

Visual Studio

Start with a familiar environment

Development Settings: General ▼

Choose your color theme



You can always change these settings later.

[Start Visual Studio](#)



In één van de volgende hoofdstukken leer je meer over de userinterface en verschillende features van Visual Studio

3. Analytisch denken

3.1. Analytisch denken



3.1.1. Waarom?

Als programmeur los je constant problemen op. Soms zijn dat kleine taken, zoals de som van twee getallen berekenen. Maar vaak zijn het grotere problemen, zoals een applicatie bouwen die de aanwezigheden van studenten bijhoudt. Daarom is het belangrijk om op voorhand exact te bepalen wat de gewenste uitkomst is en hoe je die uitkomst kan bekomen.

3.1.2. Hoe?

Door na te denken in stappen, kun je elk probleem op een gestructureerde manier aanpakken. Hetzelfde principe gebruik je bij programmeren: een probleem wordt opgedeeld in kleine, logische stappen die je één voor één kan oplossen. Zo een stappenplan noem je ook een "algoritme".

Om een goed programma te schrijven, moet je eerst weten:

- Wat is het probleem?
- Wat heb je als input?
- Wat wil je als output?
- Welke stappen zijn nodig om van input tot output te komen?

3.2. Structuur

Om op een gestructureerde manier tot een algoritme te komen, kan je het denkproces opdelen in 6 fases.

3.2.1. Probleemanalyse

- Wat is het probleem?
- Wat moet er gebeuren?
- Over welke gegevens beschik je?
- Welk resultaat wordt gevraagd?

3.2.2. Oplossingsstrategie

- Hoe los je het probleem op?
- De basisstructuren voor het programma worden hier gemaakt.

- Welke verfijnde technieken kan je hiervoor gebruiken?

3.2.3. Algoritme/Actieplan

- Stappenplan
- Stapsgewijs opgebouwd
- Een algoritme:
 - is duidelijk, ondubbelzinnig en volledig gedefinieerd
 - heeft uitgeschreven invoer- en uitvoerspecificaties
 - is gestructureerd
 - moet eindigen

3.2.4. Uitvoeren/Implementeren (=programmeren)

- Het effectief maken van het programma in bijv. C#

3.2.5. Testen

- Op basis van testgegevens
- Kan met hulptools, zoals unit testing, e.d.

3.2.6. Documenteren

- Commentaarregels om (stukken) code te verduidelijken
- Nuttig voor toekomstige aanpassingen
- Noodzakelijk bij werken in ontwikkelteams
- Documentatie/handleiding kan hieruit gegenereerd worden

3.3. Voorbeeld

Situatie: Je wil weten of een student geslaagd is op basis van zijn punten.

3.3.1. Probleemanalyse

- Input: lijst van punten op 20 voor verschillende olofs.
- Output: Geslaagd of niet geslaagd.

3.3.2. Oplossingsstrategie

- Bereken het gemiddelde.
- Vergelijk het met 10.

3.3.3. Algoritme/Actieplan

1. Tel alle punten op.
2. Deel door het aantal vakken.
3. Als het resultaat $\geq 10 \rightarrow$ Geslaagd.

4. Anders → Niet geslaagd.

3.3.4. Uitvoeren/Implementeren (=programmeren)

```
double[] scores = { 14.5, 9, 12 };
double sum = 0;

foreach(double score in scores)
{
    sum += score;
}

double average = sum / scores.Length;

if (average >= 10)
{
    Console.WriteLine("Geslaagd");
}
else
{
    Console.WriteLine("Niet geslaagd");
}
```

3.3.5. Testen

- Bereken manueel de uitkomst en vergelijk het resultaat met de output van de applicatie
- Simuleer scenario's met verschillende input

3.3.6. Documenteren

- Commentaar in de code.
- Duidelijke variabelennamen.

3.4. Oefening: Maak een algoritme voor...



Voor deze oefeningen is het niet de bedoeling dat je de uitkomst zoekt maar wel dat je een algoritme bedenkt waarmee de uitkomst **altijd** kan berekend worden

3.4.1. Grootste getal

Stel het grootst mogelijke getal samen dat bestaat uit exact 2 cijfers uit een reeks van 15 cijfers.



- De volgorde van de cijfers mag niet gewijzigd worden
- Het resultaat **moet** exact uit 2 cijfers bestaan

Voorbeeld

- 98765432111111 ॥ 98
- 81111111111119 ॥ 89
- 234234234234278 ॥ 78
- 818181911112111 ॥ 92



Stel jezelf de vraag of je algoritme ook zou werken voor een reeks van 20 cijfers, of 25... En wat als je het getal nu moet samenstellen uit 3 cijfers?

Oplossing

In het geval dat de reeks 15 cijfers bevat:

1. Zoek het grootste getal in de eerste 14 cijfers van de reeks ($14 = 15 - (2 - 1)$)
 - 15 is de lengte van de reeks
 - 2 is de lengte van het samen te stellen getal
2. Bepaal de positie van dit getal in de reeks
3. Zoek nu het grootste getal in de reeks waarvan de positie na die van het grootste getal ligt
4. Voeg de 2 getallen samen

3.4.2. Ongeldige code's

Zoek de ongeldige codes binnen een gegeven bereik van getallen.



- Een ongeldige code bestaat uit een reeks cijfers die exact 2 keer herhaald wordt:
 - 55 ॥ twee keer 5
 - 6464 ॥ twee keer 64
 - 123123 ॥ twee keer 123
- Een "reeks van cijfers" bevat altijd één of meerdere cijfers

Voorbeeld

- Het bereik 10–25 heeft 2 ongeldige codes ॥ 11 en 22
- Het bereik 95–115 heeft 1 ongeldige code ॥ 99
- Het bereik 998–1012 heeft 1 ongeldige code ॥ 1010
- Het bereik 1188511880–1188511890 heeft 1 ongeldige code ॥ 1188511885
- Het bereik 1698522–1698528 heeft geen ongeldige codes
- Het bereik 446443–446449 heeft 1 ongeldige code ॥ 446446

- Het bereik 38593856–38593862 heeft 1 ongeldige code ☹ 38593859

☒ Oplossing

1. Controleer elk getal binnen het bereik
2. Benader elk getal als tekst (bv 1212 is niet duizend tweehonderd en twaalf maar één-twee-één-twee)
3. Splits de tekst in 2 gelijke delen (bv 12 en 12)
 - een getal met een oneven lengte is dus per definitie een geldige code!
4. Als deel 1 hetzelfde is als deel 2 is het getal een ongeldige code

3.4.3. Volgende vrije plaats

Gegeven een lijst van bezette plaatsen, zoek de **eerste vrije plaats**.



- Elke plaats is ofwel bezet (`true`) of vrij (`false`)
- De lijst kan volledig vol zijn

Voorbeeld

- `true, true, false, true` ☹ index 2
- `false, true, true` ☹ index 0
- `true, true, true` ☹ geen vrije plaats



Denk na over wanneer je **mag stoppen met kijken**.

☒ Oplossing

1. Bekijk de plaatsen één voor één, beginnend bij de eerste.
2. Telkens wanneer je een plaats bekijkt, stel je jezelf de vraag:
 - a. Is deze plaats vrij?
3. Zodra je een vrije plaats tegenkomt:
 - a. Dat is het antwoord.
 - b. Je hoeft niet verder te kijken.
4. Als je alle plaatsen bekeken hebt en er is geen vrije plaats:
 - a. Dan bestaat er geen vrije plaats.

3.4.4. Dichtst bij nul

Zoek het getal dat het dichtst bij nul ligt in een reeks van getallen.



- De reeks bevat zowel positieve als negatieve getallen
- Het getal nul zelf kan ook voorkomen
- Als twee getallen even dicht bij nul liggen, mag je zelf bepalen welk je kiest

Voorbeeld

- -8, 4, -2, 10 ↗ -2
- 7, -1, 3 ↗ -1
- 5, -5, 9 ↗ 5 of -5



Denk in termen van “welk getal ligt **het minst ver** van nul?”.

Oplossing

1. Kies eerst één getal uit de reeks als voorlopig beste keuze.
2. Vergelijk dit getal met elk volgend getal:
 - a. Bepaal welk van de twee dichter bij nul ligt.
3. Als je een getal vindt dat dichter bij nul ligt dan je huidige keuze:
 - a. Vervang je keuze door dit nieuwe getal.
4. Na het vergelijken met alle getallen:
 - a. Is je laatste keuze het juiste antwoord.

3.4.5. Eerste fout in een reeks

Gegeven een oplopende reeks getallen, vind het **eerste** getal dat de volgorde doorbreekt.



- De reeks hoort telkens met +1 te stijgen
- Er is exact één fout

Voorbeeld

- 1, 2, 3, 5, 6, 7 ↗ 5
- 10, 11, 12, 13, 15, 16 ↗ 15



Denk telkens: “komt dit overeen met wat ik verwacht?”.

Oplossing

1. Begin bij het tweede getal van de reeks.

2. Vergelijk elk getal met het getal dat ervoor staat.

3. Stel jezelf telkens de vraag:

a. Is dit getal precies één groter dan het vorige?

4. Zodra dat niet meer klopt:

a. Heb je de eerste fout gevonden.

b. Je hoeft niet verder te kijken.

3.4.6. Langste herhaling

Zoek de langste opeenvolgende herhaling van hetzelfde teken in een tekst.



- De tekst bestaat uit letters en cijfers
- Het resultaat is de lengte van de langste herhaling

Voorbeeld

- aaabbcccdde ?? 4
- 11223333321 ?? 5
- abcdef ?? 1



Let op het moment waarop een herhaling **start** en **eindigt**.

Oplossing

1. Lees de tekst van links naar rechts.

2. Kijk telkens of het huidige teken hetzelfde is als het vorige.

3. Zolang dezelfde tekens blijven terugkomen:

a. Blijft de herhaling verder lopen.

4. Wanneer een ander teken verschijnt:

a. Eindigt de huidige herhaling.

b. Vergelijk deze herhaling met de langste die je tot nu toe zag.

5. Na het lezen van de volledige tekst:

a. Is de grootste herhaling die je tegenkwam het antwoord.

3.4.7. Eerste unieke waarde

Zoek het **eerste** element in een reeks dat maar één keer voorkomt.



- De volgorde is belangrijk

- Er is minstens één uniek element

Voorbeeld

- 3, 5, 3, 7, 5, 9 7
- a, b, a, c, b, d c



Denk na over hoe je kan controleren of iets **nog eens terugkomt**.

Oplossing

1. Bekijk de elementen één voor één, beginnend bij het eerste.
2. Voor elk element:
 - a. Ga na of datzelfde element later in de reeks nog voorkomt.
3. Als je een element vindt dat nergens anders voorkomt:
 - a. Dan is dat het eerste unieke element.
 - b. Je mag stoppen met zoeken.

3.4.8. Minimale wissels

Een automaat moet een bedrag teruggeven met zo weinig mogelijk munten.



- Je kent het bedrag
- Je kent de mogelijke muntwaarden

Voorbeeld

- 87 50 + 20 + 10 + 5 + 2
- 23 20 + 2 + 1



Denk telkens: "welke keuze brengt mij het snelst dichter bij nul?".

