

C# Les 1

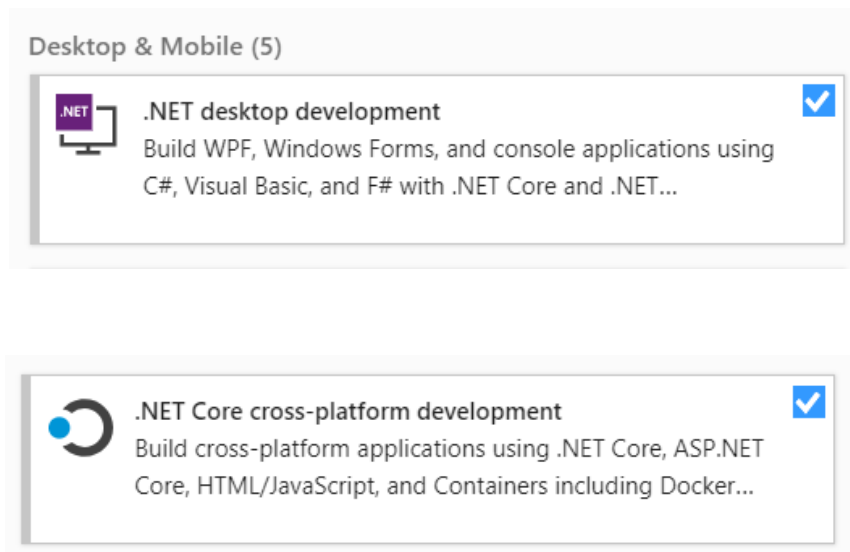
Visual Studio

We gaan Visual Studio gebruiken om een eerste C#-programma te schrijven. Dit programma biedt onder andere een handige omgeving om later grafische user interfaces (GUI) te gaan maken. Visual Studio komt in verschillende versies, we gebruiken de gratis "Community" versie.



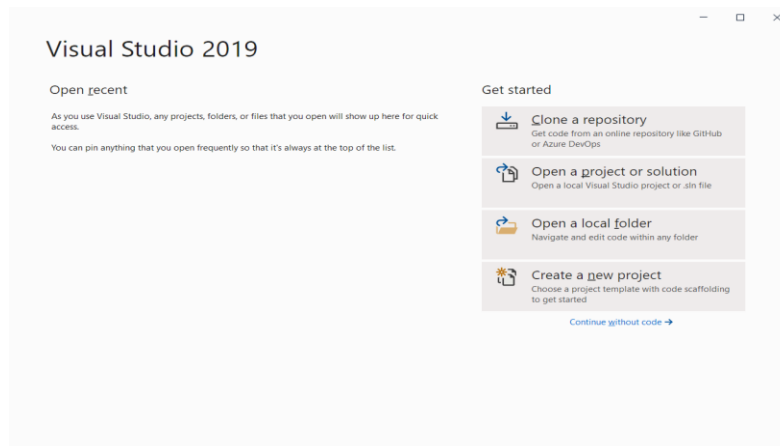
Visual Studio installeren

Je kan Visual Studio Community edition downloaden van <https://visualstudio.microsoft.com/>. Als je de Visual Studio installer start krijg je de keuze tussen allerlei opties. Wat je zeker moet aanvinken zijn deze 2 blokken:

