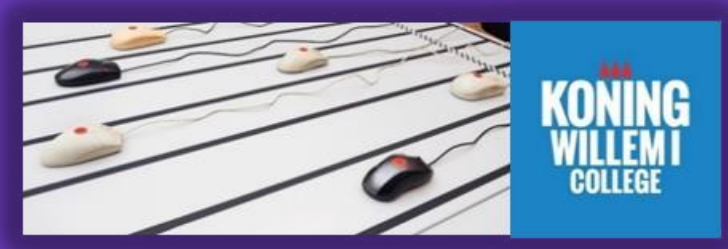
Basis standalone

Realiseren

hoofdstuk

1

C# - Introductie



Algemene informatie

Onderwerp	C# - Introductie
Leerdoel(en)	1. De student kent de geschiedenis van C# 2. De student weet wat een programmeertaal is 3. De student kan Visual studio installeren 4. De student kan een Azure DevOps project aanmaken 5. De student kan een solution aanmaken in VS en deze koppelen met Azure DevOps
Vereiste voorkennis	1. De student heeft kennis van programmeren in Javascript.
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Afgelopen jaar ben je bezig geweest met het maken van websites in HTML/CSS, Javascript en PHP. Tijdens dit thema gaan we verder met het maken van zogeheten "standalone applicaties", oftewel een computerprogramma dat geïnstalleerd en uitgevoerd dient te worden op een losse computer (vandaar de term "standalone"). In deze reader komen een aantal belangrijke termen, die je nodig hebt bij het ontwikkelen van applicaties, kort aan bod.

Daarnaast ga je in dit hoofdstuk kijken naar de ontwikkelomgeving Visual Studio. Deze tool heeft verschillende hulpmiddelen die je kunt gebruiken om straks beter en sneller applicaties mee te ontwikkelen.

Inhoud

Geschiedenis van de C# programmeertaal

Een programmeertaal is simpel gezegd een manier voor de mens om een opdracht aan de computer te geven. Een computer is simpel gezegd dan weer een uit de kluiten gewassen rekenmachine, die niet veel meer kan dan heel erg snel rekenen met de getallen 0 en 1. Daarnaast kan de computer deze 0'en en 1'en ook:

- opslaan op een harde schijf,
- opslaan in zijn geheugen, ook wel RAM (= Random Access Memory, vrij vertaalt 'geheugen met willekeurige toegang') genoemd,
- versturen naar een andere computer (bijvoorbeeld via het internet).

Ook je smartphone is niets meer dan een doodnormale computer.

