Cursus C#

# Wat is C#?

C# is een programeertaal, net als Python, Java(script), C, C++, … C# is *strongly typed*, wat betekent dat je niet zoals bij Python eender welke soort data in een variabele kan steken. Dit zorgt ervoor dat je minder fouten kan maken, en dat de taal jou meer kan helpen tijdens het schrijven, aangezien hij weet dat je met een getal of een stuk tekst bezig bent.

Het is een programmeertaal die wordt ontwikkeld door Microsoft. Dit betekent echter niet (meer) dat je deze programmeertaal enkel maar kan gebruiken voor programma’s binnen Microsoft Windows. Microsoft heeft echter een hele uitgebreide website waar je over elke optie in C# een gedetailleerde uitleg krijgt, samen met voorbeelden: <https://learn.microsoft.com/en-us/dotnet/csharp>.

# Waar?

C# wordt traditioneel gebruikt binnen het ontwikkelingsplatform Microsoft Visual Studio. Dit is een uitgebreide IDE die toelaat om tal van verschillende soorten toepassingen te ontwikkelen, van mobiele applicaties tot serverside-toepassingen. We gaan in de lessen beginnen met het meest simpele, nl. een *console* programma.

1. Open Microsoft Visual Studio
2. Kies voor ‘File’ > ‘New’ > ‘Project’.
3. Typ bovenaan in de zoekbalk ‘Console’ en kies voor ‘Console App’. Zorg hiervoor dat je niet de .NET-framework versie neemt, deze draait nl. op een verouderde versie van C#.

Afbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

1. Kies voor een projectnaam en een locatie waar je deze wil opslaan. Vink ook de optie ‘Place solution and project in the same directory’ aan en klik op ‘Next’.
2. Hier moet je nu het Framework kiezen, ofwel de versie van .NET waarmee je wil werken. Omdat veel bedrijven op verouderde technologie processen hebben draaien, worden zowel oude als nieuwe versies aangeboden. Er zijn de laatste jaren véél veranderingen op dit vlak geweest, wat uiteindelijk (voorlopig) is geland in .NET 8.0. We kiezen dan ook voor deze versie.

Je hebt hier ook de mogelijkheid om met top-level statements te werken. Je mag dit steeds aanvinken. Klik vervolgens op ‘Finish’.

# Layout

We gaan stelselmatig door de cursus de layout doorlopen maar om te beginnen al een paar belangrijke zaken:

Afbeelding met tekst, software, Computerpictogram, scherm

Automatisch gegenereerde beschrijving

* Code-editor: hierin schrijven we onze code
* Solution Explorer: hierin houden we de bestanden van ons programma bij. Lees zeker hieronder het verschil tussen een *project* en een *solution*.
* Debugger: hierdoor kan je jouw code uitvoeren

Het is belangrijk om het verschil goed te begrijpen tussen een *solution* en een *project*:

* Een *project* is een groepering van verschillende bestanden die samen een geheel uitmaken. Zo’n *project* kan je dan uitvoeren om jouw code te laten lopen.
* Een *solution* is een verzameling van verschillende *projecten*. In het begin zal er vaak maar één project zijn, dus lijkt dit één-op-één hetzelfde maar is het dus niet.

Als je een bestand wil toevoegen, klik je dus steeds met de rechtermuisknop op het project, niet de solution. Hierboven heeft het project ConsoleApp1 een gele kader, de solution waarin dit project zit een bruine kader.

# Datatypes

Voor we kunnen beginnen met programmeren, moeten we de bouwstenen van C# ontdekken. We spreken van primaire datatypes en klassen:

* Primaire datatypes zijn stukjes informatie die ondeelbaar zijn. Denk aan een geheel getal, een kommagetal, een stukje tekst (zit hier wat tussenin), enzovoort. Elke operatie die we gaan doen werken uiteindelijk met primaire datatypes. Hieronder die je een verschil tussen primaire datatypes en klassen.
* Een klasse is een verzameling van verschillende primaire datatypes (en/of andere klassen).

Neem het voorbeeld van een datum. Je zou kunnen zeggen dat een datum een geheel stuk informatie is. Maar bedenk eens wat een datum allemaal bezit: een jaartal, een maand, een dag, etc. Je kan van een dag wel zeggen dat het een werkdag of weekenddag is, maar dan enkel als je de maand en het jaartal weet. Voorts is een dag dus gewoon een nummer. Dus de *klasse* datum bestaat uit de primaire datatypes: *jaartal, maand, dag, uur, minuut, seconde* (allemaal gehele getallen).