Oplossing

1. Begin met het volledige bedrag dat nog moet teruggegeven worden.
2. Kies telkens de grootste munt die nog in het resterende bedrag past.
3. Trek die munt af van het bedrag.
4. Herhaal dit tot er niets meer overblijft.
5. De gekozen munten vormen samen het wisselgeld.

3.4.9. Correct afgesloten haakjes

Bepaal of een tekst correct afgesloten ronde haakjes bevat.



- Enkel (en) zijn van belang
- Een sluitende haak mag nooit zonder open haak komen

Voorbeeld

- (()) ?? geldig
- (() ?? ongeldig
- ()) (?? ongeldig



Denk in termen van “openstaande” haakjes.

¶ Oplossing

1. Lees de tekst van links naar rechts.
2. Tel hoe vaak je een open haak ziet zonder dat ze al gesloten is.
3. Bij elke sluitende haak:
 - a. Controleer of er een open haak was om te sluiten.
 - b. Als dat niet zo is, is de tekst ongeldig.
4. Na het lezen van de volledige tekst:
 - a. Als er nog open haakjes over zijn, is de tekst ongeldig.
 - b. Als alles netjes gesloten is, is de tekst geldig.

3.4.10. Slimme lichtschakelaar

Een gang bevat lampen die om en om aan en uit staan. Bij elke druk op een knop wisselt elke lamp van toestand.

Bedenk een algoritme om te bepalen hoeveel lampen aan staan na **N** drukken.



- De beginstatus is gekend
- Je hoeft niets letterlijk te simuleren



Probeer te zien wat er **altijd opnieuw gebeurt**.

¶ Oplossing

1. Observeer wat er gebeurt bij één druk op de knop:

- a. Elke lamp verandert van toestand.
2. Observeer wat er gebeurt bij twee drukken:
 - a. Elke lamp komt terug in zijn oorspronkelijke toestand.
3. Hieruit volgt een patroon:
 - a. Bij een even aantal drukken verandert er niets.
 - b. Bij een oneven aantal drukken wordt alles omgekeerd.
4. Gebruik dit patroon om het aantal brandende lampen te bepalen.

4. .NET Framework & C#

4.1. Wat is een framework?

Een framework is een verzameling herbruikbare code die je helpt om sneller en gestructureerde software te ontwikkelen. Je kan het zien als een bouwpakket met:

- Kant-en-klare klassen en functies (zoals List<T>, File, HttpClient, ...)
- Standaardregels en -structuren, zodat je weet hoe je moet programmeren
- Bibliotheeken (zoals System.IO, System.Net, System.Linq, ...)

Een framework neemt je veel werk uit handen. Het zorgt er bijvoorbeeld voor dat jij geen eigen geheugenbeheer of netwerkprotocollen moet programmeren.

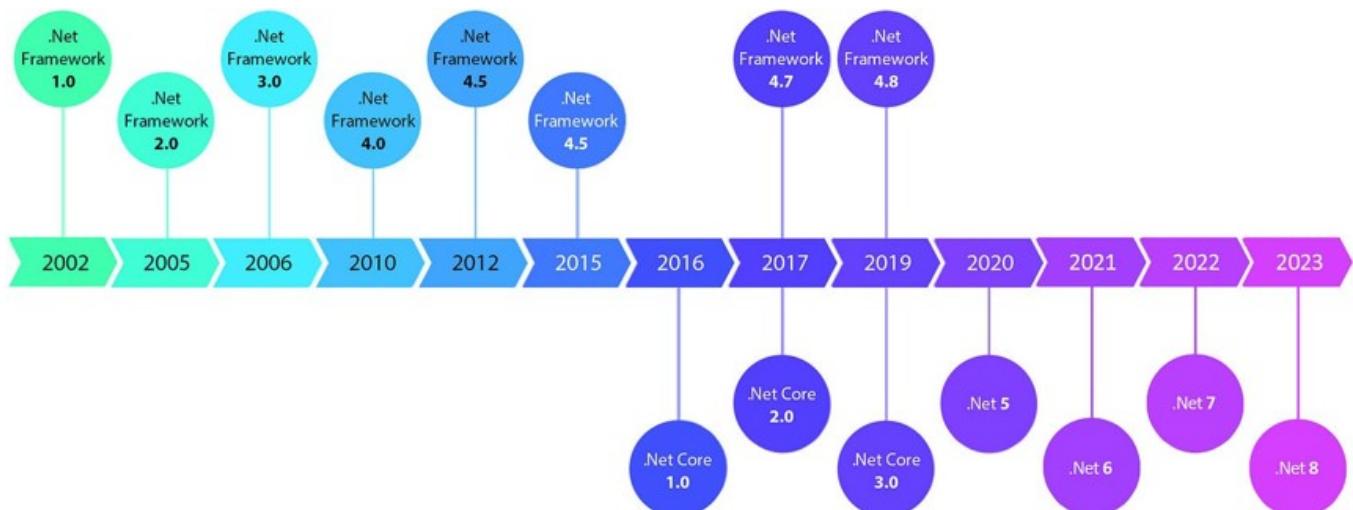
4.2. .NET Framework

.NET (uitgesproken als "dotnet") is het softwareframework van Microsoft.

.NET is the free, open-source, cross-platform framework for building modern apps and powerful cloud services.

4.2.1. Korte geschiedenis van .NET Framework

- **2002:** Eerste release van .NET Framework 1.0 door Microsoft
 - Gericht op Windows desktop-applicaties (Windows Forms) en ASP.NET voor web.
- **2005-2010:** Versies 2.0 t.e.m. 4.0
 - Introductie van o.a. generics, LINQ, Entity Framework, WPF, WCF, async programming.
- **2016:** Introductie van .NET Core
 - Lichte, cross-platformversie van .NET, los van het originele framework.
- **2020:** Microsoft kondigt .NET 5 aan
 - Start van het “unified platform” waarin .NET Core en .NET Framework samensmelten.
 - Sindsdien noemen we het gewoon .NET (zonder “Core”).



4.2.2. LTS

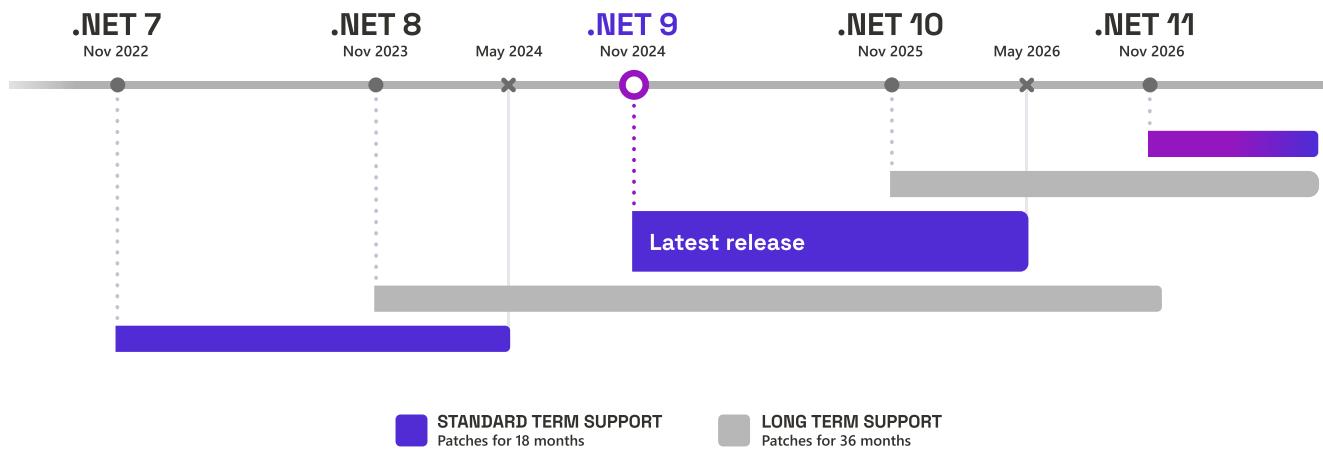
Bijna elk jaar wordt een nieuwe versie van .NET gereleased. Er zijn echter 2 releasetypes:

Type release	Ondersteuning	Doelgroep
LTS	3 jaar	Bedrijven en productieomgevingen
Current	± 18 maanden	Ontwikkelaars die de nieuwste features willen

LTS staat voor Long-Term Support. Dat betekent dat Microsoft die versie langer blijft ondersteunen met beveiligingsupdates en bugfixes.

Waarom is dit belangrijk?

- In een bedrijfsomgeving kies je meestal voor een LTS-versie: die is stabiel en krijgt lang ondersteuning.
- Tijdens ontwikkeling of testen kan je een Current-versie gebruiken om met de nieuwste mogelijkheden te experimenteren.



4.3. C#

4.3.1. Wat is C#?

C# (uitgesproken als "see sharp") is een moderne, objectgeoriënteerde programmeertaal ontwikkeld door Microsoft. Het is DE belangrijkste programmeertaal voor het .NET-platform.

4.3.2. Belangrijke kenmerken:

- Sterk getypeerd (je moet het type van variabelen kennen en respecteren)
- Ondersteunt objectgeoriënteerd programmeren (OOP)
- Wordt gebruikt voor desktop-, web-, cloud-, mobile- en gameontwikkeling

- Goede integratie met Visual Studio en .NET

4.3.3. Rol van C# binnen .NET



C# is de **taal** die je gebruikt om programma's te schrijven. Het .NET Framework of platform is de **omgeving** waarin die programma's worden uitgevoerd.

Een eenvoudige vergelijking:

- **C#** = een klusjesman die een herstelling komt uitvoeren
- **Het .NET-platform** = de gereedschapskist van de klusjesman

De klusjesman voert de herstelling nog steeds zelf uit, maar dankzij het gereedschap in zijn koffer kan hij dit sneller en efficiënter doen. Het grote voordeel is dat elke andere klusjesman de taak eveneens kan uitvoeren, zolang hij beschikt over hetzelfde gereedschap.

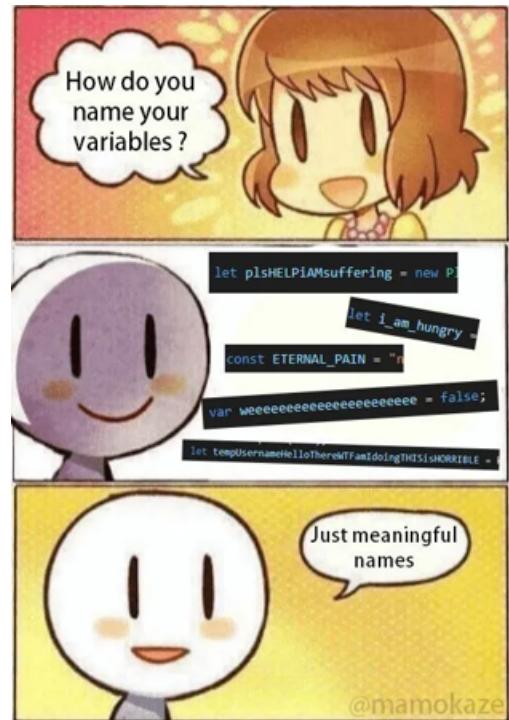


Naast C# ondersteunt het .NET Framework nog andere programmeertalen zoals F# en Visual Basic maar deze worden niet behandeld in deze cursus.