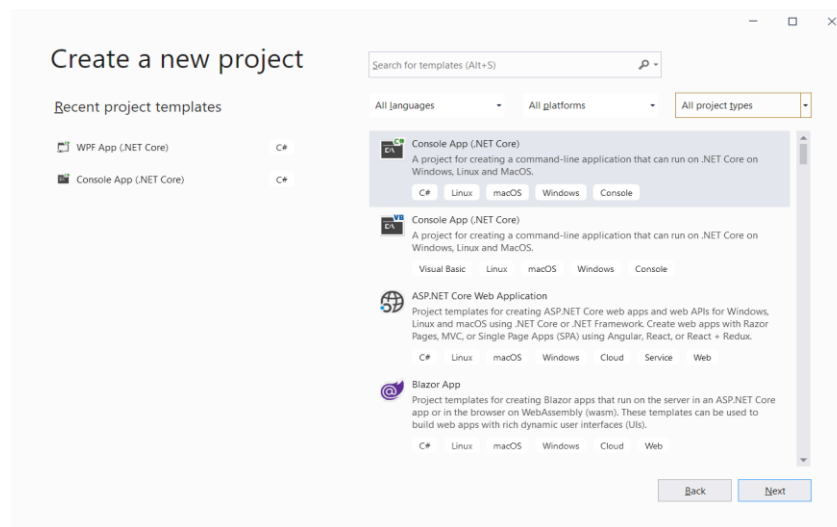


Een eerste programma

Laten we een eerste C#-programma schrijven. We starten hiervoor Visual Studio. Als je het de eerste keer opent zie je onderstaand scherm. Klik op "Create a new project" om een nieuw project te maken:



We gaan eerst een console programma maken, een programma dat tekst toont. Zoek hiervoor in de lijst de optie "Console App (.NET Core)", selecteer dit en klik op "Next". (Als je deze optie niet ziet start dan de Visual Studio Installer terug op en zorg dat de twee opties die hierboven vermeld zijn geïnstalleerd zijn (.NET desktop development en .NET Core))



Geef het project een naam en klik op "Create":

Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name

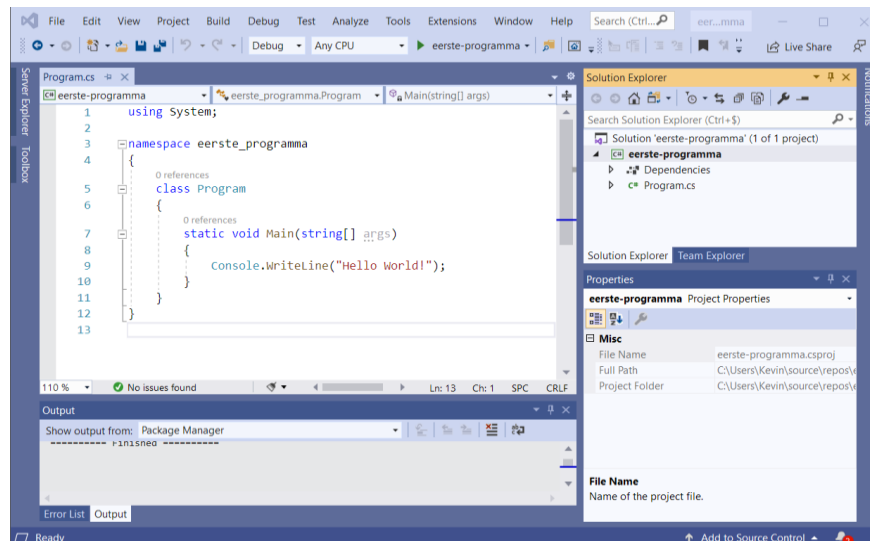
Location
 ...

Solution

Solution name ⓘ

☐ Place solution and project in the same directory

Visual Studio opent nu het bestand "Program.cs" in je nieuwe project.



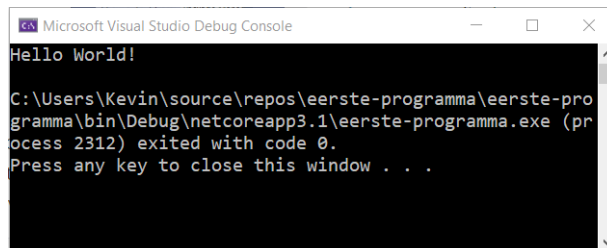
Ons project bevat één programma ("Program.cs"), maar kan eventueel later nog andere bestanden gaan bevatten. In het geopende programma zie je verschillende accolades staan. De *accoladen* staan steeds met twee (open en sluit) en vormen een soort code-blok. (Visual Studio toont stippellijntjes tussen 2 bij elkaar horende accolades, handig!). Het woordje dat net boven de open-accolade staat zegt iets over het code-blok. In het geopende programma zit dus een namespace-blok, een class-blok en een Main-blok (met static void voor).

Elk uitvoerbaar project heeft in de bestanden ergens een *Main methode* staan. Dit is het blok dat uitgevoerd zal worden als we het programma uitvoeren:

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

Laat je niet afschrikken door de "static void" en "string[] args", we bespreken later wat dit wil zeggen.