Let op: dit is wat kort door de bocht om een goed voorbeeld te vinden van een klassen. Concreet zullen we later zien dat een datum op de achtergrond slechts 1 primair datatype bevat.

Er zijn veel verschillende soorten datatypes. De meeste zijn overeenkomstig maar verschillen omdat ze uit meer bits bestaan, en dus ook grotere waarden kunnen aannemen:

* Gehele getallen: int, byte, small, long. Hier gaan we quasi altijd *int* gebruiken (32-bits).
* Kommagetallen: float, double, decimal. Hier gaan we quasi altijd *float* gebruiken (32-bit). Belangrijk: kommagetallen schrijven we in C# met een punt, niet met een komma.
* Tekst: string, char. Dit is een beetje een uitzondering. Een *char* is één karakter, bijvoorbeeld de hoofdletter A. Een string is een stuk tekst, en dus eigenlijk een aaneenschakeling van verschillende chars. Dus eigenlijk is string geen primair datatype. Echter omdat als we iets van tekst gebruiken dit meestal meer dan 1 letter is, is string een pseudo primair datatype.
* Logica: bool. Een bool (boolean) is een bijzonder datatype. Het bestaat uit slechts 1 bit en kan dus ook maar 2 waarden hebben, nl. 1 of 0. Booleans worden vaak gebruikt om een voorwaarde te stellen, bijvoorbeeld: is een getal groter dan 0, is een tekst leeg, enzovoort. Een boolean is dus een soort van stelling of ja-nee vraag waarop slechts 1 antwoord mogelijk is, nl. waar (1) of niet waar (0). Is het Engels (en dus in C#) spreken we van *true* en *false*.

# Variabelen

In C# (en eender welke programmeertaal) werken we met variabelen. Dit zijn stukjes geheugen waarin we iets kunnen invullen, een primair datatype of een klasse. Belangrijk is bij C#, in tegenstelling tot Python, dat een bestaande variabele niet van type kan veranderen. Het onderstaande gaat dus niet:

Afbeelding met tekst, Lettertype, lijn, schermopname

Automatisch gegenereerde beschrijving

Hier zie je ook al direct hoe we variabelen gaat aanmaken. Je kan ervoor kiezen om direct een type in te stellen voor een variabele, maar eenvoudiger is om Visual Studio dit zelf te laten uitzoeken. We schrijven dus *var* om een variabele te declareren, vervolgens kiezen we een naam (niet beginnend met een getal, geen spaties of -) en stellen al dan niet een initiële waarde in. Als je geen initiële waarde instelt, moet je wel het type op voorhand (dus in plaats van *var*) specifiëren:

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

**We zien hier ook al dat we elke lijn met een ; moeten afsluiten.**

Om iets op het scherm te schrijven, gebruiken we *Console.WriteLine()*.

Afbeelding met tekst, Lettertype, lijn, wit

Automatisch gegenereerde beschrijving

# Wiskundige operatoren

Met variabelen kan je rekenen. De meest eenvoudige operaties zijn optellen (+), aftrekken (-), vermenigvuldigen (\*) en delen (/). Dit kan in combinatie met andere variabelen of direct echte waarden. Zie hieronder een aantal voorbeelden:

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Een belangrijke opmerking bij delen: als je een int (geheel getal) deelt door een andere int, dan krijg je ook een int. Hierbij wordt het gedeelte na de komma weggelaten. Wil je toch een kommagetal als uitkomt, vorm je één van de variabelen om naar een kommagetal:

Afbeelding met tekst, Lettertype, lijn, schermopname

Automatisch gegenereerde beschrijving

## Modulo

Met de modulo kan je de rest na deling bepalen. Dit doe je d.m.v. het %-teken. Zie hieronder een voorbeeld met de rest na deling van 5 door 4. De modulo wordt vaak gebruikt om te kijken of een bepaald getal een deler is van een ander getal, of om te kijken of een getal een even getal is (geen restwaarde na delen door 2).



## Complexe wiskunde operaties

Ook meer complexe wiskundige operaties kan je uitvoeren in C#. Hiervoor moet je echter gebruik maken van functies die reeds voorzien zijn in de Math-bibliotheek. Om functies van een bibliotheek te gebruiken, moet je eerst de naam van de bibliotheek typen, gevolgd door een punt en dan de functie. Hieronder een lijst van de meest belangrijke functies en een paar voorbeelden:

* var macht = Math.Pow(2, 3); // resultaat = 8
* var wortel = Math.Sqrt(9); // resultaat = 3

tip: N-machtwortel kan je verkrijgen door Math.Pow(grondgetal, 1.0/exponent)