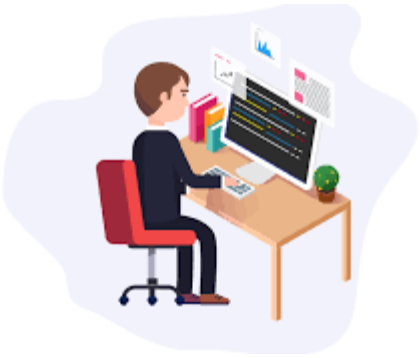
```
ALGOL 60

procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m,
    is transferred to y, and the subscripts of this element to i and k;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

In 1960 werd de programmeertaal Algol gecreëerd. Deze taal werd vooral populair in wiskundige en academische kringen. Via enkele tussenstappen wordt rond 1973 de programmeertaal C gecreëerd, die direct ingezet wordt om het besturingssysteem "UNIX" en later ook "LINUX" te bouwen.

De taal C wordt verder ontwikkeld tot de taal C++ (eigenlijk staat er in programmeertermen C + 1, ofwel een nieuwe, verbeterde versie van C). In 2001 kondigt Microsoft (onder druk van de oprukkende nieuwe open-source taal Java) een nieuwe taal genaamd C# aan. Het hekje (tegenwoordig bekend als Hashtag) staat in de muziekwereld voor zoiets als een halve toon hoger. Hiermee zegt Microsoft dus dat C# een hogere variant is op C++, terwijl de naam C++ dus weer aangeeft dat het een verbetering is op de taal C.

Tegenwoordig is C# dé standaardtaal van Microsoft, waarin niet alleen desktopapplicaties (stand alone), maar ook mobiele apps, websites, artificiële intelligentie en nog veel meer applicaties te bouwen zijn.

Wat is een standalone applicatie?



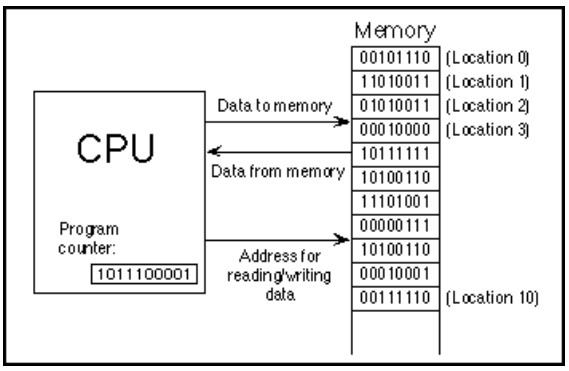
Een applicatie is een verzameling opdrachten die door een computer uitgevoerd kunnen worden. Standalone is een engelse term, die je ook wel kunt vertalen als "losstaand". De term standalone applicatie betekend duszoveel als "*een verzameling opdrachten die op een losstaand systeem uitgevoerd kunnen worden*". In de praktijk bedoelen we hiermee een computerprogramma wat op een computer geïnstalleerd staat.

Wanneer je een applicatie opstart, loopt de computer alle instructies / commando's af (in volgorde / sequentieel) totdat het einde van de lijst is bereikt. Je applicatie stopt als het einde is bereikt of door interactie met een gebruiker. Een gebruiker kan bijvoorbeeld op het kruisje drukken om het venster / de applicatie af te sluiten.

Wat is een programmeertaal?

Zoals gezegd, is een programmeertaal niets meer dan een manier voor een mens om een computer opdrachten te laten uitvoeren. De mens (jij dus) moet hiervoor dus met de computer communiceren. Dit communiceren gaat via een programmeertaal. Een programmeertaal is dus de manier voor een mens om met de computer te praten (vandaar de term "programmeer**taal**").

Het probleem is echter dat je een computer alleen direct kunt aansturen door binaire reeksen (reeksen van 0 en 1) aan de processor door te geven. Dit noemen we machine instructies. Elke computer (= CPU) beschikt over een vooraf ingebouwde set van machine instructies.



Toen de eerste computers gemaakt werden, moesten programmeurs applicaties maken in binaire reeksen. Het was snel duidelijk dat dit te complex was en dat er hierdoor veel fouten ontstonden. Je moet maar eens proberen om een reeks van nullen en enen te onthouden en foutloos in te typen.

```
section     .text
global      _start                              ;must be declared for linker (ld)

_start:                                         ;tell linker entry point

    mov     edx,len                             ;message length
    mov     ecx,msg                             ;message to write
    mov     ebx,1                               ;file descriptor (stdout)
    mov     eax,4                               ;system call number (sys_write)
    int     0x80                                ;call kernel

    mov     eax,1                               ;system call number (sys_exit)
    int     0x80                                ;call kernel