De Main methode staat dus in een *class* "Program" en in een *namespace* "eerste_programma", dit gaan we voorlopig ook even negeren, hierover later meer. Waar je je nu moet op concentreren is de code binnen het Main-blok, dit is de code die gaat worden uitgevoerd. Om het programma zoals het er nu staat uit te voeren kan je *ctrl-F5* drukken of op het groene play pijltje klikken in de balk bovenaan. Visual Studio opent een command line en toont de uitvoer:



```
Microsoft Visual Studio Debug Console
Hello World!

C:\Users\Kevin\source\repos\eerste-programma\eerste-programma\bin\Debug\netcoreapp3.1\eerste-programma.exe (process 2312) exited with code 0.
Press any key to close this window . . .
```

Je eerste C#-programma is een feit!



Merk op:

- Console.WriteLine(...) is dus blijkbaar een instructie die iets in de commandline toont.
- Goed om weten is dat elke regel in C# moet worden afgesloten met een punt-komma.
- Als je tekst typt in Visual Studio merk je dat het programma je helpt om de code netjes te zetten. Fouten, zoals het vergeten van komma's, worden ook meteen aangegeven. Handig!

Imperatief programmeren?

C# (C sharp) is een *object-georiënteerde imperatieve* taal. Zoals bij vele talen is de *syntax* (= schrijfwijze) geïnspireerd op de imperatieve taal C.

Imperatief wil zeggen dat je voor elk te nemen stapje een instructie geeft aan de computer. De computer zal deze commando's één voor één, in de volgorde zoals ze geschreven staan (van boven naar onder) uitvoeren. In een imperatieve taal kan je werken met het geheugen van de computer door variabelen te maken en aan te passen, zoals je reeds in de module Start to Program deed. Een blok code dat je vaak nodig hebt kan je hergebruiken door dit in een methode (= een functie) te zetten. Je kan ook gebruik maken van reeds bestaande methodes in C#.

Wat *object-georiënteerd* programmeren precies inhoudt gaan we later meer in detail bekijken. We focussen ons eerst even op het imperatieve aspect. De meeste populaire talen zijn imperatieve talen en als je al met een andere populaire programmeertaal hebt gewerkt, zoals Java, Python, PHP, C++, Visual Basic of Javascript, zal je zeker elementen herkennen in C#.

Statements

Elke instructie of stapje dat je aan de computer geeft om uit te voeren wordt een *statement* genoemd. Een statement staat meestal op 1 lijn, zoals wanneer je een variabele aanmaakt of wijzigt. Maar een statement kan ook op meerdere lijnen staan zoals een if-statement of een loop.

In ons programma staat momenteel in het Main-blok één statement: namelijk een Console.WriteLine statement. We gaan dit blok verder uitbreiden.

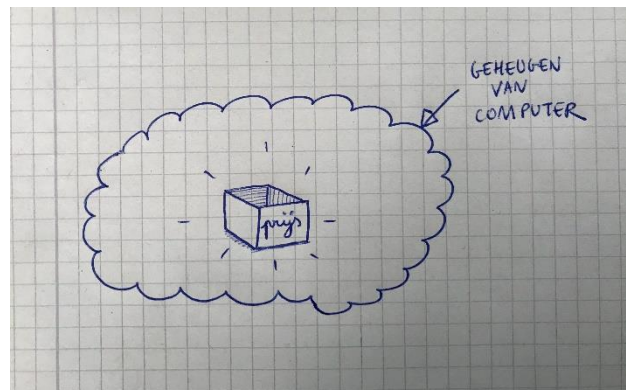
Assignment statement

Het assignment-statement of het toekennings-statement is de regel die een variabele een waarde (= een inhoud) geeft. Je hebt hiervoor eerst een variabele nodig. Een variabele is een geheugenplaats in de computer waarin je iets tijdelijk kan bewaren. Anders dan bij Python of Javascript is het in C# gebruikelijk om het *data-type* (of kortweg type) van een variabele te vermelden bij het aanmaken.

Het data-type bepaalt wat je in de variabele kan bewaren en wat je met de variabele kan doen. Een stukje tekst kan je bijvoorbeeld niet bewaren in een variabele voor getallen. En met een variabele met een stukje tekst in kan je dan weer niet rekenen.