* var nwortel = Math.Pow(8, 1.0 / 3); // resultaat = 2
* var sin = Math.Sin(90);
* var cos = Math.Cos(90);
* var tan = Math.Tan(90);
* var invsin = Math.Asin(0.25);
* var invcos = Math.Acos(0.25);
* var invtan = Math.Atan(0.25);
* var absoluut = Math.Abs(-1);
* var afgerondBoven = Math.Ceiling(3.5); // resultaat = 4
* var afgerond = Math.Round(3.5); //resultaat = 4
* var afgerondBeneden = Math.Floor(3.5); // resultaat = 3
* var logBase\_e = Math.Log(10);
* var log10 = Math.Log(8);
* var pi = Math.PI;
* var tau = Math.Tau;
* var e = Math.E;

# Collecties

## Arrays

Een variabele kan ook een array zijn. Een array is een lijst van verschillende waarden, die van hetzelfde datatype zijn. Een array gebruikt vierkante haakjes [ ] wordt als volgt opgesteld:

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

Hier maak je een lijst aan van 3 plaatsen groot (new int [3]) en vul je de 3 plaatsen op. Let op: arrays zijn *zero-based*, dat wil zeggen dat ze vanaf 0 beginnen te tellen. *nummers[0]* wijst dus naar de eerste plaats in de lijst.

## List

Een array heeft een specifieke grootte, waar je niet boven kan gaan na initialisatie. Wil je toch een collectie de dynamisch groter en kleiner wordt, gebruik dan List<>. Hieronder vind je een voorbeeld van een lijst met getallen. Er worden 3 getallen (nl. 1, 2, 3) toegevoegd en vervolgens het eerste getal (1) getoond op het scherm en dan het tweede getal (2) verwijderd.

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

## For each

Om een collectie te overlopen, kan je een for each loop gebruiken. Deze gaat een bepaald stuk code voor elk element in de collectie uitvoeren. Zie onderstaand voorbeeld waarbij voor elk getal in de collectie de deling door 2 wordt getoond op het scherm. Dat i die hier wordt gebruikt is een placeholder voor de huidige waarde die wordt overlopen, en is wat betreft de naam vrij te kiezen.

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

## For

Als je in plaats daarvan de indexen (plaatsen in de collectie) wil overlopen, kan je een for-lus gebruiken. Onderstaand voorbeeld overloopt de indexen van een lijst en toont het resultaat van de deling door 2 per getal. Ook hier is de naam van de placeholder voor de index vrij te kiezen.

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

## Basisoefeningen

1. Maak een lijst en steek hier 5 getallen in. Overloop deze lijst en toon de getallen op het scherm.
2. Maak een lijst en steek hier 3 getallen in. Overloop deze lijst en tel alle getallen bij elkaar op.
3. Maak een lijst en steek hier 5 strings (tekst) in (bv. voornamen of namen van/voor huisdieren). Overloop deze lijst en toon deze strings op het scherm.
4. Maak een lijst van getallen. Laat automatisch hier alle getallen van 1 t.e.m. 100 in steken. Overloop vervolgens deze lijst en toon de getallen op het scherm om dit te testen.

# Structuren

Normaal gezien loopt een programma van boven naar beneden, lijn per lijn. Echter, soms ga je bepaalde lijnen niet willen uitvoeren onder een bepaalde voorwaarde, of bepaalde lijnen herhalen.

## Logische operatoren

Bij programmeren maken we vaak gebruikt van booleans (zie primaire datatypes). Deze kunnen ook worden berekend door een stelling op te stellen en het resultaat hiervan op te slaan. Hieronder zie je een voorbeeld van een getal waarvan wordt bepaald of het groter is dan 0 en dit resultaat wordt getoond op het scherm.

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

Analoog heb je ook <, <=, >=, >. Je kan ook kijken of 2 waarden wel of niet aan elkaar gelijk zijn. Dit doe je met de == (gelijk) of != (niet gelijk).

Afbeelding met tekst, Lettertype, schermopname, wit

Automatisch gegenereerde beschrijving

## Voorwaardelijke uitvoer

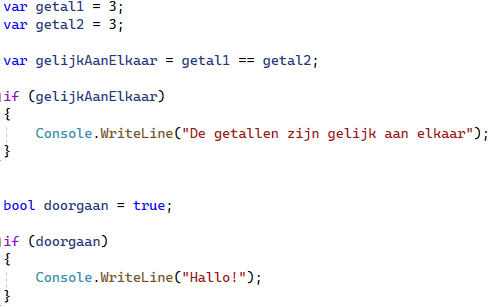
### If

Met de if-structuur kan je code enkel uitvoeren als die aan een bepaalde voorwaarde voldoet. In onderstaand voorbeeld wordt het getal enkel op het scherm getoond als deze groter is dan 0.