section     .data

msg     db  'Hello, world!',0xa                 ;our dear string
len     equ $ - msg                             ;length of our dear string
```

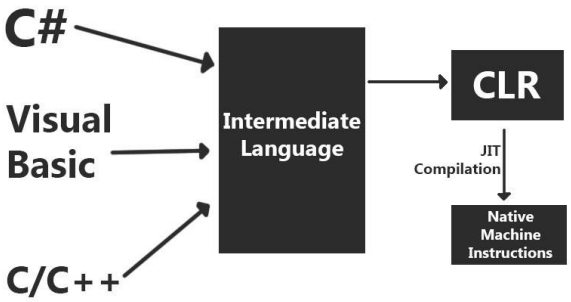
De eerste programmeertaal moest hierin uitkomst gaan bieden. De taal 'Assembly' werd ontwikkeld. Deze taal verving de machine instructies (binair) door begrijpbare instructies. Het probleem van deze taal is nog altijd dat een programma voor een bepaalde CPU werd geschreven. Zat er in de computer een andere CPU, dan werkte het programma niet.

Om dit probleem te ondervangen ging men de zogenoemde hogere programmeertalen ontwikkelen. C# is een van de hogere programmeertalen. Je kunt in deze taal een applicatie maken die op meerdere CPU's gebruikt kan worden. Daar zorgt de compiler voor.

Compiler, CIL en JIT Compiler

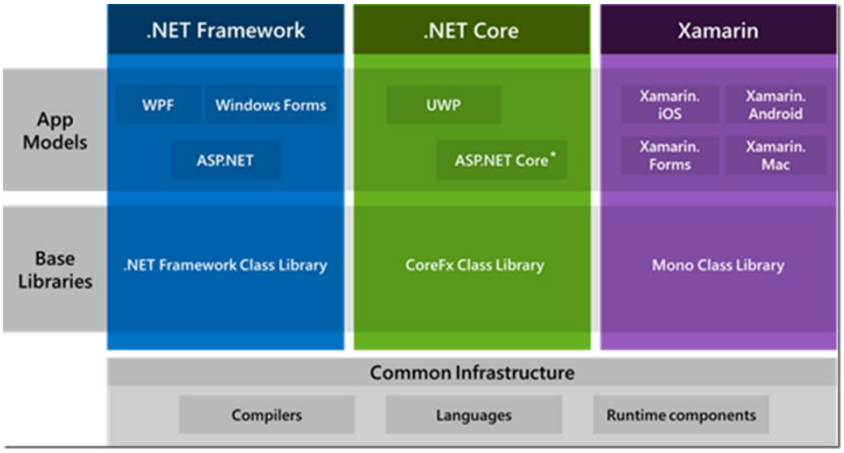
Voor de liefhebbers gaan we nu gaan we even diep op de stof in en duiken in de werking van een C# applicatie. Begrijp je dit niet? Maak je dan niet druk, in de praktijk zul je hier niets mee te maken hebben (de computer doet immers al het werk).

Jouw gemaakte C# code zal niet direct naar executeerbare code (Assembly) omgezet worden, aangezien Assembly erg gevoelig is voor het type processor van je computer en daarnaast ook allerlei belangrijke functionaliteiten mist (Garbage collection, etc). In plaats daarvan wordt deze code omgezet naar zogenaamde Intermediate Language (IL). Een soort tussenstap zeg maar.



Bij het executeren/ uitvoeren van je programma zal de JIT (Just in Time) compiler deze IL code omzetten naar bruikbare machine instructies. Hierdoor kan je geschreven code op verschillende CPU's draaien en kunnen er aan jouw programma allerlei functies toegevoegd worden (Garbage collection, Win32 DLLs, etc) die normaal gezien niet in een Assembly programma zitten.

Frameworks



De programmeertaal C# heeft de beschikking over meerdere zogeheten frameworks. Maar wat houdt een framework nu in? Als je een applicatie gaat maken, dan moet je alles programmeren. Om het programmeurs iets makkelijker te maken, kunnen ze gebruik maken van verschillende 'bibliotheken' ofwel frameworks waar het een en ander al is geprogrammeerd.

Standaard stukjes code, klassen (hierover later dit thema meer) die jij kan (her)gebruiken. Hierdoor hoef je minder zelf te maken en zien applicaties er uniformer uit. Immers, iedereen gebruik dezelfde standaard frameworks en dus dezelfde componenten om applicaties te maken. Vergelijk het met zogenaamde prefab bouwprojecten voor huizen (https://nl.wikipedia.org/wiki/Prefabricage), waarbij hele onderdelen van huizen (bijvoorbeeld hele muren) in een fabriek gemaakt worden. Het bouwen van een huis is dan alleen nog maar de verschillende onderdelen met elkaar verbinden.

Je kan je wellicht voorstellen dat het veel voordelen heeft om standaard elementen te gebruiken (routine in fabriek -> dus sneller en goedkoper, gecontroleerde omgeving waarin het gemaakt wordt -> dus hogere kwaliteit). Dezelfde voordelen zijn er bij standaard elementen die je gebruikt om te programmeren. Je snapt waarschijnlijk waarom veel programmeurs deze standaard frameworks gebruiken.