Onderstaande regel maakt een lege variabele aan waarin je een geheel getal (in het Engels *integer*) kan bewaren:

```
int prijs;
```

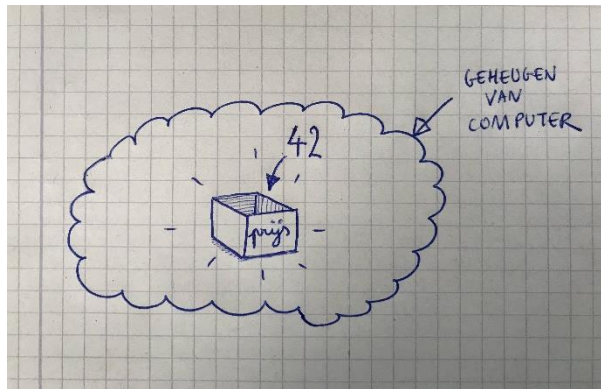


Bovenstaande regel is een variabele *declaratie*: er is nu een lege geheugenplaats met de naam *prijs* gemaakt waarin je later een geheel getal kan bewaren (to declare = vaststellen of aangeven). Zoals je misschien al doorhad is *int* kort voor integer

Nadat de variabele is gemaakt met het woordje *int* mag je deze niet nog eens aanmaken: een tweede keer "int prijs" schrijven geeft een fout omdat de variabele reeds bestaat.

We kunnen nu de geheugenplaats of variabele gaan vullen met een bepaalde waarde:

```
int prijs; // variabele declaratie voor een integer (= geheel getal )
prijs = 42; // toekenning van de waarde 42 aan de variabele "prijs"
```



Het gelijkheidsteken wordt ook wel de assignment operator of toekenningsoperator genoemd omdat je hiermee een waard **toekent** aan een variabele. De dubbele slash is het **commentaarteken**: de tekst hierachter wordt niet uitgevoerd.

Wat gebeurt er als je de bovenstaande 2 regels omwisselt? Werkt de code dan nog? Probeer de code eens uit te voeren en lees de fout-melding. Vind je de fout-melding begrijpbaar?

Meestal zet je de declaratie en toekenning in één regel, zoals hieronder. (Maar soms is het ook nuttig om de declaratie en toekenning op te splitsen in twee regels, zoals we hierboven deden. We zullen later bekijken in welke situaties we de declaratie en toekenning opsplitsen.)

Een declaratie en toekenning in één regel ziet er zo uit:

```
int getal2 = 99;
```

Het is ook handig om de termen in het Engels te kennen omdat dit de taal is waarin je de meeste informatie en documentatie zal vinden.

declaration = variabele aanmaken, bijvoorbeeld met *int*. Dit doe je maar 1 keer per variabele.

assignment = een variabele een inhoud geven met de toekenningsoperator

assignment operator = toekenningsoperator (=)

value = waarde of inhoud van een variabele

Met ints kan je rekenen met +, -, * en /. Probeer het eens uit door deze code in het Main-blok te plaatsen:

```
int breedte = 5;
int hoogte = 12;
int oppervlakteRechthoek = breedte * hoogte;

Console.WriteLine(oppervlakteRechthoek);
```

De 3^{de} regel in bovenstaande code bevat naast een declaratie ook een toekenning met een berekening. Deze regel wil zeggen: *maak een geheugenplaats waar een geheel getal in past en noem deze geheugenplaats "oppervlakteRechthoek"; berekenen de vermenigvuldiging van breedte en hoogte en plaats dit in die net gemaakte geheugenplaats.*

Er zijn overigens nog andere datatypes zoals komma-getallen (floats), strings en booleans.

Oefening 1: "arithmetic expressions" en "assignments"

Omdat programmeurs graag ingewikkelde namen gebruiken wordt een rekensom niet zomaar rekensom genoemd, maar zeggen we *arithmetic expression*.