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

Let op dat de voorwaarde die je bij een if schrijft, eigenlijk een boolean is. Je kan dus ook het volgende doen:



## If-else

De if-else-structuur is een uitbreiding van de if-structuur die ook code uitvoert als er aan een bepaalde voorwaarde NIET wordt voldaan.

Afbeelding met tekst, schermopname, Lettertype, lijn

Automatisch gegenereerde beschrijving

### If-else if-else

Met de if-else if-else structuur kan je meerdere voorwaarden nagaan alvorens uiteindelijk ,als alle voorwaarden niet waar waren, in de else terecht te komen.

Afbeelding met tekst, schermopname, Lettertype, nummer

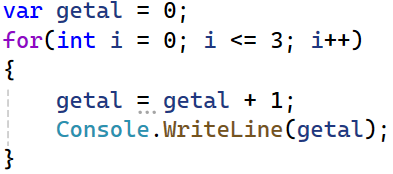
Automatisch gegenereerde beschrijving

## Herhaaldelijke uitvoer

Je kan ervoor kiezen om code meermaals uit te voeren. Dit kan ofwel een vast aantal keer (begrensd) of tot een bepaalde voorwaarde wordt bereikt (onbegrensd).

### Begrensd (vast aantal keer)

Dit lijkt er op de for-lus voor het overlopen van de indexen van een lijst, waarbij we een aantal getallen overlopen tot een eindgetal. Onderstaande lus wordt bijvoorbeeld 3 keer uitgevoerd.



### Onbegrensd (variabel aantal keer)

Hier ga je na iedere iteratie kijken of aan een bepaalde voorwaarde is voldaan. Deze code zal een getal telkens met 1 verhogen tot het gelijk is aan 0.

Afbeelding met tekst, Lettertype, schermopname, lijn

Automatisch gegenereerde beschrijving

# Oefeningen

*IF / IF – ELSE / IF ELSE-IF ELSE*

Voorbeeld:

int getal = 5;

if (getal > 0)

{

Console.WriteLine($"Getal {getal} is groter dan 0");

}

else if(getal < 0)

{

Console.WriteLine($"Getal {getal} is kleiner dan 0");

}

else

{

Console.WriteLine($"Getal {getal} is gelijk aan 0");

}

Oefeningen:

1. Bepaal voor 2 getallen (variabelen) welk getal het grootste is en toon dit op het scherm.
2. Neem 2 variabelen, genaamd ‘teller’ en ‘noemer’, en bereken het quotiënt van deze 2 getallen een getal is groter dan 10.
3. Neem 2 variabelen, genaamd ‘getal’ en ‘deler’, en kijk of ‘deler’ een gehele deler is van ‘getal’. (een gehele deler is een getal dat waarbij de restwaarde na de deling door dit getal gelijk is aan 0, ofwel er is geen restwaarde).

*LIJST*

Voorbeeld:

List<int> getallen = new List<int>();

getallen.Add(1);

getallen.Add(2);

getallen.Add(3);

getallen.Add(4);

foreach(int getal in getallen)

{

Console.WriteLine(getal);

}

Oefeningen:

1. Maak een lijst van 5 willekeurige getallen en bepaal welke getallen even zijn.
2. Maak een lijst van 10 getallen en bepaal het grootste en kleinste getal.
3. Maak een lijst van 8 getallen en tel alle getallen bij elkaar op.

*BEGRENSDE HERHALING*

Voorbeeld:

for(int teller = 1; teller <= 10; teller++)

{

Console.WriteLine(teller);

}

Oefeningen:

1. Toon alle gehele getallen tussen -5 en +5.
2. Bepaal voor alle getallen van 0 t.e.m. 10 of dit een even getal is. (tip: de restwaarde van een getal gedeeld door 2 is bij een even getal steeds 0)
3. Bepaal of 371 een priemgetal is.

*ONBEGRENSDE HERHALING*

Voorbeeld:

int x = -8;

do

{

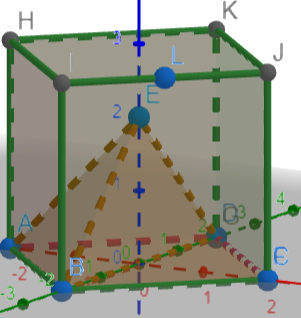
x = x + 1;

} while (x < 0);

Oefeningen:

1. Toon de kleinste deler van 839665 (behalve 1).
2. Bepaal de grootste faculteit die kleiner is dan 1.000.000.
3. Toon welke getallen van 1 t.e.m. 1000 een priemgetal zijn.