Binnen C# zijn er veel verschillende frameworks die je kunt gebruiken. Zo is er .Net (spreek uit als dot net) Framework, .Net Core, Xamarin, ADO.net, ASP.net.

Vanaf 2021 wordt door Microsoft het *.Net Framework* niet verder meer ontwikkeld (versie 4.8 is de laatste versie die beschikbaar is en komt) en is *.Net Core* overgegaan naar *.Net*. Ofwel de toevoeging Core is verdwenen. Een van de voordelen van deze bibliotheken is dat deze 'cross-platform' werkt. Je kan dus met dezelfde code een applicatie bouwen die zowel op Windows, als op Linux als op MacOS werkt!

Ontwikkelomgeving

Voor het ontwikkelen in C# gebruiken wij op de opleiding het programma Microsoft Visual Studio. Microsoft Visual Studio is een Integrated Development Environment (IDE) van Microsoft. Een IDE is een programma wat je helpt met het schrijven, compileren en testen van je programmacode. Het biedt een complete set ontwikkelingstools om computerprogramma's in diverse programmeertalen voor met name Windows-omgevingen te ontwikkelen. Visual Studio gebruikt software-ontwikkelingsplatformen van Microsoft zoals Windows API, Windows Forms, Windows Presentation Foundation.

Visual Studio bevat een broncode-editor die zowel IntelliSense (het programma denkt met je mee en geeft je opties om commando's te typen) als refactoren (je code anders formuleren zodat hij bijvoorbeeld beter onderhoudbaar wordt) ondersteunt. Visual Studio beschikt over een geïntegreerde debugger waarmee je jouw programmacode kan doorlopen op zoek naar een programmeerfout. Daarnaast bevat Visual Studio een profiler, vormenontwerper voor het bouwen van GUI-applicaties, webontwerper, klasseontwerper en databaseschemaontwerper. Kortom een enorm uitgebreid gereedschap om mee te kunnen programmeren.

De ICT-Academie heeft voor jullie een enorm uitgebreide licentie (*Visual Studio Enterprise*) weten te bemachtigen, waardoor jij de toegang hebt tot bovenstaande opties en nog veel meer. In het bedrijfsleven is *Visual Studio* dé standaard in .NET/ Microsoft georiënteerde bedrijven en hiervoor die bedrijven dan ook flink de portemonnee trekken om deze licenties te kopen.

📖 Zoek op deze webpagina op, wat een enterprise subscription per maand (in dollars) kost.

Installeren van je werkomgeving

Maak nu **Oefening 1.1**, zodat jouw werkomgeving werkend is voordat je verder gaat.

GIT

Git is een vrij gedistribueerd versiebeheersysteem, ontworpen door Linus Torvalds. Deze persoon ken je wellicht als de bedenker van Linux, een gratis besturingssysteem voor computers. Het is dus een versiebeheersysteem.

GIT lijkt een afkorting van iets, maar is het niet. Het is een naam verzonnen door Torvalds zelf. Git is de benaming van een vervelend persoon in Engelse straattaal, waardoor de maker wel eens gezegd schijnt te hebben dat hij al zijn software naar zichzelf noemt. Eerst was er Linux (een gratis variant van Unix) en toen maakte hij Git. Hij vindt zichzelf dus een vervelend persoon.

Verder geeft men aan dat Git van alles kan betekenen, afhankelijk van je stemming 😄