Voorspel wat er in onderstaande WriteLines getoond zal worden...

```
Console.WriteLine(5 + 3);
Console.WriteLine(-6 - 6 + 2);
Console.WriteLine(5 + 5 * 5);
Console.WriteLine(10 + 10 / 2);

int x = 10;
int y = 5;
int z = 3;

Console.WriteLine(x + y + z);
Console.WriteLine(x + y * z);

y = x;

Console.WriteLine(x * y);

z = z + 2;
x = x * 2;

Console.WriteLine(x);
Console.WriteLine(y);
Console.WriteLine(z);
```

Een input inlezen en in een variabele zetten

Met `Console.ReadLine()` kan je een variabele inlezen vanuit de command line. Deze instructie geeft een string terug. We hebben hiervoor dus een variabele van het type `String` nodig.

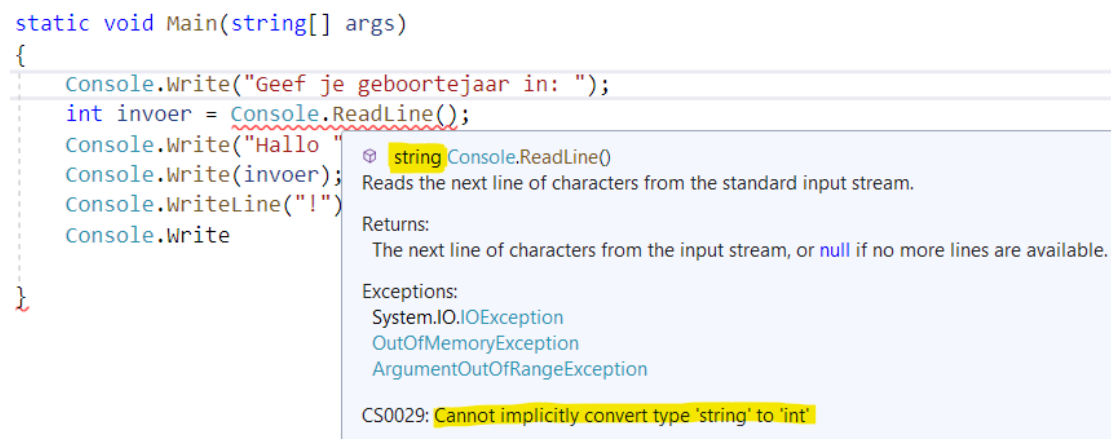
```
Console.Write("Geef je naam in: ");
String invoer = Console.ReadLine();
Console.Write("Hallo ");
Console.Write(invoer);
Console.WriteLine("!");
```

In bovenstaande stukje code geeft de `Console.ReadLine()` de ingegeven tekst door aan de variabele *invoer*. `Console.Write()` is het broertje van `Console.WriteLine()`, het schrijft tekst op het scherm, **maar springt niet naar een volgende lijn** na het schrijven.

Een getal inlezen

`Console.ReadLine()` geeft een waarde van het type `String` terug. Als je dit in een `int` wil zetten zal Visual Studio een fout geven, je ziet een rode zigzaglijn onder de regel staan. Als je met de muis over de rode zigzaglijn zweeft worden er details getoond. Visual Studio zegt dat `Console.ReadLine()` een string terug geeft en dat dit niet automatisch ("implicitly") kan omgezet worden.

```
static void Main(string[] args)
{
    Console.Write("Geef je geboortejahr in: ");
    int invoer = Console.ReadLine();
    Console.Write("Hallo ");
    Console.Write(invoer);
    Console.WriteLine("!");
    Console.Write
```



`string Console.ReadLine()`
Reads the next line of characters from the standard input stream.

Returns:
The next line of characters from the input stream, or `null` if no more lines are available.

Exceptions:
`System.IO.IOException`
`OutOfMemoryException`
`ArgumentOutOfRangeException`

CS0029: Cannot implicitly convert type 'string' to 'int'

We moeten zelf nog de waarde omzetten naar een `int`. We gebruiken hiervoor `int.Parse()`. Dit werkt wel:

```
Console.Write("Geef je geboortejahr in: ");
int getal = int.Parse(Console.ReadLine());
Console.Write("Je gaf dit in: ");
Console.Write(getal);
Console.WriteLine("!");
```


Kommagetallen (floats)

Als je kommagetallen wil gebruiken heb je een ander data-type nodig dan int (int staat voor integer = een geheel getal). Een data-type dat kommagetallen ondersteunt is float.

```
float prijs1 = 2;  
float prijs2 = 0.99F;
```

Als je een getal in een float wil bewaren met een stukje achter de komma moet je een punt gebruiken (i.p.v. een komma) en moet je een F plaatsen achter het getal.

Stappen opschrijven

Als je een programma wil maken, maar niet meteen weet waar je moet beginnen, kan het helpen als je eerst de stappen opschrijft (in gewone taal) die je programma moet doen. Als voorbeeld nemen we een eenvoudig programma dat een berekening maakt van 2 ingegeven getallen: het toont het totaal en het totaal plus 21% btw.

De stappen die je opschrijft zouden zo kunnen zijn:

- Vraag en lees prijs 1
- Vraag en lees prijs 2
- Tel prijs 1 en prijs 2 op en bewaar dit in de variabele *totaal*
- Bereken totaal + btw (= totaal x 1.21) en bewaar dit in de variabele *totaalIncBtw*
- Toon totaal en totaalIncBtw

Vertaald naar C# wordt dit:

```
Console.Write("Geef prijs 1: ");  
float prijs1 = float.Parse(Console.ReadLine());  
  
Console.Write("Geef prijs 2: ");  
float prijs2 = float.Parse(Console.ReadLine());  
  
float totaal = prijs1 + prijs2;  
float totaalIncBtw = totaal * 1.21F;  
  
Console.Write("Het totaal is ");  
Console.Write(totaal);  
Console.WriteLine(" euro");  
  
Console.Write("Het totaal inclusief BTW is ");  
Console.Write(totaalIncBtw);  
Console.WriteLine(" euro");
```

Naarmate de programma's die je maakt ingewikkelder worden zal je meer moeten nadenken en meer en meer de stappen eerst moeten uitdenken en opschrijven vooraleer je gaat coderen. Dit noemt men de analyse. Sommige programmeurs vinden het handig om dit eerst met pen en papier te doen zodat je snel kan schrappen en pijlen kan tekenen.



Bij grotere projecten kan de analyse-fase erg uitgebreid zijn en is deze soms groter dan de codeer-fase.

Oefening 2: Oppervlakte rechthoek

- A. Maak een programma dat de oppervlakte van een rechthoek berekent (en toont). Het programma leest hiervoor 2 getallen in: de lengte en de breedte.
- B. Breidt het programma verder uit zodat het na de oppervlakteberekening van de rechthoek 2 nieuwe getallen (basis en hoogte) inleest en hiermee de oppervlakte van een driehoek berekent.

```
C:\Users\so19079\so...
OPPERVLAKTE RECHTHOEK BEREKENEN
=====
Geef de breedte in cm: 3
Geef de lengte in cm: 14
De oppervlakte is: 42cm2
```

```
C:\Users\so19079\source\repos\Console...
OPPERVLAKTE RECHTHOEK BEREKENEN
=====
Geef de breedte in cm: 4
Geef de lengte in cm: 5
De oppervlakte van de rechthoek is: 20cm2

OPPERVLAKTE DRIEHOEK BEREKENEN
=====
Geef de basis in cm: 13
Geef de hoogte in cm: 3_
```

Wat gebeurt er met komma-getallen? Hoe komt dit? Probeer dit op te lossen met het data-type *float* (gebruik eventueel google).

Wat gebeurt er als je tekst ingeeft i.p.v. een getal? Dit ongewenste gedrag gaan we later oplossen.

if(...) {...}

Met het if-statement kan je een voorwaarde koppelen aan het codeblok binnen de accolades. De code binnen het codeblok wordt dan uitgevoerd als de voorwaarde *waar* of *true* is. De voorwaarde of *condition* komt binnen de ronde haken van de if te staan:

```
if(condition)
{
    // codeblok
}
```

Een voorbeeldje:

```
String voornaam = Console.ReadLine();

if(voornaam == "Kevin")
{
    Console.WriteLine("Dag meneer")
}
```

In de voorwaarde hierboven vergelijken we de ingegeven voornaam met de tekst “Kevin” met het **dubbele** gelijkheidsteken (==). Het dubbele gelijkheidsteken is de **gelijkheidsoperator** of **equality operator**.