*ALGEMENE OEFENINGEN*

1. Los de volgende opdracht stap-voor-stap op:
   1. Toon alle gehele getallen tussen 1 en 40
   2. Bepaal per geheel getal (tussen 1 en 40, alsook hierna) de waarde van de deling door 2.
   3. Bepaal per geheel getal de rest na deling door 2 (modulo).
   4. Toon op het scherm alle positieve, gehele delers van 40.
2. Toon de wet van Pythagoras aan met de 3-4-5 driehoek.
3. Bewijs de wet van Pythagoras.
4. Bewijs de grondformula van de goniometrie: ( voor een zelfgekozen waarde x.
5. Bereken het restvolume van de balk (2 x 2 x 3) na het inbrengen van een vierkante piramide (2 x 2 x 2).
6. Bereken het GGV en KGD van 10 836 en 8 736.
7. Ontbind de veelterm m.b.v. de discriminant.
8. Ontbind de veelterm met de methode van Horner.

# Functies

Binnen C# kan je een klasse maken met functies. Functies zijn dat stukjes code die je vanaf elders kan oproepen en dit verschillende keren.

Aan deze functies moet je soms ook extra informatie meegeven. Denk bijvoorbeeld aan de functie *Math.Pow(2,3).* Hier moeten een grondtal en exponent aan worden meegegeven worden alvorens de functie kan werken. Dit noemen we **parameters**. Voor elke parameter geef je het datatype op en naam die je doorheen de functie hiervoor zal gebruiken.

Ook heb je 2 soorten functies: functies die data teruggeven en functies die enkel zaken uitvoeren. Denk bijvoorbeeld aan Console.WriteLine(). Deze functie schrijft wel informatie op het scherm, maar deze functie geeft geen berekend resultaat terug dat je kan opslaan in een variabele. *Math.Pow(x,y)* daarintegen geeft wel een waarde terug, nl. de macht van xy. Bij elke functie moet je het **return type** opgeven. Dit is het soort informatie die wordt teruggegeven. Indien jouw functie niets teruggeeft, schrijf je hier *void*. Voor in een functie effectief iets terug te geven, gebruik je return … en hierna dan de waarde of variabele die je wil teruggeven.

Een functie heeft de volgende hoofding:

public return\_type NaamFunctie(datatype\_parameter1, naam\_parameter1, …)

Een paar voorbeelden:

public int som(int getal1, int getal2){

return getal1 + getal2;

}

public double product(double a, int b){

return a \* b;

}

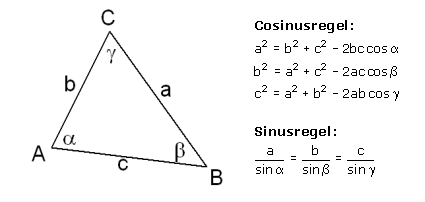
public void toonSom(int a, int b){

Console.WriteLine(som(a,b));

}

Oefeningen:

1. Maak een klasse Rekenmachine en maak hierin de volgende functies. Een functie die:
   1. de som van 2 decimale getallen berekend.
   2. het verschil van 2 decimale getallen berekend.
   3. het product van 2 decimale getallen berekend.
   4. het quotiënt van 2 decimale getallen berekend.
   5. de nde-macht van een decimaal getal berekend.
   6. de nde-machtswortel van een decimaal getal berekend.



1. Maak een klasse Driehoek geef deze 6 eigenschappen (zijden a,b,c en hoeken alfa, beta, gamma).

Begin met 2 functies aan te maken, nl. rad\_to\_deg(x) die x aantal radialen in graden omvormt en deg\_to\_rad(x) die x aantal graden in radialen omvormt.

Maak vervolgens functies aan voor de 3 cosinusregels (bereken\_a, bereken\_b,bereken\_c) zodat je de drie zijden kan berekenen.

Maak ook een functie berekenSinus aan die tekst als parameter krijgt, nl. de naam van de waarde die je wil berekenen, bv. “a” of “beta”. Afhankelijk van de waarde van de parameter (if else) ga je na welke waarde je moet berekenen en of dit mogelijk is (zijden/hoeken > 0). Indien dit mogelijk is, geef je de waarde in. Indien dit niet mogelijk is, geef je 0 terug.

Maak als laatste nog een functie vervolledig() die alle eigenschappen van de driehoek probeert in de vullen. Als je bijvoorbeeld zijde a, zijde b en hoek gamma hebt ingegeven, worden de andere eigenschappen van de driehoek ingevuld met hun berekende waarden.