5. Naming Conventions

5.1. Wat zijn naming conventions



Naming conventions zijn afspraken over hoe we namen geven aan dingen in code, zoals klassen, variabelen, methodes en properties. Ze zorgen ervoor dat code:

- leesbaar is voor andere programmeurs,
- makkelijker te onderhouden is,
- er consistent uitziet in grotere projecten.

Een goede naam vertelt wat iets doet of wat het voorstelt.

```
int age = 25;    // Goed
int x = 25;      // Slecht: wat betekent x?
```

5.2. Verschillende schrijfwijzen (casings)

Er bestaan meerdere manieren om woorden in namen te schrijven. Dit noemen we casing styles. Hieronder leggen we de bekendste uit.

5.2.1. PascalCase

- Elke beginletter van een woord is een hoofdletter.
- Geen spaties of underscores.
- Vaak gebruikt voor: klassen, methodes, properties

```
public class StudentDetails
{
```

```
public string FullName { get; set; }  
public void PrintDetails() { }  
}
```

5.2.2. camelCase

- Eerste woord begint met een kleine letter.
- Volgende woorden beginnen met een hoofdletter.
- Vaak gebruikt voor: variabelen, parameters, velden

```
int studentAge = 21;  
string firstName = "Alice";
```

5.2.3. Overige

snake_case

- Woorden worden gescheiden door een underscore (_).
- Geen hoofdletters.
- Komt vaak voor in andere talen zoals Python of in databasenamen.

kebab-case

- Woorden worden gescheiden door een koppelteken (-).
- Komt vooral voor in bestandsnamen of URL's.
- **Niet geldig in C#-code!**

5.3. PXL Coding Conventions

In deze cursus volgen we de [PXL Coding Conventions](#). Deze zijn gebaseerd op de algemene richtlijnen van Microsoft en worden dus ook veruit het meest gebruikt. Belangrijk om te onthouden is dat we alle benamingen in onze code **in het Engels** schrijven!



"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."

6. Werken met Visual Studio

6.1. Wat is Visual Studio



Visual Studio is een programma dat speciaal gemaakt is om software te ontwikkelen. Je gebruikt het om:

- code te schrijven
- fouten in de code op te sporen
- je programma uit te voeren

Visual Studio helpt je bij elke stap in het proces.

6.2. Wat is een IDE?

Visual Studio is een voorbeeld van een **IDE**. Dat betekent **Integrated Development Environment**, of in het Nederlands: een geïntegreerde ontwikkelomgeving.

Een IDE bevat verschillende onderdelen die samenwerken:

6.2.1. Solution Explorer

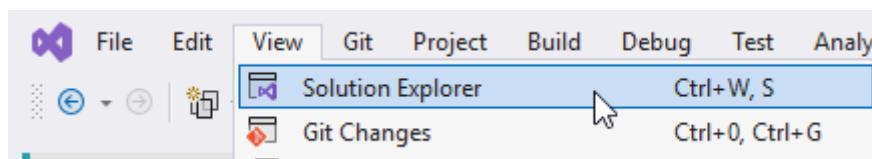
Dit is het overzicht van je project:

- Je ziet de bestanden in je project (zoals `Program.cs`)
- Je kan bestanden openen of nieuwe toevoegen



Een **solution** is een soort map die één of meerdere projecten kan bevatten. In de meeste gevallen begin je met slechts één project.

Als het Solution Explorervenster niet zichtbaar is of gesloten werd, kan je dit venster altijd terug opnieuw openen via het **View**-menu



6.2.2. Editor

In de editor schrijf je de code:

- De editor toont de inhoud van een bestand
- Je kan hier je code aanpassen
- Fouten worden automatisch aangeduid

Bovenaan in de editor worden de geopende bestanden als tabbladen weergegeven. Het huidige bestand wordt als actief tabblad getoond.

6.2.3. Programma uitvoeren

Eens je code geschreven is, wil je natuurlijk testen of ze werkt.

1. Klik op de groene **Startknop** bovenaan.
2. Of druk op de sneltoets **F5**.
3. Er opent een zwart venster: dit is het **consolevenster** waarin je programma draait.



Als je een fout hebt gemaakt, dan stopt de uitvoering en zie je een foutmelding. Dit is normaal — we leren later hoe we fouten kunnen oplossen.

The screenshot shows the Visual Studio IDE interface. The code editor window displays the file 'Program.cs' with the following content:

```
1  namespace PxConsoleApp;
2  {
3      internal class Program
4      {
5          static void Main(string[] args)
6          {
7              Console.WriteLine("Hello, World!");
8          }
9      }
}
```

The 'Solution Explorer' window on the right shows a single project named 'PxConsoleApp' containing a file 'Program.cs'. A red box highlights the top menu bar, the code editor area, and the Solution Explorer area.

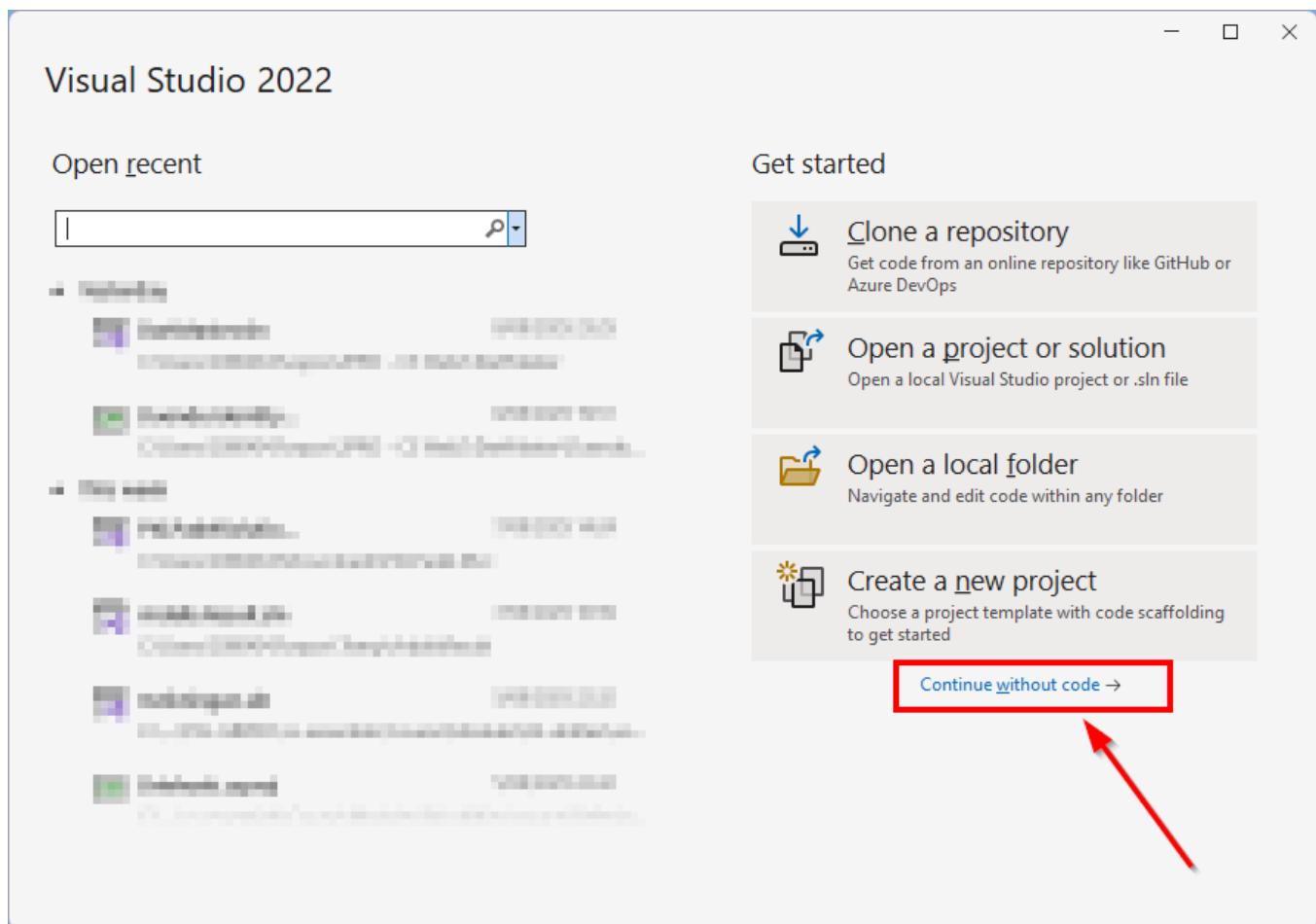
6.2.4. Opslaan



Vergeet niet om regelmatig op te slaan! Dit doe je via: **File > Save All** of via sneltoets **Ctrl + Shift + S**.

6.3. Instellingen

Nu je de belangrijkste onderdelen van de IDE kent is het tijd om Visual Studio een eerste keer te starten! Omdat je eerst enkele instellingen moet aanpassen kies je deze keer voor `Continue without code` in het startscherm van Visual Studio.



6.3.1. IntelliSense

Visual Studio helpt je tijdens het typen van code met slimme suggesties. Deze functie heet **IntelliSense**:

- Het toont een lijst met mogelijke commando's of variabelen.
- Je kan met de pijltjestoetsen een keuze maken en op `Tab` drukken om automatisch aan te vullen.
- Dit versnelt je werk en voorkomt typefouten.

[IntelliSense] | *intellisense.gif*

AI

In recente versies van Visual Studio wordt AI gebruikt om IntelliSense verder te verfijnen. Zo worden er soms al suggesties gedaan nog voordat je bent begonnen met typen! Voor een startende programmeur lijkt ons dit wat teveel van het goede. We raden je aan om deze feature uit te schakelen.

Volg deze stappen:

1. Ga naar het menu: `Tools → Options`

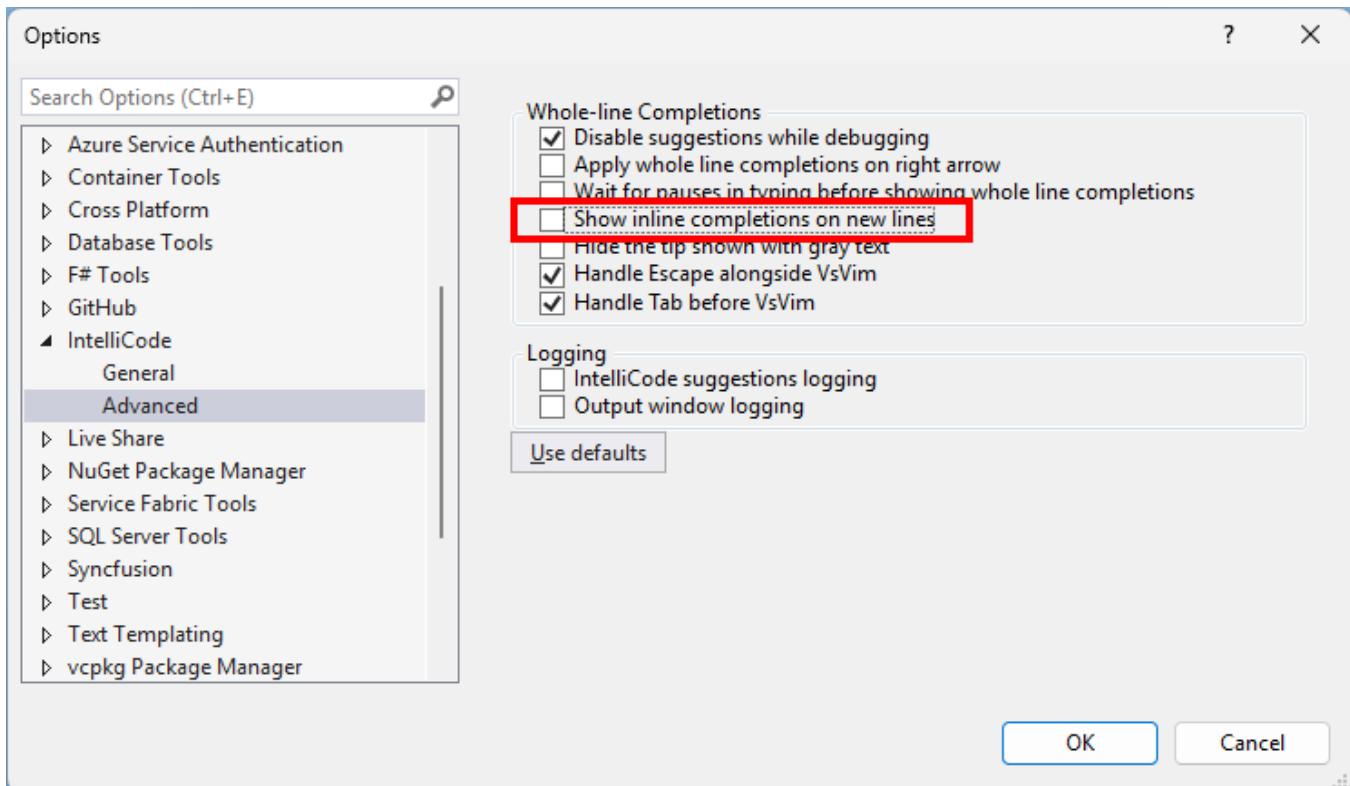
2. Navigeer links naar: **IntelliCode** → **Advanced**

3. Zoek de optie:

Show inline completions on new lines

Schakel deze optie **uit**

4. Klik op **OK** om te bevestigen



6.3.2. CodeLens

CodeLens is een functie in Visual Studio die je helpt om de context van je code te begrijpen, zonder de editor te verlaten. Het toont je relevante informatie zoals verwijzingen naar code-elementen, wijzigingen, bugs, werkitems, codebeoordelingen en eenheidstests, direct boven de code. Hoewel dit voor een ervaren programmeur soms handig kan zijn zorgt dit bij startende programmeurs vaak voor verwarring. Daarom raden we je aan om ook deze functie voorlopig uit te schakelen.

1. Ga naar het menu: **Tools** → **Options**

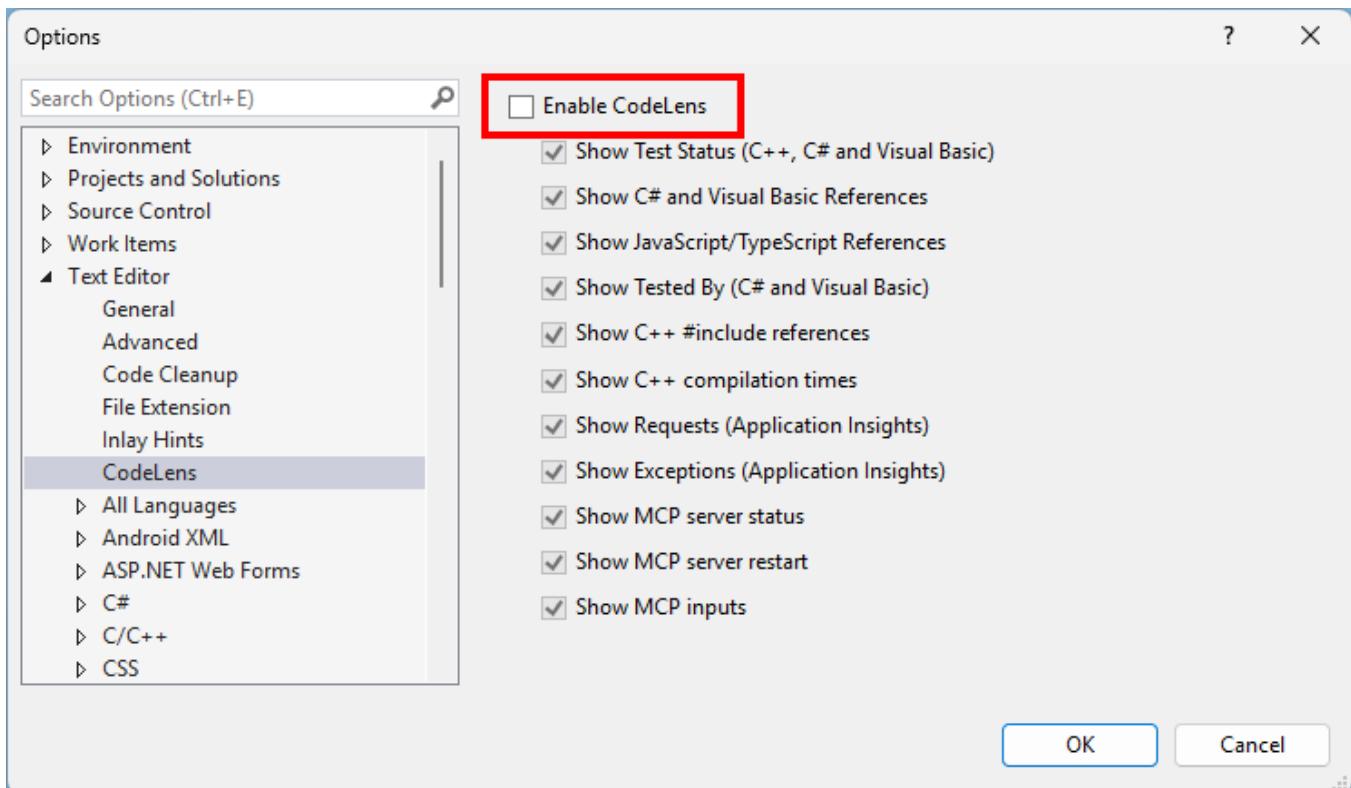
2. Navigeer links naar: **Text Editor** → **CodeLens**

3. Zoek de optie:

Enable CodeLens

Schakel deze optie **uit**

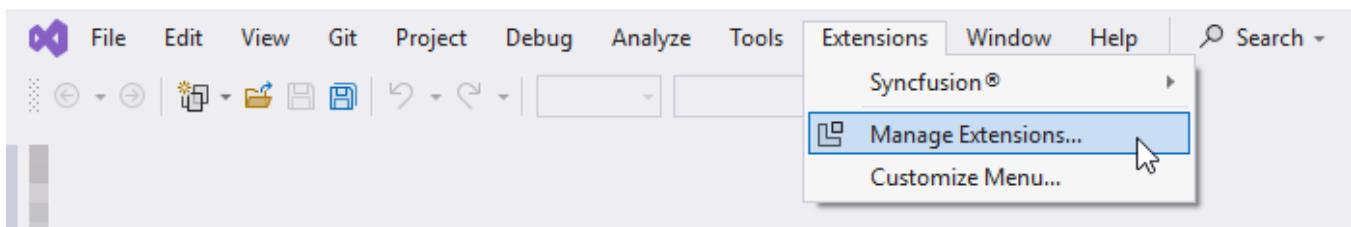
4. Klik op **OK** om te bevestigen



6.4. Extensions

Extensions zijn kleine uitbreidingen die je in Visual Studio kan installeren om extra functionaliteit toe te voegen. Je kan hiermee je ontwikkelomgeving handiger, overzichtelijker of slimmer maken, zonder zelf iets te programmeren.

Je installeert een extension via: **Extensions → Manage Extensions**. Daar vind je een grote bibliotheek van tools die je (meestal) gratis kan toevoegen.



6.4.1. Rainbow Braces

Wanneer code veel `{ }` of `()` bevat, wordt het soms moeilijk om te zien welke haakjes bij elkaar horen. [Rainbow Braces](#) lost dat op door elk haakjespaar een andere kleur te geven.

Voordelen:

- Je ziet sneller waar een codeblok begint en eindigt.
- Je merkt foutieve of ontbrekende haakjes sneller op.
- Het maakt lange methodes en geneste lussen veel overzichtelijker.



Na installatie moet Visual Studio soms herstart worden vooraleer de kleuren zichtbaar

zijn.

6.4.2. IndentRainbow

Veel beginners hebben moeite met het correct inspringen van code. [IndentRainbow](#) geeft elke indentatieniveau een andere kleur, waardoor je meteen ziet hoe diep code genest staat.

Voordelen:

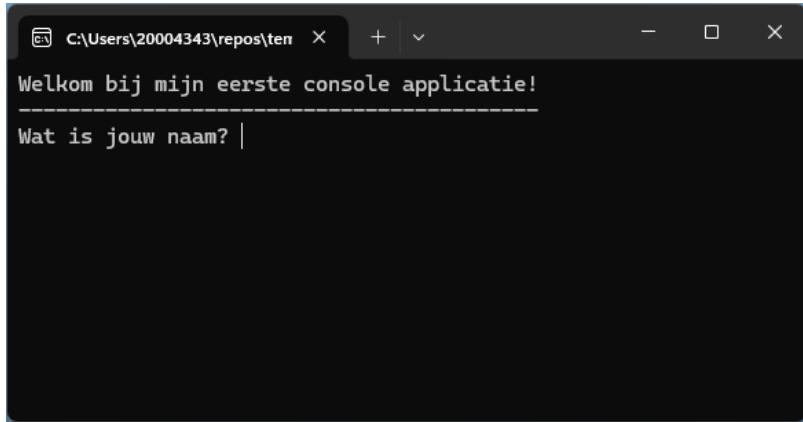
- De structuur van je code wordt visueel veel duidelijker.
- Je ziet sneller wanneer een regel verkeerd is ingesprongen.
- Handig bij langere methodes, lussen en if-statements.

7. Consoleapplicaties

7.1. Wat is een consoleapplicatie

Een consoleapplicatie is een programma dat werkt via tekst in een zwart venster: de console. Je ziet geen knoppen of vensters zoals bij grafische apps. Alles gebeurt met tekst die je typt of die het programma toont.

Voorbeeld van een consolevenster:



A screenshot of a Windows Command Prompt window titled 'C:\Users\20004343\repos\ten'. The window contains the following text:
Welkom bij mijn eerste console applicatie!