In het vorige stukje code zal de tekst “Dag meneer” enkel getoond worden als de gebruiker “Kevin” ingeeft. Je kan dit zo lezen: “Als de waarde van de variabele *voornaam* **gelijk is aan** de tekst “Kevin” voer dan de code (tussen de accolades) uit.”.

De **equality operator** (==) geeft de waarde *true* als wat voor de operator staat **gelijk is aan** wat er achter staat. Als de 2 waarden verschillen geeft de operator de waarde *false*.

Het zusje van de equality operator is de **inequality operator**, het uitroepteken-gelijkheidsteken:

```
!=
```

Met de inequality operator test je of 2 waarden **verschillend** zijn.

De **inequality operator** (!=) geeft de waarde *true* als wat voor de operator staat **verschillend is van** wat er achter staat. Als de 2 waarden gelijk zijn geeft de operator de waarde *false*. Je kan dit lezen als “is niet gelijk aan” of “is verschillend van”.

Een voorbeeld:

```
String voornaam = Console.ReadLine();

if(voornaam != "Kevin")
{
    Console.WriteLine("Dag cursist!");
}
```

Je kan dit zo lezen: “Als de waarde van de variabele *voornaam* verschillend is van de tekst “Kevin” voer dan de code uit.”.

Getallen vergelijken

Naast de gelijkheid (==) en ongelijkheid (!=) heb je voor getallen ook nog groter dan, kleiner dan, groter dan of gelijk aan en kleiner dan of gelijk aan.

| | |
|----|---------------------------|
| < | Kleiner dan |
| > | Groter dan |
| <= | Kleiner dan of gelijk aan |
| >= | Groter dan of gelijk aan |

Welke boodschap zal er hieronder getoond worden?

```
int leeftijd = 18;

if(leeftheid >= 18)
{
    Console.WriteLine("Eindelijk!");
}

if(leeftheid < 18)
{
    Console.WriteLine("Nog even wachten...");
}
```

if/else

Je kan door gebruik te maken van *if* en *else* 2 code-blokken meegeven. Als de voorwaarde die je bij de if vermeldt *true* is zal het eerste code-blok worden uitgevoerd (het tweede blok wordt dan overgeslagen). Als de voorwaarde *false* is wordt enkel het tweede code-blok bij de else uitgevoerd.

```
String voornaam = Console.ReadLine();

if (voornaam == "Kevin")
{
    Console.WriteLine("Dag meneer!");
}
else
{
    Console.WriteLine("Dag cursist!");
}
```

if/else if/.../else

Je kan de if ook verder gaan opsplitsen:

```
String voornaam = Console.ReadLine();

if (voornaam == "Sophie")
{
    Console.WriteLine("Dag directeur!");
}
else if (voornaam == "Kevin")
{
    Console.WriteLine("Dag meneer!");
}
else
{
    Console.WriteLine("Dag cursist!");
}
```

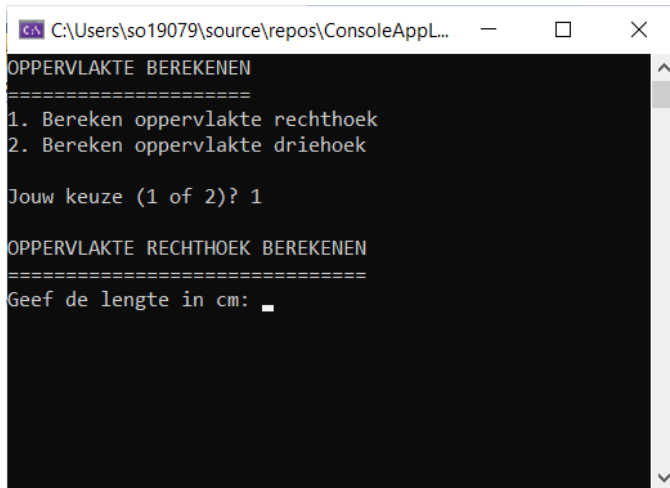
Een voorwaarde of **condition** die enkel *true* of *false* kan zijn wordt ook wel een **boolean condition** genoemd (naar de wiskundige Boole).

Oefening 3: Oppervlakte berekenen met keuze-menu

- A. Maak een keuze-menu voor de oppervlakte-berekeningen. Bij keuze 1 vraag je de lengte en breedte op om de oppervlakte van een rechthoek te berekenen, bij keuze 2 de basis en hoogte van een driehoek.

Voorzie een boodschap "Foute keuze :(" voor als er iets anders wordt ingetypt dan 1 of 2.

Gebruik je hier best strings of ints?



```
C:\Users\so19079\source\repos\ConsoleAppL...
OPPERVLAKTE BEREKENEN
=====
1. Bereken oppervlakte rechthoek
2. Bereken oppervlakte driehoek

Jouw keuze (1 of 2)? 1

OPPERVLAKTE RECHTHOEK BEREKENEN
=====
Geef de lengte in cm: _
```

- B. Voeg een derde keuze toe om de oppervlakte van een cirkel te berekenen aan de hand van de straal. Weet je hier nog de formules voor? Zoek eventueel dingen op.

while(...) {...}

Het while-statement lijkt qua schrijfwijze op het if-statement (zonder else).

```
while(condition)
{
    // codeblok
}
```

Je geeft hier ook een voorwaarde mee tussen de ronde haakjes en er is ook een codeblok tussen accolades. De voorwaarde wordt hier ook weer gecontroleerd op true voor het codeblok wordt uitgevoerd, net zoals bij de if.

Het grote verschil is dat de while na het uitvoeren van het codeblok de voorwaarde **terug opnieuw** gaat controlleren of deze true is. Als deze voorwaarde nog steeds true is wordt het codeblok opnieuw uitgevoerd, net zolang tot de voorwaarde false wordt. De while wordt dus gebruikt om zaken te herhalen of te *lopen*.



Je moet er dus wel voor zorgen dat de voorwaarde op een gegeven moment false wordt anders blijft de code “oneindig” herhalen...

Wat denk je dat onderstaand voorbeeld doet? **Schrijf de uitvoer die je verwacht op alvorens je de code test.**

```
int i = 0;

while(i<10)
{
    Console.WriteLine(i);
    i = i + 1;
}
```

Wat denk je dat onderstaand voorbeeld doet?

```
String keuze = Console.ReadLine();

while(keuze != "quit")
{
    keuze = Console.ReadLine();
}
```

OEFENINGEN

4. Oppervlakte berekenen - herhalend

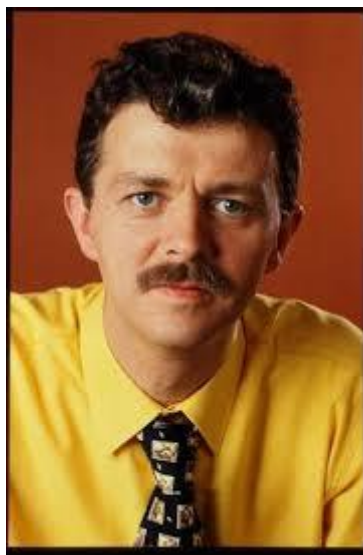
Breidt het programma dat de oppervlakte berekent verder uit zodat de gebruiker kan blijven berekenen totdat ze beslist om te stoppen. Je kan hiervoor if-jes gebruiken in een while.

Zoek op hoe je de command line leeg kan maken (clear) zodat je het menu kan verbergen nadat er een keuze werd gemaakt door de gebruiker:

5. Gemiddelde neerslag berekenen

Een bekende weerman wil de gemiddelde neerslag berekenen van de voorbije dagen. Hij wil de neerslagcijfers kunnen ingeven en het gemiddelde laten berekenen.

Maak hiervoor een programma dat verschillende positieve getallen inleest tot een negatief getal wordt ingegeven. Na het ingeven van de getallen wordt het gemiddelde van de getallen berekend en getoond.



6. Rekenen

Maak een programma waarmee je eenvoudige rekensommen kan maken. De gebruiker kan via een menu kiezen of hij wil gaan optellen, aftrekken, vermenigvuldigen of delen. Vervolgens worden er 2 getallen gevraagd en wordt de juiste bewerking gemaakt en de uitkomst getoond.

Zorg dat de gebruiker kan blijven rekenen tot ze beslist om te stoppen. Maak het scherm tussendoor leeg.