- Het kan een willekeurige combinatie van 3 letters zijn die je ook nog eens kan uitspreken.
- Het kan een fout uitgesproken Get zijn
- Het kan het Engelse straattaal woord zijn voor een vervelend persoon
- Het kan Global Information Tracker betekenen als je dat leuk vindt
- Het kan Goddamn Idiotic Trunckload of sh*t zijn als het niet wil werken

Kortom, het is een manier om verschillende versies van je code te beheren EN een manier om code in een projectgroep met elkaar te kunnen delen.

Wat doet Git dan precies? Het zorgt ervoor dat er online een kopie van je code aanwezig is, maar daarnaast ook van vorige versies van je code. Het grote voordeel van Git is dat jij op jouw machine kan programmeren en als je daarmee klaar bent jij de gewijzigde bestanden naar Git kunt overzetten. Git zorgt er vervolgens voor dat de vorige versies ook online bewaard worden. Kom je er dus achter dat een aanpassing niet goed is, kun je op die manier terug naar een vorige versie van je applicatie.

Bij Git heb je alleen verbinding nodig met het internet als je wijzigingen wilt doorzetten naar de online bibliotheek. Daarnaast kunnen een aantal programmeurs tegelijkertijd aan (weliswaar verschillende) onderdelen van een applicatie werken zonder dat ze last hebben van elkaar. Ze kunnen onafhankelijk van elkaar hun aanpassingen doorzetten naar de centrale Git opslag.

Over Git is heel veel te vertellen en te lezen. Dat gaat te ver voor deze reader.

Wat we wel willen vertellen hier is dat je op een aantal manier met Git kunt werken. Tijdens deze thema's gebruiken we Git met name in combinatie met Microsoft Azure DevOps. Hieronder leggen we uit hoe je dit kan instellen.

Instellen Microsoft Azure DevOps
Azure DevOps is een verzameling van services die je gratis bij je Visual Studio abonnement geleverd krijgt waarmee het samenwerken aan programmacode geregeld kan worden. Alles wat je nodig hebt om in project verband een softwareprogramma te bouwen kun je hier vinden.

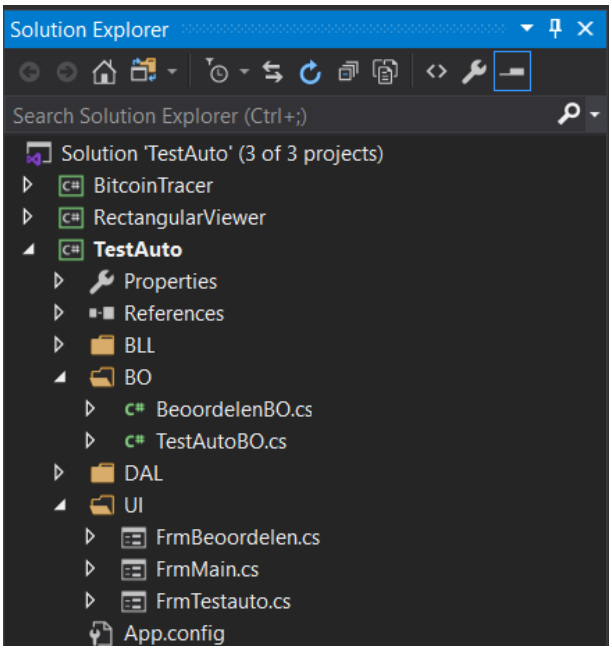
Binnen Azure DevOps kun je onder meer de volgende services gebruiken:

- Git repositories
- Continuous deployment / delivery
- Projectmanagement via een backlogsysteem (Agile / SCRUM)
- Testssuite (geautomatiseerd je software testen)

Solutions en Projecten

Binnen Visual Studio kun je verschillende projecten maken. Ieder project levert een aparte applicatie op. Ieder project zit binnen Visual Studio in een zogenoemde solution. Een solution is een verzameling van projecten die bij elkaar hoort. Dit zouden in de praktijk verschillende applicaties van dezelfde opdracht of voor dezelfde klant kunnen zijn. Binnen dit thema gebruiken we een solution voor alle oefeningen die je gaat maken. We maken dus één solution aan waarbinnen we voor iedere oefening een project zullen maken. Deze solution zal dus al je uitwerkingen van oefeningen bevatten, maar ook meegemaakte instructies die de docent in lessen voordoet.

Voorbeeld:



*Hier zie je de Solution Explorer, waarin de **Solution** genaamd "TestAuto" 3 verschillende projecten bevat. Het **Project** TestAuto is geopend, hiervan zie je de bestanden.*