Wat is jouw naam? |

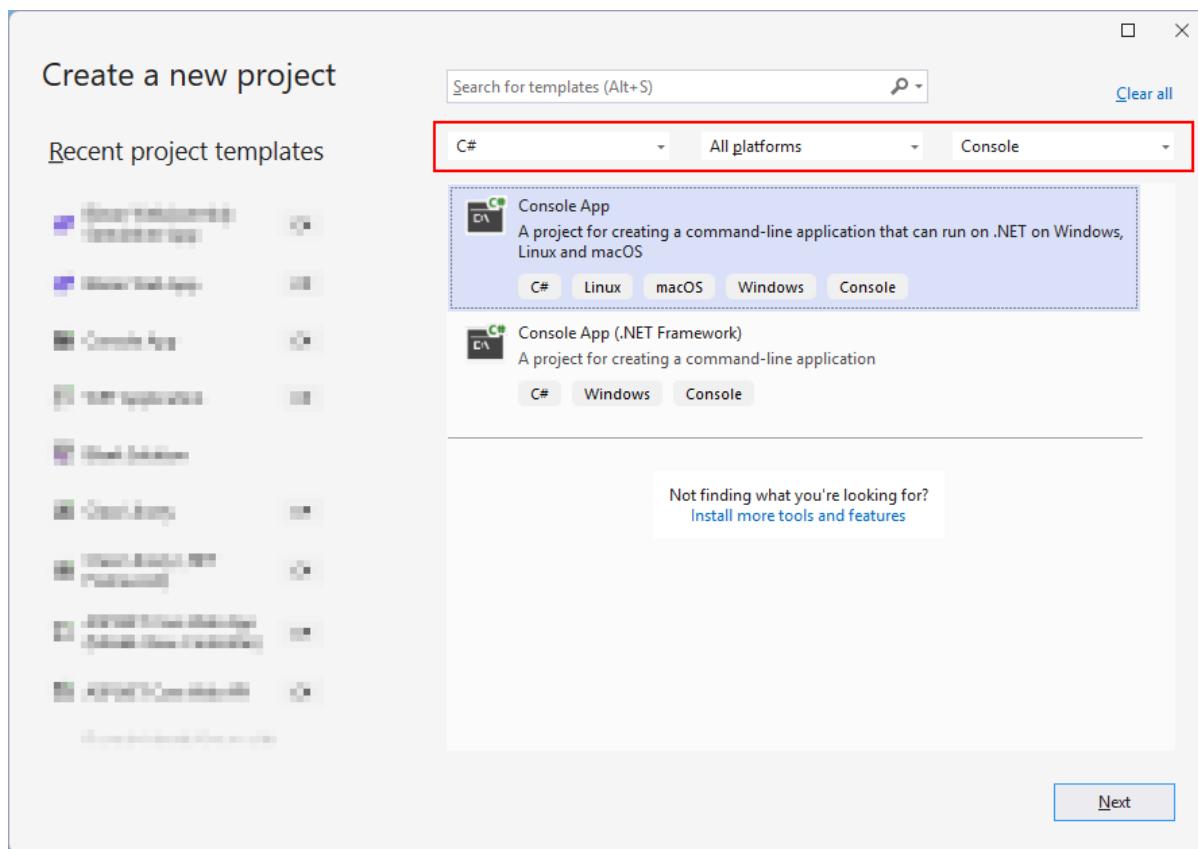
Consoleapplicaties zijn ideaal om de basis van programmeren te leren: variabelen, methodes, logica...

7.2. Een consoleapplicatie maken

Je start met een simpel programma dat iets op het scherm toont:

7.2.1. Project aanmaken

1. Open Visual Studio.
2. Kies **Create a new project**.
3. Selecteer **Console App**.



Je kan de filters bovenaan gebruiken om de programmeertaal (**C#**) en het projecttype (**Console**) te selecteren.



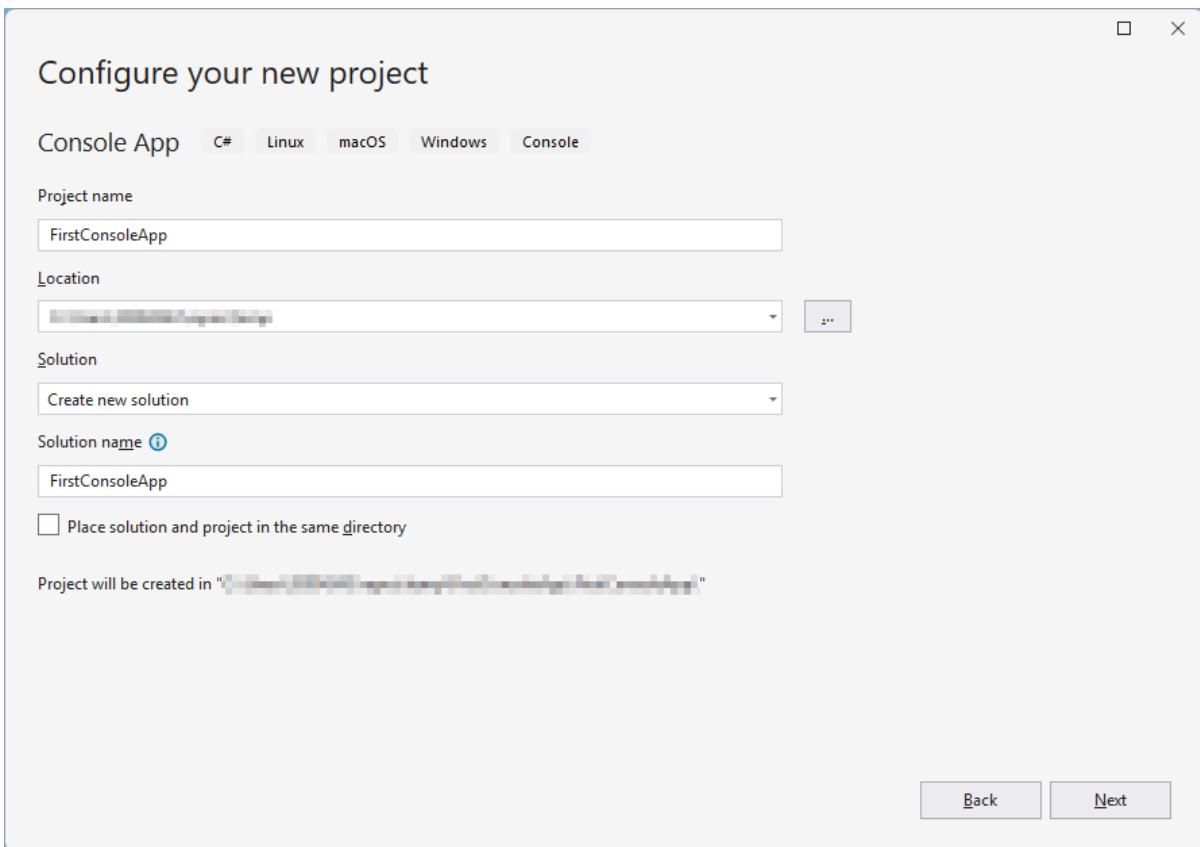
De template **Console App (.NET Framework)** kan gebruikt worden om een applicatie aan te maken op basis van het .NET Framework 4.8. In deze cursus maken we echter enkel gebruik van het nieuwere .NET 8 of later.

4. Geef het project een naam, bv. `FirstConsoleApp`.



Een projectnaam mag:

- **geen speciale tekens** bevatten zoals / ? : & \ * " < > | # %
- **geen spaties** bevatten (hoewel dit technisch wel mogelijk is wordt dit sterk afgeraden)
- geen Unicode-controlekarakters bevatten
- geen surrogate-tekens bevatten
- geen systeem-gereserveerde naam zijn, waaronder: CON, AUX, PRN, COM1 of LPT2
- niet . of .. zijn

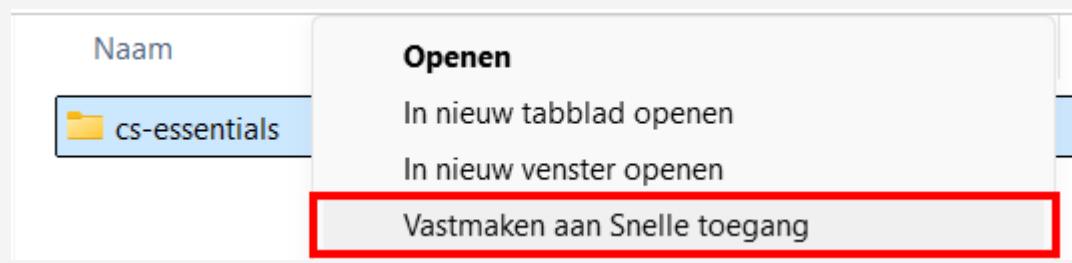


Je kan een project in eender welke folder aanmaken, maar we geven je graag enkele tips:

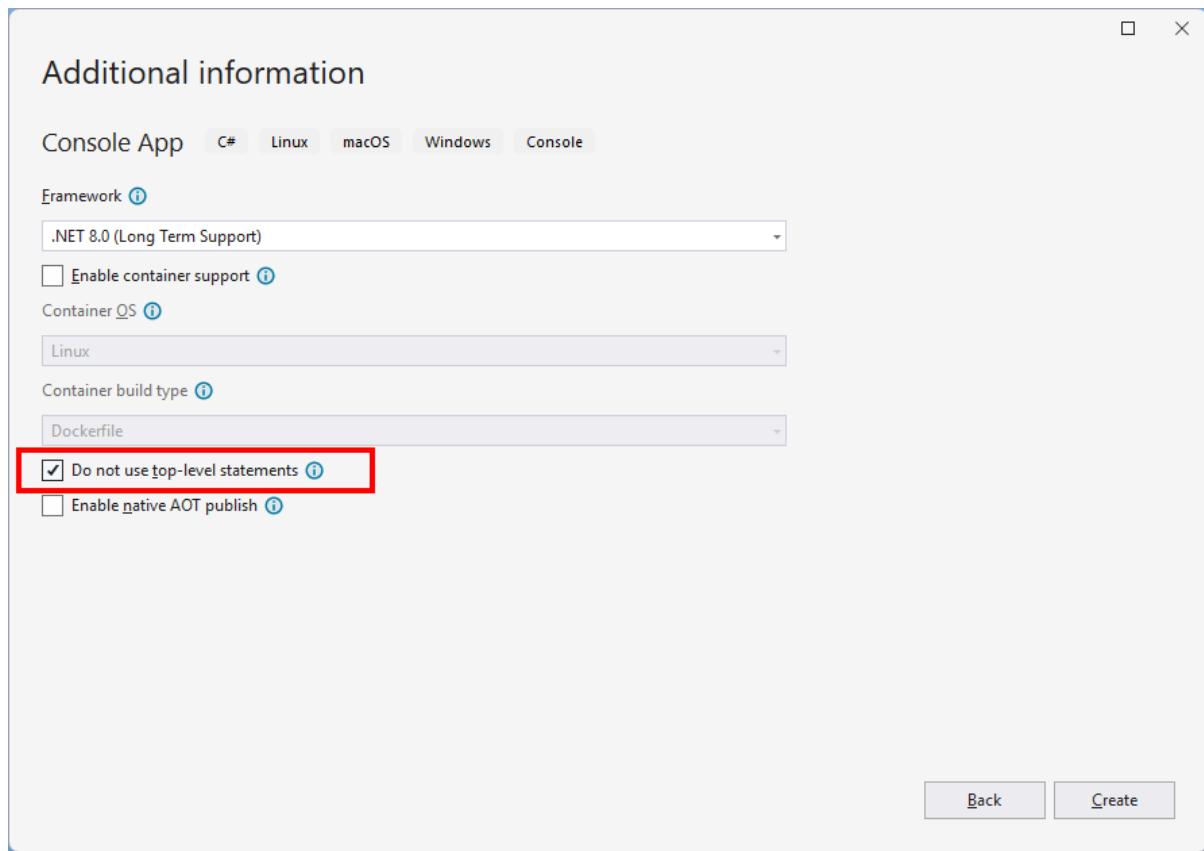
- Maak voor elk olod een aparte folder aan, zo staan alle oefeningen steeds netjes bij elkaar. Eventueel kan je nog subfolders aanmaken per hoofdstuk.
- Zorg dat deze folder niet beheerd wordt door software zoals OneDrive, iCloud, ... Dit zorgt voor veel overbodige download/upload activiteit.
- Maak een snelkoppeling op je bureaublad of pin de folder vast zodat je altijd snel je vorige projecten terug kan vinden.



Een goed voorbeeld om als Location te gebruiken is dus C:\Users\[studentnummer]\source\repos\cs-essentials



5. Selecteer het laatste "Long Term Support"-framework, op dit moment .NET 8.0, en duidt het vinkje aan bij Do not use top-level statements



6. Klik op **Create**.

Je ziet nu een bestand `Program.cs` met de volgende code:

`Program.cs`

```
namespace FirstConsoleApp
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, world!");
        }
    }
}
```

7.2.2. "top-level statements"?

Bij het aanmaken van een nieuw consoleproject zag je een optie: "**Do not use top-level statements**". Maar wat betekent dit nu precies?

Een C# programma bestaat altijd uit **klassen** en **methodes**. De start van elk programma zit in een speciale methode die `Main` heet. Bijvoorbeeld:

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

```
{  
    Console.WriteLine("Hello, world!");  
}  
}
```

Om het eenvoudiger te maken voor beginners, laat C# je ook toe om **zélf** die `Main`-methode niet te schrijven. Je mag dan gewoon dit typen:

```
Console.WriteLine("Hello, world!");
```

C# zal automatisch de rest voor jou aanvullen "achter de schermen". Dit noemen we **top-level statements**.

Waarom gebruiken wij dat NIET?

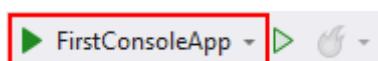
Ook al lijkt dat makkelijker, wij **kiezen ervoor om top-level statements uit te schakelen**. Waarom?

- Je leert beter begrijpen **hoe een echt programma is opgebouwd**.
- Het wordt later makkelijker om methodes te schrijven en klassen te gebruiken.
- Je ziet dezelfde structuur als bij grotere projecten en professionele voorbeelden.

Kortom: het is een klein beetje moeilijker in het begin, maar je zal **snel begrijpen waarom dat een voordeel is**.

7.2.3. Uitvoeren

Druk op **F5** of klik op **Start**. Je ziet het resultaat in een zwart consolevenster.



7.3. Projectstructuur

Wanneer je een nieuw consoleproject aanmaakt in Visual Studio, krijg je automatisch een aantal bestanden en mappen. Hieronder leggen we kort uit wat elk onderdeel betekent.

```
FirstConsoleApp/  
    └── Program.cs  
    └── FirstConsoleApp.csproj  
    └── obj/  
        └── ...
```

7.3.1. Program.cs



Dit is het belangrijkste bestand in een eenvoudige consoleapplicatie. Het bevat de `Main`-methode, oftewel het startpunt van je programma:

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello world!");
    }
}
```

Je kunt later extra methodes toevoegen in dit bestand of aparte klassen in nieuwe bestanden aanmaken.

7.3.2. FirstConsoleApp.csproj

Dit is het projectbestand. Het bevat informatie over je projectinstellingen, doelplatform (.NET 8, .NET 6, ...), en welke dependencies of pakketten gebruikt worden.

Voorbeeld:

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
</PropertyGroup>

</Project>
```

Meestal moet je dit bestand niet handmatig aanpassen. Visual Studio beheert dit automatisch.

7.3.3. obj/ en bin/

Bij het uitvoeren (builden) van je project worden tijdelijke bestanden opgeslagen in de map `obj/`. De map `bin/` bevat het uiteindelijke `.exe`-bestand van je programma.



Je hoeft `obj/` en `bin/` niet op te nemen in Git of bij het delen van je project. Ze worden automatisch opnieuw gegenereerd.

7.3.4. Extra bestanden

Als je je programma uitbreidt, kan je nieuwe `.cs`-bestanden toevoegen met eigen klassen of functies. Deze worden automatisch mee opgenomen in het project.

7.4. Interactie met de gebruiker

In je eerste programma's gebruik je de console om informatie (tekst) te tonen en in te lezen. De console kan je zien als een eenvoudige tekstinterface: het zwarte scherm waarin je programma draait.

Het .NET-framework voorziet enkele handige methodes om je tekst af te drukken op het scherm en invoer van de gebruiker terug inlezen. Deze basisfuncties zijn erg belangrijk: ze vormen de manier waarop je programma 'praat' met de gebruiker en zijn vaak de eerste stap om input en output te begrijpen.

7.4.1. Tekst tonen

Console.WriteLine()

Toont tekst in het consolevenster, de cursor wordt verplaatst naar de volgende lijn.

Program.cs

```
Console.WriteLine("Welkom bij C# Essentials!");
Console.WriteLine("Deze tekst komt ONDER de eerste regel.");
```

The screenshot shows a Microsoft Visual Studio Debug window. The title bar says "Microsoft Visual Studio Debug". The window contains two lines of text: "Welkom bij C# Essentials!" and "Deze tekst komt ONDER de eerste regel.". A red box highlights the first line of text.

Console.Write()

Toon tekst in het consolevenster, de cursor blijft op het einde van de tekst staan.

Program.cs

```
Console.Write("Welkom bij C# Essentials!");
Console.Write("Deze tekst komt NA de vorige tekst.");
```

The screenshot shows a Microsoft Visual Studio Debug window. The title bar says "Microsoft Visual Studio Debug". The window contains two lines of text: "Welkom bij C# Essentials!" and "Deze tekst komt NA de vorige tekst.". The second line starts directly after the end of the first line, indicating the cursor remains at the end of the previous text.

7.4.2. Tekst lezen

Console.ReadLine()

Wacht tot de gebruiker iets typt en op Enter drukt:

```
Console.WriteLine("Wat is jouw naam?");
string name = Console.ReadLine();
Console.WriteLine("Hallo " + name + "!");
```

Console.ReadKey()

Wacht tot de gebruiker eender welke toets intypt:

```
Console.WriteLine("Druk op ESC om verder te gaan...");
Console.ReadKeyInfo info = Console.ReadKey();
if (info.Key == ConsoleKey.Escape)
{
    Console.Write("De Escape toets werd ingedrukt.");
}
```

7.5. Gegevens omzetten

Alle input via `ReadLine()` is **tekst** (`string`). Als je een getal nodig hebt, moet je dit *omzetten*:

```
Console.WriteLine("Hoe oud ben je?");
string input = Console.ReadLine();
int age = int.Parse(input);
Console.WriteLine($"Volgend jaar ben je {age + 1} jaar.");
```

Of korter:

```
int age = int.Parse(Console.ReadLine());
```



Als de gebruiker geen getal intypt, crasht het programma! We leren later hoe we dat opvangen.

7.6. Kleur aanpassen in de console

Je kan de kleuren van tekst en achtergrond in de console aanpassen met de `Console.ForegroundColor` en `Console.BackgroundColor` properties.

```
Console.ForegroundColor = ConsoleColor.Green;
Console.WriteLine("Dit is groene tekst!");

Console.BackgroundColor = ConsoleColor.Yellow;
```

```
Console.WriteLine("Met een gele achtergrond!");
```



De kleur blijft actief tot je ze zelf wijzigt of reset. Wil je alles weer naar standaardwaarden zetten? Gebruik dan:

```
Console.ResetColor();
```



Er zijn veel ingebouwde kleuren beschikbaar, zoals:

- `ConsoleColor.Red`
- `ConsoleColor.Blue`
- `ConsoleColor.White`
- `ConsoleColor.Black`

Je kan de volledige lijst vinden op: <https://learn.microsoft.com/en-us/dotnet/api/system.consolecolor>

7.7. Demo

Maak nu zelf een nieuwe console-applicatie aan en kopieer onderstaande code naar de main-methode. Start de applicatie met de startknop en inspecteer het resultaat!

```
string name;
int number;

//Inlezen
Console.Write("Geef een naam: ");
name = Console.ReadLine();
Console.Write("Geef een getal: ");
number = int.Parse(Console.ReadLine());

//Afdruk
Console.WriteLine("Afdruk");
Console.WriteLine("-----");
Console.WriteLine($"Naam: {name}\t Getal: {number}");
Console.WriteLine("Druk op enter om af te sluiten...");
Console.ReadLine();
```

```
Geef een naam: PXL
Geef een getal: 55
Afdruk
-----
Naam: PXL      Getal: 55
Druk op enter om af te sluiten...
```

7.8. Build Errors

Standaard stelt Visual Studio een verwarrende vraag als er fouten in je code zitten:

"Er zijn buildfouten. Wilt u de laatste succesvolle versie uitvoeren?"

Dat is niet ideaal, je wil liever dat je programma **alleen start als de code foutloos is**. Je kan dit instellen in Visual Studio.

7.8.1. Instelling aanpassen

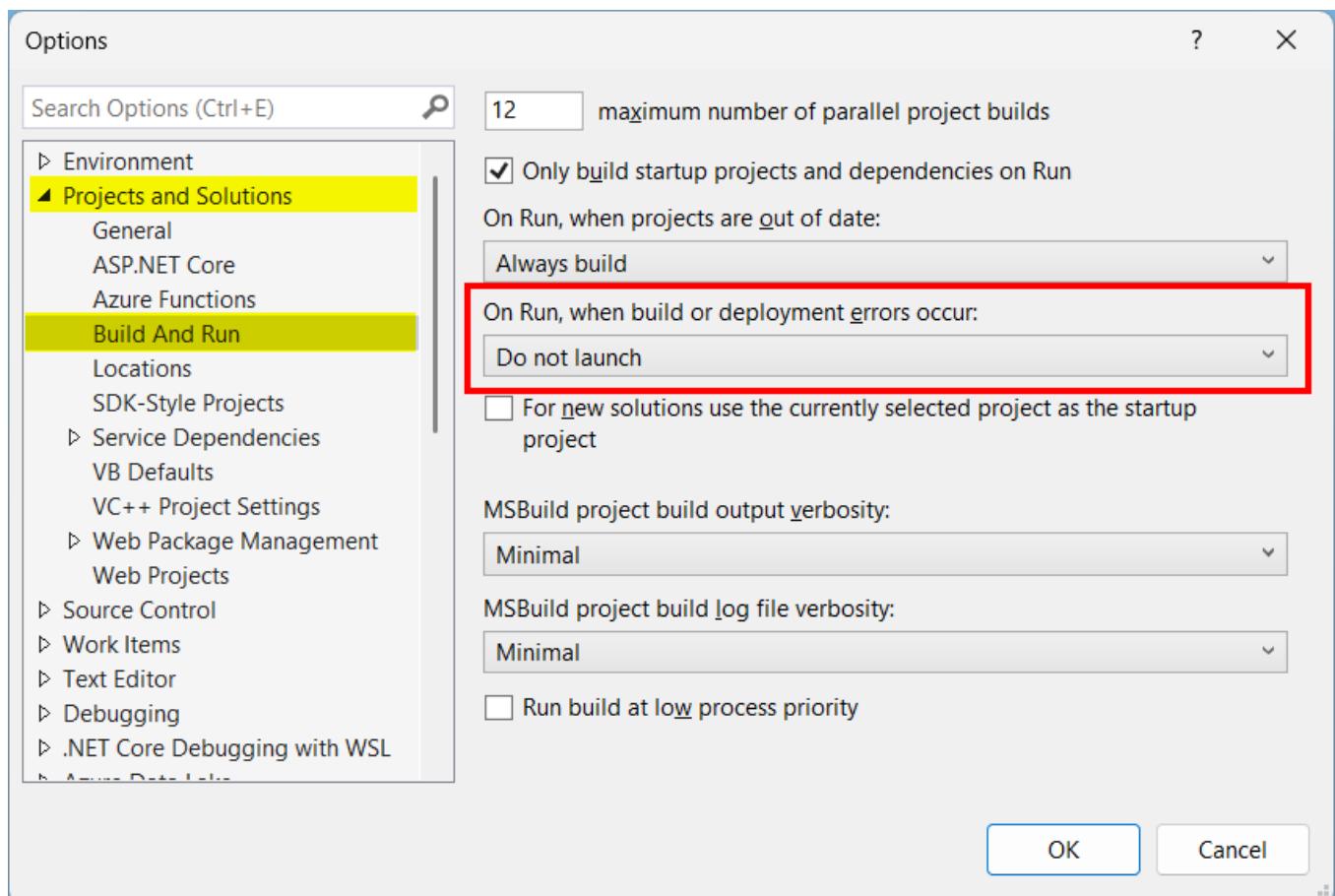
Volg deze stappen:

1. Ga naar het menu: Tools → Options
2. Navigeer links naar: Projects and Solutions → Build and Run
3. Zoek de optie:

On Run, when build or deployment errors occur

en kies: Do not launch

4. Klik op OK om te bevestigen



8. Oefeningen



8.1. Tutorial

[Een .NET-consoletoepassing maken met Visual Studio](#)

8.2. Favoriete kleur

Maak een programma dat:

1. Vraagt naar de naam van de gebruiker.
2. Vraagt naar zijn/haar favoriete kleur.
3. Een zin toont zoals:

```
Wat is je naam?  
Emma  
Wat is je favoriete kleur?  
Blauw  
Leuk om je te leren kennen, Emma! Blauw is een mooie kleur.
```

Oplossing

Program.cs

```
Console.WriteLine("Wat is je naam?");  
string name = Console.ReadLine();  
Console.WriteLine("Wat is je favoriete kleur?");  
string color = Console.ReadLine();  
  
Console.WriteLine($"Leuk om je te leren kennen, {name}! {color} is een mooie kleur.");
```

8.3. Leeftijd

Maak een programma dat:

1. Het huidige jaartal toont.
2. Vraagt naar het geboortejaar.
3. De huidige leeftijd toont.



Gebruik `DateTime.Today.Year` om het huidige jaartal te bekomen

```
Het huidige jaartal is 2025
Wat is je geboortejaar?
1985
Je leeftijd is momenteel 40
```

Oplossing

Program.cs

```
Console.WriteLine($"Het huidige jaartal is {DateTime.Today.Year}");
Console.WriteLine("Wat is je geboortejaar?");
string input = Console.ReadLine();
int birthYear = int.Parse(input);
int age = DateTime.Today.Year - birthYear;

Console.WriteLine($"Je leeftijd is momenteel {age}");
```

9. Source Code Management

9.1. Versiebeheer

Wanneer je code schrijft, maak je voortdurend aanpassingen. Soms werk je iets uit dat niet goed blijkt te zijn of loopt je project ineens fout na een kleine wijziging.

Met versiebeheer kan je eenvoudig teruggaan naar een vorige versie van je code, zonder dat je handmatig telkens een kopie moet bewaren.

Een versiebeheersysteem zorgt er ook voor dat je:

- veilig kan experimenteren;
- fouten snel kan terugdraaien;
- in team kan samenwerken zonder dat je elkaars werk overschrijft;
- wijzigingen netjes kan documenteren.

9.2. Git en GitHub

Git is de tool. GitHub is de plaats waar je jouw code deelt.

Git is een programma dat lokaal op je computer draait. Hiermee beheer je de verschillende versies van je project.

GitHub is een online platform waar je je Git-projecten (repositories) kan bewaren, delen of samenwerken met anderen.

Git	GitHub
Lokaal versiebeheer	Online opslag van Git-projecten
Werkt op je computer	Werkt in je browser
Je maakt snapshots van je project	Je deelt en beheert projecten met anderen

Je hebt dus **beide nodig**: Git om lokaal te werken, GitHub om je werk in te leveren of samen te werken.

9.2.1. Git

Installatie

Om Git te kunnen gebruiken op je computer, moet je het eerst installeren.

Volg deze stappen:

1. Ga naar de downloadpagina van de officiële website: <https://git-scm.com/downloads>
2. Selecteer het besturingssysteem van je computer/laptop (meestal is dit Windows)
3. Klik op de tekst "Click here to download".

The screenshot shows the official Git website at git-scm.com. The top navigation bar includes links for 'About', 'Documentation', 'Downloads' (selected), 'Community', and a search bar. The main content area is titled 'Download for Windows' and features a prominent yellow button labeled 'Click here to download'. Below this, there's a link to 'Other Git for Windows downloads' and a 'Standalone Installer' link.

4. Open het installatiebestand en volg de instructies tot het programma geïnstalleerd is.

Tijdens de installatie worden een aantal geavanceerde opties gevraagd:

- Gebruik notepad als standaard editor
- Gebruik `main` als *default branch*
- Gebruik voor de overige instellingen de standaard waarden

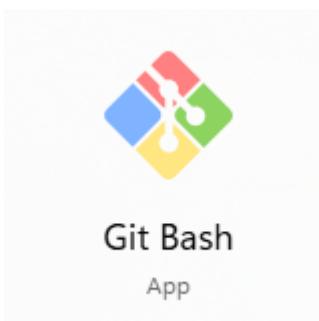
!

The two screenshots show the 'Choosing the default editor used by Git' screen and the 'Adjusting the name of the initial branch in new repositories' screen. The first screen has a dropdown menu set to 'Use Notepad as Git's default editor'. The second screen shows a radio button for 'Override the default branch name for new repositories' selected, with 'main' entered in the input field.

5. Na de installatie kan je Git gebruiken via de terminal (Command Prompt) of via tools zoals Visual Studio.

Configuratie

1. Start het programma *Git Bash* op



2. Gebruik onderstaande commando's om je gebruikersnaam en e-mailadres in te vullen

```
git config --global user.name "Jouw Naam"
```

```
git config --global user.email "voornaam.achternaam@student.pxl.be"
```



Je gebruikt hier vanzelfsprekend je eigen naam en e-mailadres!

```
MINGW64:/c/Users/20004343
20004343@5CD322B2FQ MINGW64 ~
$ git config --global user.name "John Doe"
20004343@5CD322B2FQ MINGW64 ~
$ git config --global user.email "john.doe@student.pxl.be"
20004343@5CD322B2FQ MINGW64 ~
$ |
```

3. Sluit *Git Bash*

4. Vanaf nu worden jouw aanpassingen correct gelinkt aan je naam en e-mailadres.

9.2.2. GitHub

Account

Om je projecten online te kunnen bewaren en delen, heb je een account nodig op GitHub

[github icon] | <https://github.githubassets.com/favicons/favicon.svg>

1. Surf naar <https://github.com/signup>

2. Gebruik:

- Je PXL-mailadres (voornaam.achternaam@student.pxl.be)
- Een sterk wachtwoord **dat je kan onthouden**
- **VoornaamAchternaamPXL** als gebruikersnaam!



Je gebruikersnaam is hetgeen dat zichtbaar is voor je lector wanneer je een opdracht indient. Gebruik dus **zonder uitzondering** je voornaam en achternaam gevuld door "PXL" als gebruikersnaam! Gebruik een hoofdletter voor de eerste letters van zowel je voornaam als je achternaam.

- *Belgium* als regio/land

3. Klik op *Continue* en bevestig je e-mailadres via de mail die je ontvangt.

¶ Je hebt nu een GitHub-account en kan aan de slag! ¶

GitHub Classroom

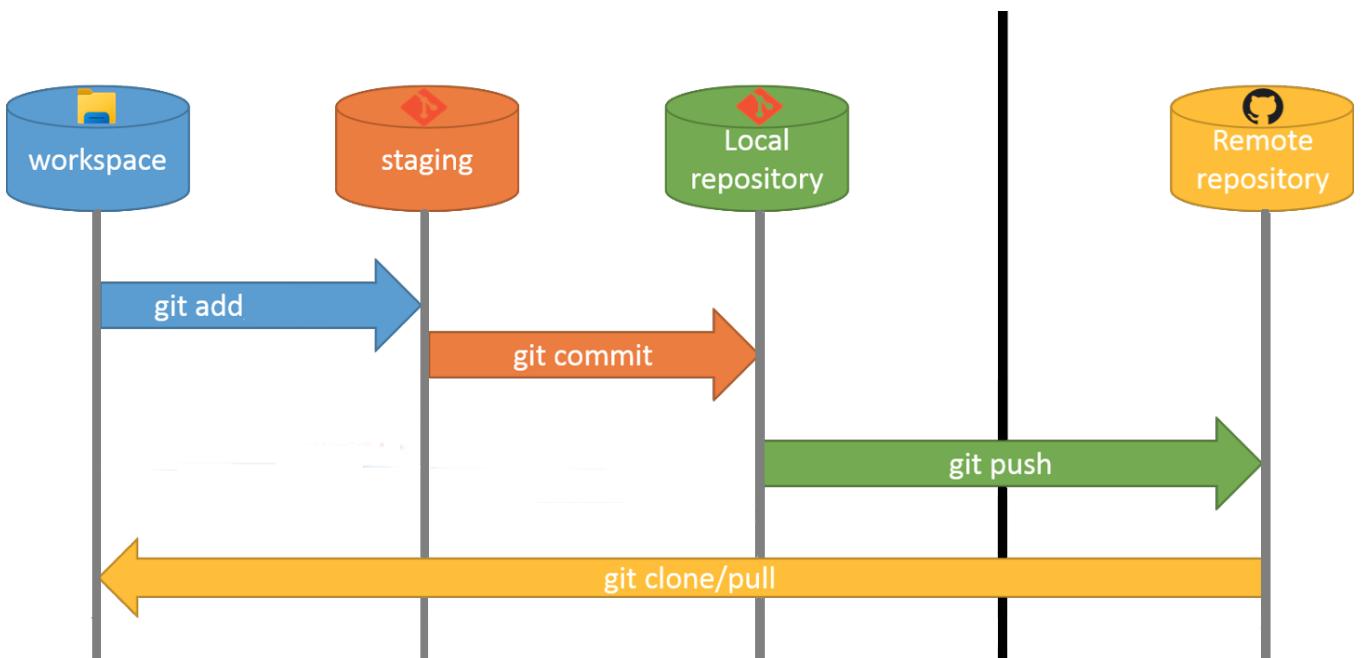
Later in deze cursus maken we gebruik van **GitHub Classroom**. Hiermee krijg je een unieke link om opdrachten te openen en in te dienen.

Wanneer je op zo'n link klikt, gebeurt het volgende:

1. Je wordt gevraagd om in te loggen met je GitHub-account.
2. Je krijgt een persoonlijke repository.
3. Er wordt automatisch een GitHub-project voor jou klaargezet.

Dat project kan je daarna openen en bewerken (=clonen) in Visual Studio.

9.3. Workflow



9.3.1. Vier zone's

Git werkt in vier duidelijk gescheiden zones die elk een specifieke rol hebben in jouw workflow:

1. **Workspace (working directory)** - De map op je computer waar je actief bestanden maakt, wijzigt of verwijdert.
2. **Staging area** - De tussenzone waarin je zelf aanduidt welke wijzigingen je wil opnemen in de volgende commit.
3. **Lokale repository** - Hier worden de commits bewaard. Dit is je volledige projectgeschiedenis op je eigen computer.
4. **Remote repository** - De online versie van je project, meestal op GitHub. Andere gebruikers kunnen hier toegang toe hebben.

9.3.2. Basiscommando's

Clone: project binnenhalen van GitHub

Met **clonen** haal je een bestaande repository van GitHub naar je computer. Je krijgt automatisch de

workspace én een gekoppelde lokale repository.

```
git clone <url-van-je-repo>
```

Na het clonen beschik je over:

- een lokale werkmap met alle bestanden
- een verborgen `.git-map` die je lokale repository bevat
- een koppeling met de remote repository op GitHub

Vanaf nu werk je volledig lokaal tot je wijzigingen terug doorstuurt naar GitHub.

Add: bestanden klaarzetten in de staging area

Wanneer je wijzigingen aanbrengt in je workspace, worden die nog nergens opgeslagen. Met `git add` duid je aan welke wijzigingen je wil voorbereiden voor de volgende commit.

```
git add .
```

of

```
git add <bestandsnaam>
```

De staging area bevat dus alleen de wijzigingen die jij explicet selecteert.

Commit: een momentopname bewaren in de lokale repository

Met een commit maak je een snapshot van alle bestanden **in de staging area**. Daarbij hoort altijd een duidelijke commitboodschap.

```
git commit -m "Loginformulier toegevoegd"
```

- !
- De commit wordt enkel bewaard in je **lokale repository**.
 - Je wijzigingen zijn dus nog niet zichtbaar op GitHub.
 - Je kan meerdere commits na elkaar maken voordat je gaat pushen.

Push: je commits doorsturen naar de remote repository

Met het push-commando stuur je alle lokale commits naar GitHub.

```
git push
```

Daarmee worden je veranderingen definitief gesynchroniseerd tussen je lokale repository en de remote repository.

Pull: de laatste wijzigingen ophalen van GitHub

Wanneer anderen aan hetzelfde project werken, of wanneer je zelf op meerdere computers werkt, moet je soms de nieuwste versies binnenhalen.

Dat doe je met:

```
git pull
```

Dit commando:

- haalt de nieuwste commits van GitHub op
- voegt ze samen met jouw lokale branch

Zo werk je altijd verder op de meest recente versie van het project.

Status: bekijken wat er aangepast is

Het `status`-commando gebruik je voortdurend tijdens het werken. Het toont:

- welke bestanden gewijzigd zijn
- welke bestanden staan in de staging area
- of je commits klaar hebt staan om te pushen
- op welke branch je zit

```
git status
```

Dit commando geeft je een duidelijk overzicht van de huidige toestand van je project.

10. Git in Visual Studio

10.1. Basic Workflow in Visual Studio 2022

10.1.1. Een nieuwe repository maken

In deze video leer je hoe je op een heel eenvoudige manier een nieuwe Git repository kan aanmaken (`init` + `commit`) én deze ook online kan delen via GitHub (`push`). Daarna kan je ook de wijzigingen die remote werden aangebracht makkelijk synchroniseren met je lokale repository (`pull`).

► <https://www.youtube.com/watch?v=oCvb-Q5IXb8> (YouTube video)



Wanneer je code wil delen met je lector, een medestudent of (later) met een collega is dit de **enige juiste** manier om dit te doen. Deel dus nooit screenshots van code, zip-bestanden of gekopieerde code-fragmenten maar werk met een gedeelde GitHub repository!

10.1.2. Een bestaande repository *clonen*

In de praktijk zal je niet vaak een volledig nieuwe repository moeten maken maar werk je vaak verder aan een reeds bestaande repository. Met het `clone`-commando kan je een remote-repository (meestal GitHub) *downloaden*. Daarna kan je met de overige basiscommando's (`commit`, `push` en `pull`) je lokale repository synchroniseren.

► <https://www.youtube.com/watch?v=oCvb-Q5IXb8> (YouTube video)



De volledige video kan je op [YouTube](#) vinden. Let op: het `fetch`-commando en het gebruik van *branches* wordt niet behandeld in deze cursus.

10.2. Tutorial



Wanneer je een evaluatie hebt van dit olod zal je vaak moeten werken in een (gepersonaliseerde) GitHub repository. Daarom is het héél belangrijk dat je vlot een clone kan maken en je wijzigingen kan synchroniseren met het `commit`- en `push`-commando! Het clonen van een repository kan je oefenen met deze tutorial.

10.2.1. Remote repository

Voor deze tutorial maak je gebruik van een reeds bestaande repository:

1. Browse naar <https://github.com/pxl-grpro-cseseentials/tutorial-clone>
2. Klik rechts bovenaan op de groene knop met de tekst "<> Code"
3. Kopieer de web-URL van de repository

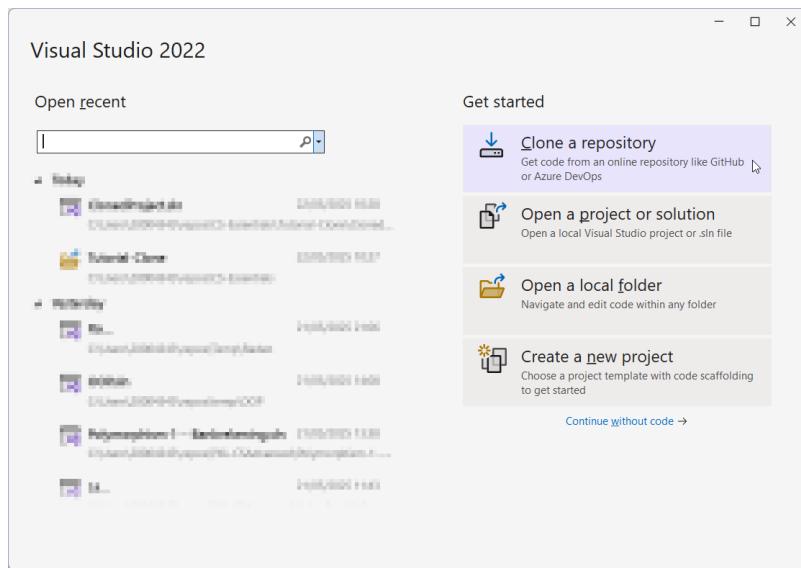
The screenshot shows a GitHub repository named 'Tutorial-Clone'. In the top right corner, there is a green 'Code' button with a dropdown arrow. A red box highlights this button. Below it, a sub-menu has 'Clone' selected. Another red box highlights the 'Copy url to clipboard' button, which contains a hand cursor icon. The URL 'https://github.com/PXL-CSEssentials-Templates/' is shown next to it.



Je hebt enkel rechten om een *clone* te maken van deze repository. Je kan uiteraard wijzigingen lokaal *committen* maar je hebt niet de rechten om deze *commits* ook te *pushen* naar de remote repository. Indien je ook dit wil oefenen kan je er voor kiezen om eerst een fork te maken van bovenstaande repository. Op die manier maak je eerst een "kopie" van de repository naar je eigen Github-account. Daarna kan je de stappen van de tutorial gewoon volgen.

10.2.2. Clone met Visual Studio

1. Open nu Visual Studio en selecteer **Clone a repository**



2. Plak de web-URL van de GitHub repository in het veld *Repository location*
3. Selecteer een lokale map op je computer waar je het project wil bewaren

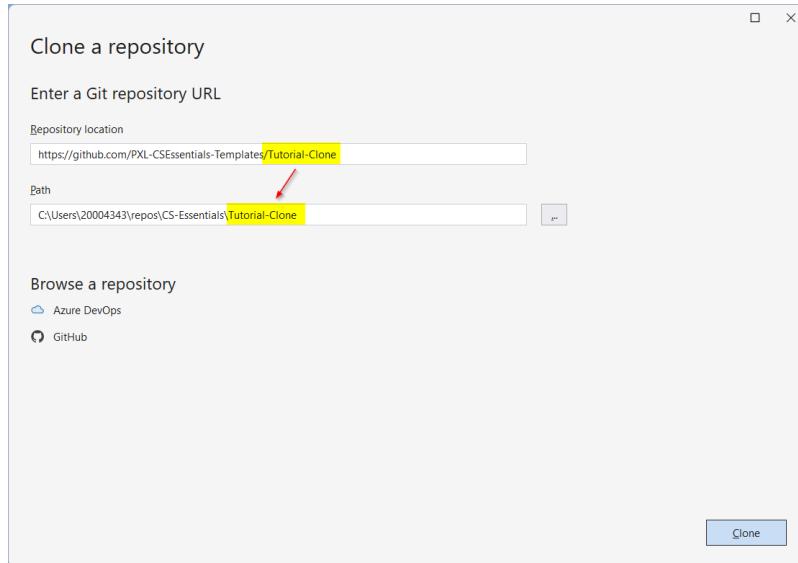


Visual Studio maakt op deze manier **geen extra map aan** voor je repository. Kies dus altijd een pad dat eindigt op de gewenste mapnaam, bijvoorbeeld: `C:\...\repos\CS-Essentials\Tutorial-Clone` waarbij:

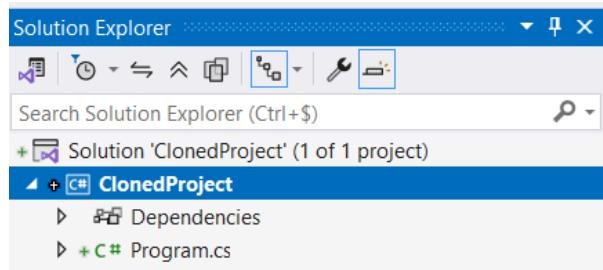
- `CS-Essentials` de "parent"-map is waar je alle projecten van C# Essentials zal bewaren
- `Tutorial-Clone` de "child"-map is waar je **enkel en alleen dit project** zal

bewaren

4. Klik op **Clone**



Visual Studio downloadt nu de volledige projectmap naar je computer en opent automatisch het project.



10.2.3. Kennisclip

► [videos/scm-clone.mp4 \(video\)](#)

Clone een repository met Visual Studio