Voordat je gaat programmeren moet je niet vergeten dat je straks een hele verzameling applicaties krijgt. Om ervoor te zorgen dat al deze applicaties terug te vinden zijn, maken we een speciale folder (map of directory ook wel genoemd) aan waarin we onze projecten gaan bewaren.

Instellen van Azure Devops

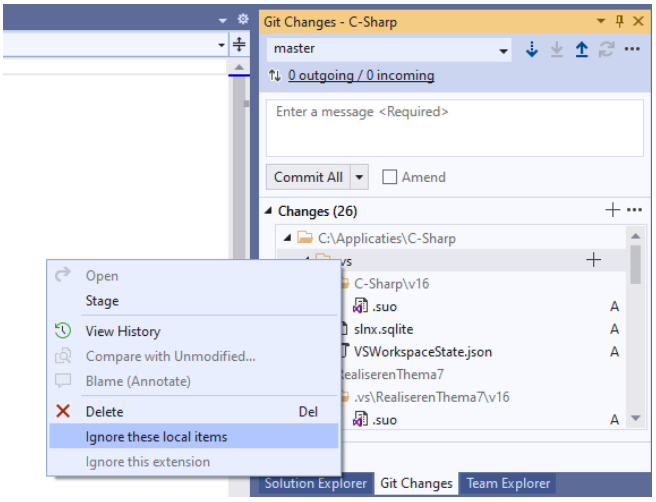
Belangrijk:

Je kunt met onderstaande stappen pas verder, als je Oefening1.1 (installeren werkomgeving) gedaan hebt.

Je kunt nu **Oefening 1.2** gaan maken met behulp van onderstaande informatie.

Je kunt nu **Oefening 1.3** gaan maken met behulp van onderstaande informatie.

Je code in jouw repository plaatsen.



→ Open het tabblad **Git Changes**. Daar zie je iets vergelijkbaars als op de afbeelding hieronder. Dit is een overzicht van de wijzigingen in je lokale map die naar jouw online repositorie gezet moeten worden. Echter je wil niet AL deze wijzigingen kopiëren. Bovenin staat een mapje **.vs**. Dit mapje hoeft niet naar je repository, dus **klik** met de rechtermuis knop op dit mapje en kies voor **Ignore these local items**.

→ Klik vervolgens op het bestand **.suo**, wederom met de rechtermuisknop en kies voor **Ignore this extension**.

→ Klik vervolgens op het mapje **obj**, wederom met de rechtermuisknop en kies voor **Ignore these local items**.

→ Je houdt nu alleen je project en je code over. Vul bovenin een omschrijving van deze backup in (bijvoorbeeld **Eerste commit**).

→ Klik nu op het **pijltje langs Commit All** en kies voor **Commit All and Push**. Je code wordt nu in je online Repository / project geplaatst.

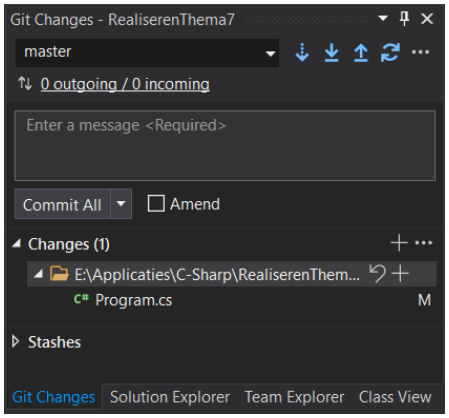
Je kunt dubbelchecken je code inderdaad verstuurd is, door in de Azure DevOps portal naar "Repos" te gaan.

Je bent nu de trotse eigenaar van een Azure DevOps project, waarin we een Git repository gevuld staat met de code van je Solution. Je computer kan nu vredig crashen, je code zal altijd bewaard blijven 😊.

Code commit en push

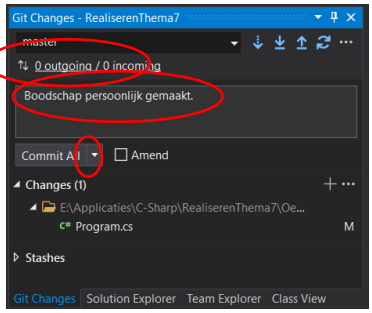
In de vorige stappen heb je een Microsoft Azure DevOps project gemaakt en gekoppeld aan je solution in Visual Studio. Deze repositorie op DevOps gebruik je voor backup, versiebeheer en delen van je code (in dit geval met je docent). Daarvoor moet je natuurlijk weten hoe je wijzigingen die je op je eigen laptop gemaakt hebt kunt uploaden naar je repositorie op Microsoft Azure DevOps. Dit doe je als volgt.

→ Maak een aanpassing in je bestaande console applicatie **Oefening1_2**. Pas de tekst "Hello World!" aan naar "Hello World! <naam> was here!". Waarbij je dan <naam> vervangt door je eigen naam.



→ Klik op het tabblad Git Changes (links langs Solution Explorer) of kies voor View – Git changes in het menu.

→ Schrijf een *commit message* van je wijzigingen voor je collega's en klik op het driehokje achter **Commit All** en selecteer **"Commit all and Sync"**. Het kan zijn dat gevraagd wordt of je jouw wijzigingen in jouw solution wilt opslaan.



→ Check of je de onderstaande melding ziet: