

Variabelen

Programming Basics

INHOUD

1	STATEMENTS	3
2	IDENTIFIERS	4
3	VARIABLEN	5
3.1	Filmpje	5
3.2	Variabelen gebruiken	5
3.3	Variabelen benoemen	5
3.4	Variabelen declareren	5
3.4.1	Voorbeeld	6
3.5	Scope van een variabele	9
4	PRIMITIEVE DATATYPES	11
4.1	Omzetting van het ene datatype/object naar het andere	13
4.1.1	Van ... naar ...	13
1.1.1.1	Convert	13
1.1.1.2	Casting	13
4.1.2	Van ... naar string.	14
4.1.3	Van een string naar een getal	14
1.1.1.3	Parse	14
4.2	Impliciet getypeerde lokale variabelen	14
5	WISKUNDIGE OPERATOREN	16
5.1	Overzicht	16
5.2	Toepassing	16
6	SAMENGESTELDE TOEKENNING	20

1 STATEMENTS

Een statement is een opdracht die een actie uitvoert. Meerdere statements kunnen gegroepeerd worden in methoden.

Statements volgen in C# een set strikte regels die hun opbouw beschrijft. Deze regels worden algemeen **syntax** genoemd. Een fout tegen de syntactische regels van een programmeertaal is dan een '**syntax error**'. Eén van deze syntaxregels is het feit dat een statement steeds gevolgd wordt door een **;** (puntkomma / semicolon).

Voorbeeld van een statement:

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

Voorbeeld van een methode met twee statements:

```
public void DoubleTwo()
{
    Console.WriteLine("Het dubbele van 2 is 4");
    Console.ReadLine();
}
```

2 IDENTIFIERS

Elementen zoals namespaces, klassen, methoden, variabelen, controls, ... krijgen in C# een naam. De naam die je aan een element geeft, noemen we de **identifier**. Een identifier moet aan volgende regels voldoen

- Enkel letters (kleine en hoofdletters), cijfers en het teken _ (underscore) zijn toegelaten
- Het eerste teken mag **geen** cijfer zijn
- C# is hoofdlettergevoelig: **resultaat** is niet gelijk aan **Resultaat**
- C# reserveert een aantal woorden (**reserved identifiers**) voor intern gebruik. Deze kan je niet opnieuw gebruiken als identifier (bv. if, base, ...)



Identifiers

Een lijst van gereserveerde sleutelwoorden vind je bij:

- [MSDN \(Microsoft Developers Network\)](#)

3 VARIABELEN

3.1 FILMPJE

Op Udacity.com vind je een filmpje waarin heel eenvoudig wordt uitgelegd wat een variabele is. Je vindt het in de cursus 'Android Basics: User Input', Making an App Interactive: Part 1, 9. Quiz: The Need for Variables. Eens je je aangemeld hebt voor deze gratis cursus, kun je deze [link](#) gebruiken.

3.2 VARIABELEN GEBRUIKEN

Een variabele is een opslagruimte die een waarde kan bevatten. Je kan een variabele zien als een stukje geheugen dat je een naam hebt gegeven. Dit geheugenplekje bevat informatie die gewijzigd kan worden in de loop van het programma. Een variabele kan tekst, getallen, datums ... bevatten.

Tijdens het programmeren zal je veelvuldig gebruik maken van variabelen.

3.3 VARIABELEN BENOEMEN

Een variabele wordt steeds aangesproken met de naam. Het is belangrijk dat je voor jezelf een goede manier ontwikkelt om variabelen een naam te geven. Niets is zo vervelend als programmacode te moeten lezen die vol staat van variabelennamen zoals a, b, c, ... of number1, number2, number3,

Bij het lezen van dergelijke code moet je je als programmeur steeds gaan afvragen welke informatie de variabele bevat. Veel duidelijker zijn variabelennamen zoals purchaseAmount, salesAmount, profit, vatAmount, total, ... Voor de naamgeving van variabelen hanteren we volgende afspraken (conventies):

- Begin een variabelennaam **niet** met een hoofdletter.
- Gebruik **camelCase** notatie: start bij namen die bestaan uit meerdere woorden elk woord met een hoofdletter (niet het eerste), alle andere letters zijn kleine letters.
- Gebruik **geen** Hongaarse notatie: een (vroeger) veel gebruikte manier om variabelen een naam te geven, waarbij aan de naam een prefix werd geschreven die duidt op het type van de variabele. We gebruiken dit wel voor de naamgeving van WPF controls, maar **niet** voor de naamgeving van variabelen.

3.4 VARIABELEN DECLAREREN

Variabelen bewaren waarden. In C# kunnen veel verschillende types van waarden opgeslagen en verwerkt worden: gehele getallen (integer), kommagetallen (floating-point number), tekst (string), tekens (character), ... Bij de declaratie van een variabele geef je aan welk type gegevens de variabele zal gaan bijhouden. In het declaratie-statement bepaal je het type en de naam van een variabele:

```
int age;  
string lastName;
```

Hier declareer je een variabele met de naam 'age' van het type integer (geheel getal) en een variabele met de naam 'lastName' van het type string (tekenreeks/tekst).

- Merk nogmaals op dat elk statement eindigt met een komma-punt.
- Het type `int` is één van de primitieve datatypes binnen C# (meer hierover verderop).
- In C# ben je verplicht elke variabele te declareren vooraleer je deze gebruikt!
- Nadat je een variabele hebt gedeclareerd of bij de declaratie kan je deze een waarde toewijzen:

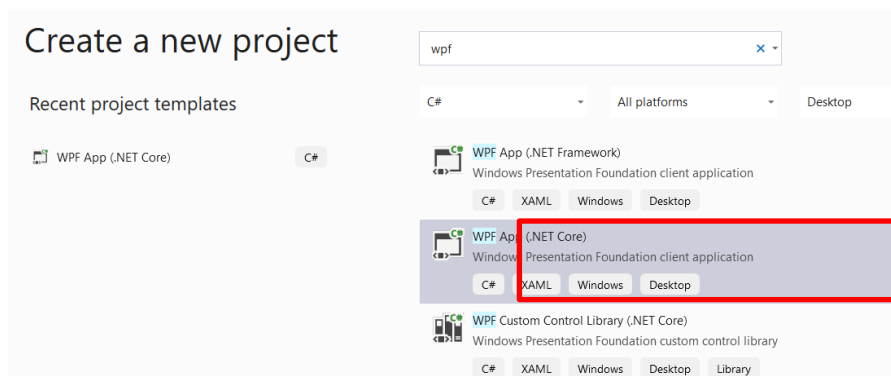
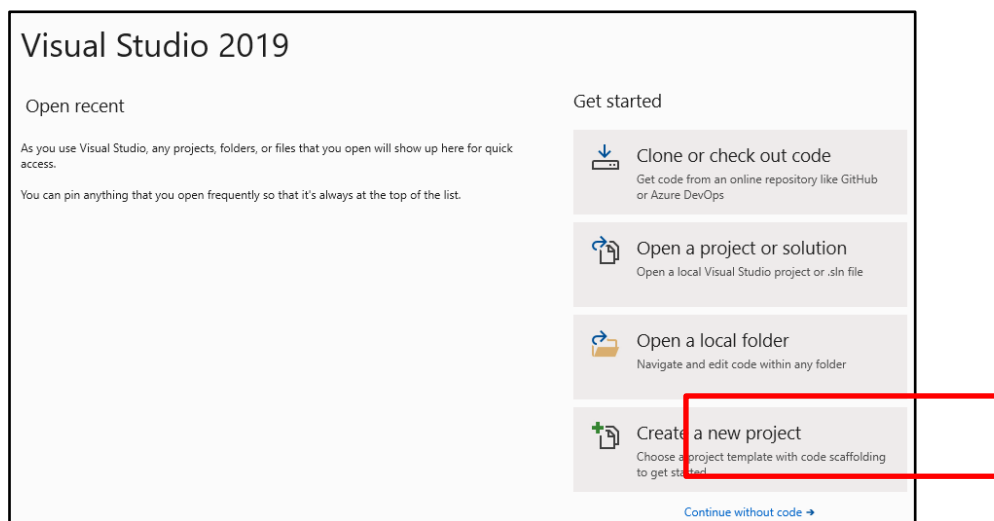
```
int age;
age = 42;
int price = 15;
```

- Het gelijkheidsteken is in C# de toekenningoperator: je kent de waarde 42 toe aan de variabele `age`.
- Je kan in C# een variabele declareren en initialiseren (een initiële waarde toekennen) in één coderegel:

```
int age = 42;
```

3.4.1 VOORBEELD

Maak een nieuw Wpf project aan met de naam `Prb.Variables.Hello.Wpf`.



WPF Application C# Windows Desktop

Project name

Prb.Variables.Hello.Wpf

Location

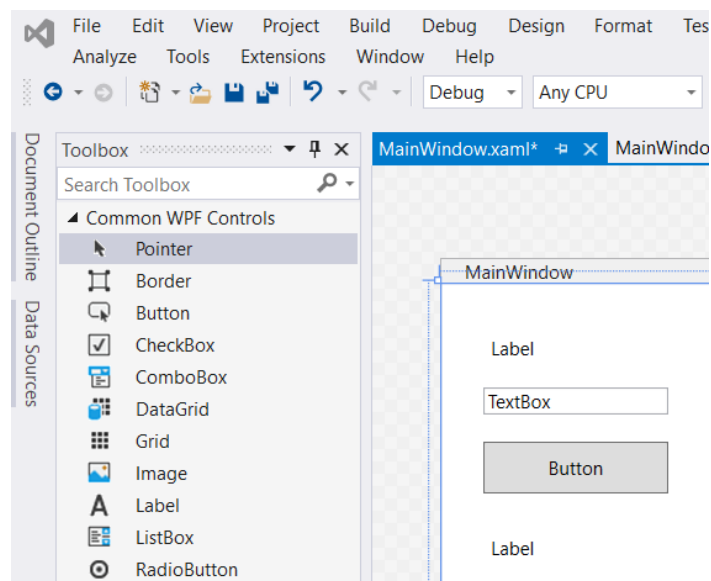
C:\Users\herman.de.beukelaer\source\repos

Solution name ⓘ

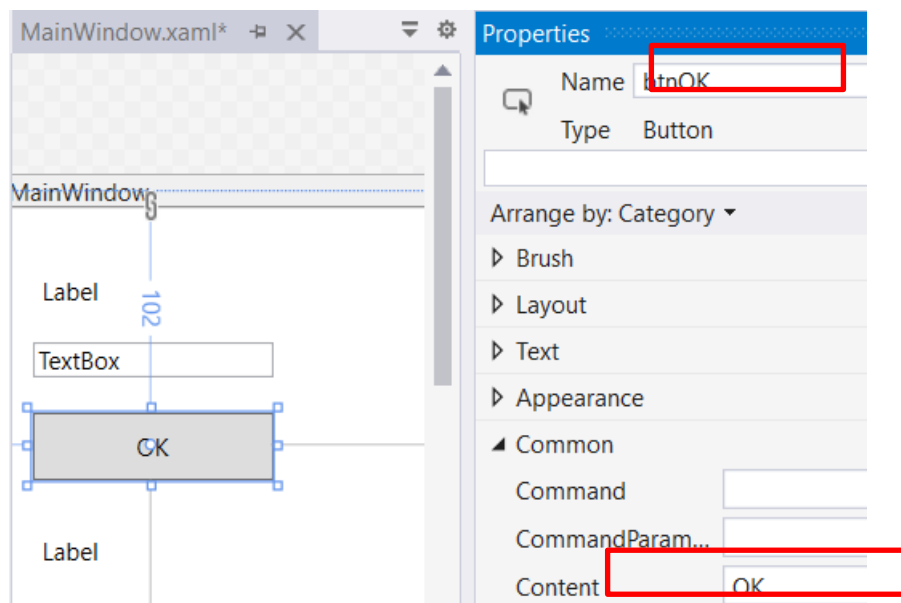
Prb.Variables.Hello

☐ Place solution and project in the same directory

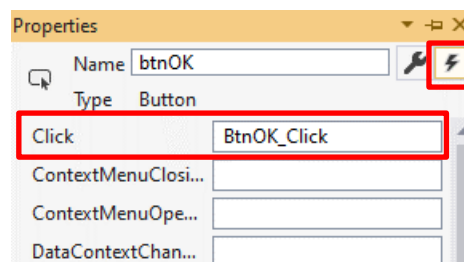
Sleep een textbox, een button en twee labels naar het formulier.



Pas de naam (txtUserName, btnOK, lblUpperCase) en de tekst van de objecten aan.



Maak de event handler methode BtnOK_Click aan:



```
private void BtnOK_Click(object sender, RoutedEventArgs e)
{
    string name = txtUserName.Text;
    name = name.ToUpper();
    lblUpperCase.Content = name;
}
```

Declareer een string met de naam *name*. Initialiseer deze string met de waarde van de eigenschap Text van de control txtUserName: de TextBox.

```
string name = txtUserName.Text;
```

Ken aan de variabele *name* de waarde toe van de variabele *name* waarop de methode ToUpper() werd toegepast: alle letters worden hoofdletters:

```
name = name.ToUpper();
```


De waarde van de string `name` wordt getoond in het label met de naam `lblUpperCase`.

```
lblUpperCase.Content = name;
```

3.5 SCOPE VAN EEN VARIABLE

Als we een variabele declareren, is die geldig voor gans het stuk code waarin ze gedeclareerd wordt. Een variabele kan geldig zijn binnen een codeblok (bvb. een lus of selectieblok, waarover later), een methode, een klasse of een project. Eens het codeblok, de procedure ... uitgevoerd zijn, verdwijnt de variabele uit het geheugen.

De volgende codevoorbeelden maken dit duidelijk. **Voeg nog een label toe aan je MainWindow en noem dit `lblFeedback`.**

Hieronder wordt een variabele naam gedeclareerd in de methode `TxtUserName_TextChanged`.

```
namespace Prb.Variables.Hello.Wpf
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void TxtUserName_TextChanged(object sender, TextChangedEventArgs e)
        {
            string name;
            name = txtUserName.Text;
            lblFeedback.Content = "Je naam is " + name;
        }
    }
}
```

Eens de methode doorlopen werd, is de informatie die in `name` zat terug verdwenen. Willen we iets met die variabele aanvangen onder een button bvb., dan moeten we de tekst uit `txtUserName` terug oproepen.

```
private void BtnOK_Click(object sender, RoutedEventArgs e)
{
    string name;
    name = txtUserName.Text;
    MessageBox.Show("Ben jij werkelijk dé " + name);
}
```

Eigenlijk hebben we nu twee variabelen met dezelfde naam, die een onafhankelijk leven leiden van elkaar. Als we de variabele op het niveau van de partial class `MainWindow` declareren, blijft ze beschikbaar zolang

de MainWindow actief is. Vanuit elk codeonderdeel kan de waarde van *name* dan opgevraagd en aangepast worden.

```
public partial class MainWindow : Window
{
    string name;

    public MainWindow()
    {
        InitializeComponent();
    }

    private void TxtUserName_TextChanged(object sender, TextChangedEventArgs e)
    {
        name = txtUserName.Text;
        lblFeedback.Content = "Je naam is " + name;
    }

    private void BtnOK_Click(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Ben jij werkelijk dé " + name);
    }
}
```

In de bovenstaande code kunnen we de variabele *name* overal binnen MainWindow gebruiken: in `txtUserName_TextChanged` kennen we een waarde toe, in `BtnOK_Click` vragen we de waarde op.



Code repository

De volledige broncode van dit voorbeeld is te vinden op

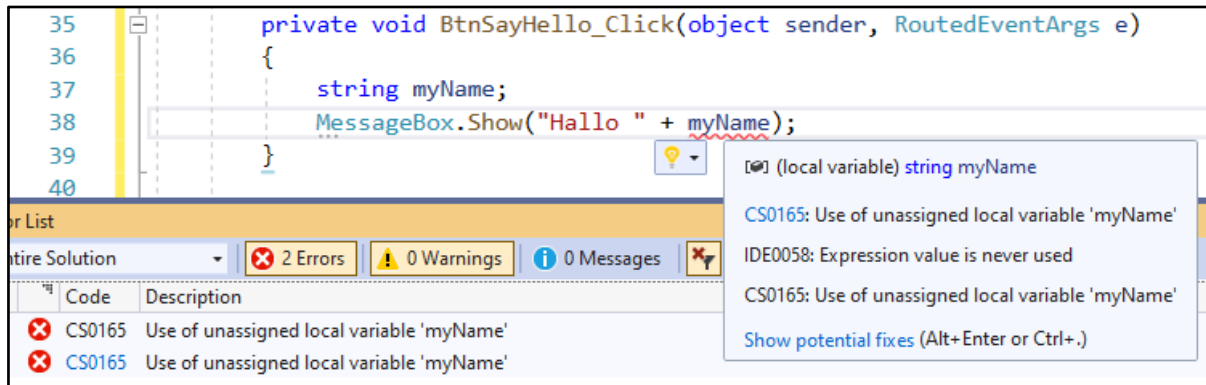
 `git clone` <https://github.com/howest-gp-prb/cu-h2-variabelen-hello.git>

4 PRIMITIEVE DATATYPES

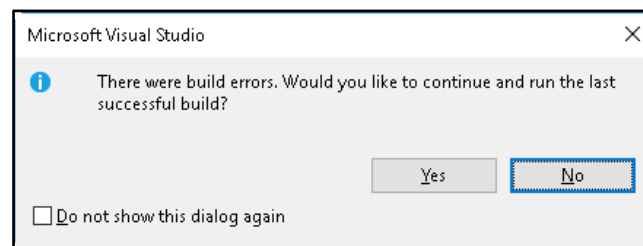
C# heeft een aantal ingebouwde types die we **primitieve datatypes** noemen:

data type	omschrijving	grootte (bits)	bereik	voorbeeld
byte	positieve gehele getallen	8	0 – 255	byte wheels; wheels = 4;
sbyte	gehele getallen	8	-128 - 127	sbyte story; story = -2;
short	gehele getallen	16	-32 768 to 32 767	short salary salary = 1500
int	gehele getallen	32	-2^{31} tot 2^{31}	int count; count = 42;
long	gehele getallen	64	-2^{63} tot 2^{63}	long wait; wait = 42L;
float	kommagetallen (pos / neg)	32	$1,5 \times 10^{-45}$ tot $3,4 \times 10^{38}$	float test; test = 0.42F;
double	preciezere kommagetallen (pos / neg)	64	5×10^{-324} tot $1,7 \times 10^{308}$	double trouble; trouble = 0.42;
decimal	geldbedragen	128	28 betekenisvolle cijfers	decimal coin; coin = 0.42M;
string	tekenreeks	16 bits / teken	nvt	string car; car = "Honda";
char	1 teken	16	0 tot $2^{16} - 1$	char sex; sex = 'V';
bool	booleaanse waarde: waar / onwaar	8	Waar / onwaar (0 / 1)	bool found; found = false;

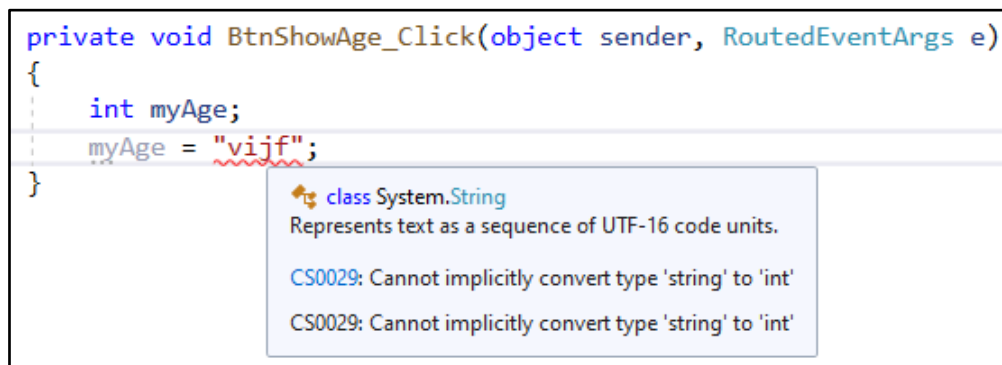
In C# krijg je een error wanneer je een niet toegekende variabele wenst te gebruiken:



Als je toch probeert om de code te bouwen, krijg je een foutboodschap:

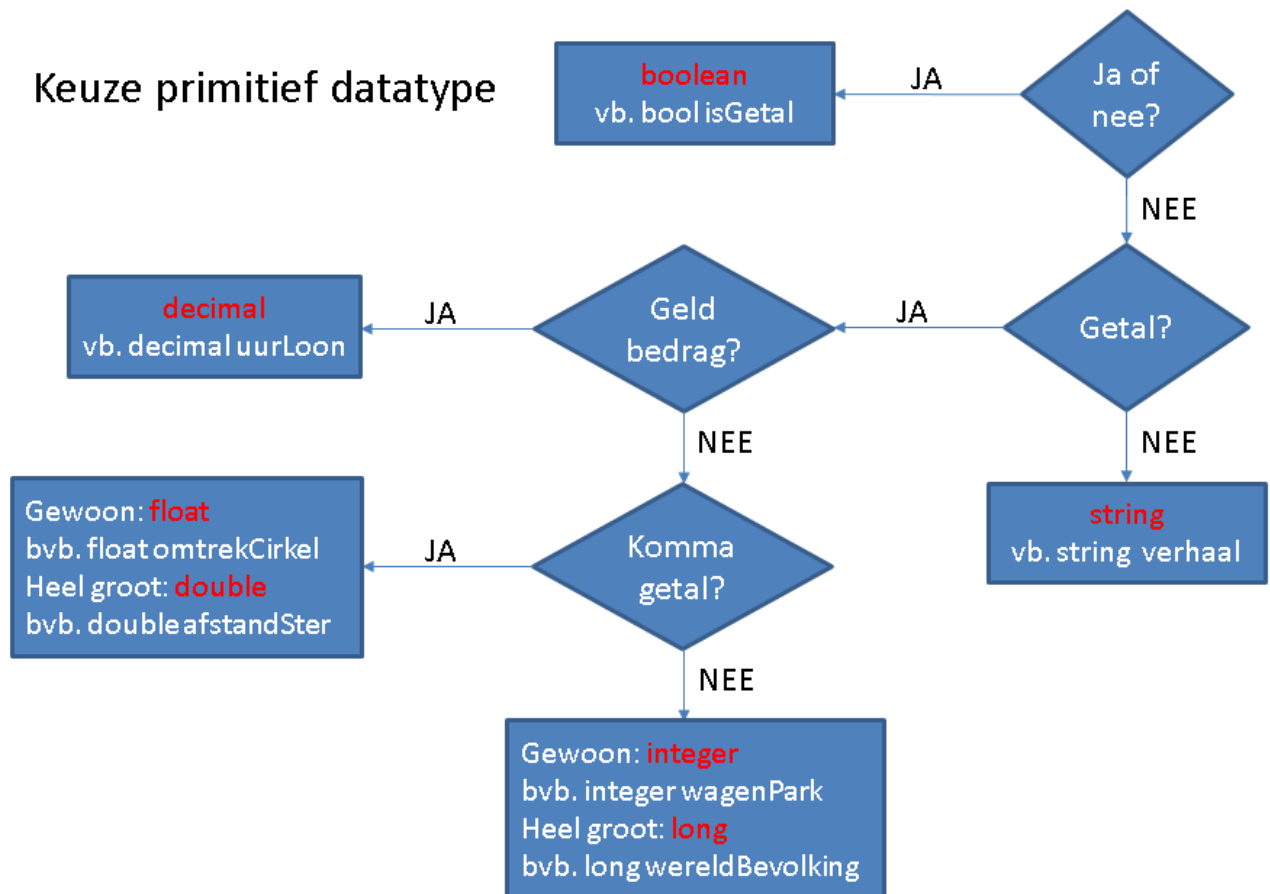


Ook als je een ongeldige waarde aan een variabele wil toekennen, wijst Visual Studio je terecht:



Je kunt onderstaand schema gebruiken om te beslissen welk datatype je best gebruikt voor een variabele.

Keuze primitief datatype



4.1 OMZETTING VAN HET ENE DATATYPE/OBJECT NAAR HET ANDERE

4.1.1 VAN ... NAAR ...

1.1.1.1 CONVERT

Met de Convert-functie kun je allerlei objecten omzetten naar allerlei andere types, zoals boolean, string, getallen...

Hier converteren we bijvoorbeeld een getal naar een boolean. Nul wordt dan false, alle andere getallen true. Was alles maar zo simpel in het leven...

```
int number = 0;
bool trueOrFalse = Convert.ToBoolean(number);
Console.WriteLine("Getal = " + trueOrFalse);
```

1.1.1.2 CASTING

Casting is een conversie van het ene type naar het andere. Vóór het te converteren object wordt het gewenste type tussen haakjes geplaatst.

```
long longNumber = 0;
int number = (int)longNumber;
```

Net hetzelfde resultaat zou je bekomen met

```
long longNumber = 0;
int number = Convert.ToInt32(longNumber);
```

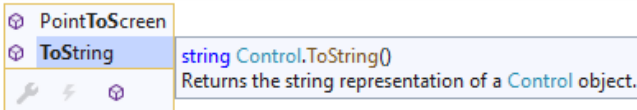
Met casting kunnen we ook meer complexe conversies uitvoeren.

```
ListBoxItem city;
city = (ListBoxItem)lstCities.SelectedValue;
```

4.1.2 VAN ... NAAR STRING.

Aan de datatypes die we hierboven leerden kennen, zijn mogelijke functies gekoppeld. Zo kun je in een string karakters opsporen en veranderen, de lengte bepalen... Heb je een object waarop je die functies wil uitoefenen, dan kun je er eerst de functie ToString() op toepassen. Daarna kun je de stringfuncties op het omgezette object toepassen.

```
Button test = new Button();
string testString = test.tos|
```



In bovenstaand voorbeeld kun je op testString alle functies van een string uitoefenen.

4.1.3 VAN EEN STRING NAAR EEN GETAL

1.1.1.3 PARSE

Soms krijg je in je code een numerieke waarde binnen in string-formaat. Wil je daarop wiskundige bewerkingen uitvoeren, dan moet je die tekst omzetten naar een getal, 'parsen' in het jargon.

```
string numberAsText = "1";
int number = int.Parse(numberAsText);
Console.WriteLine("Getal = " + number);
```



Meer informatie

- [Meer info bij Microsoft](#)

4.2 IMPLICIET GETYPEERDE LOKALE VARIABELEN


C# ondersteunt ook het gebruik van het sleutelwoord **var** voor de declaratie van variabelen:

```
var myIntVar = 5;
var myStringVar = "Hallo";
```

Nu zal .Net Core na initialisatie zelf het type bepalen.

Een variabele gedeclareerd met var moet onmiddellijk geïnitieerd worden. Het type kan achteraf niet meer gewijzigd worden.

```
var myAge = 5;  
myAge = "vijf";
```

 **class System.String**
Represents text as a sequence of UTF-16 code units.

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

5 WISKUNDIGE OPERATOREN

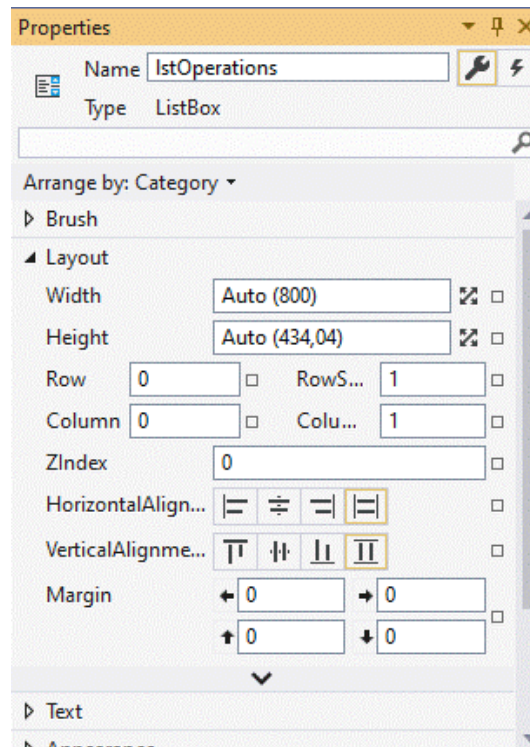
5.1 OVERZICHT

operator	beschrijving	voorbeeld
+	optellen	<code>int sum; sum = 5 + 3;</code>
-	afrekken	<code>int diff; diff = 5 - 3;</code>
*	vermenigvuldigen	<code>int product; product = 5 * 3;</code>
/	delen	<code>double quotient; quotient = 5 / 3;</code>
%	rest bij gehele deling (modulo)	<code>int remainder; remainder = 5 % 3;</code>
++	verhoog met 1	<code>int i; i = 1; i++;</code>
--	verminder met 1	<code>int i; i = 10; i--;</code>

5.2 TOEPASSING

1. Maak een nieuwe Solution `Prb.Variables.Operators` aan in Visual Studio met hierin een Wpf project `Prb.Variables.Operators.Wpf`.
2. Sleep vanuit de Toolbox een `ListBox` binnen de Grid van het WPF-Window en geef hem de naam `lstOperations`.
Een `ListBox` kan je in WPF gebruiken om een lijst van waarden weer te geven. Het doel van de toepassing is een aantal zinnen onder elkaar op het scherm te tonen.

3. Pas eventueel de waarden voor de eigenschappen `Margin`, `HorizontalAlignment`, `VerticalAlignment`, `Width` en `Height` aan bij de properties (**F4**) zoals hieronder getoond.

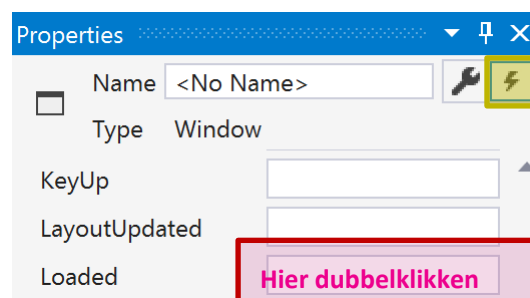


De ListBox vult nu het volledige Window.

In de XAML-code zien we dit:

```
<Window x:Class="Prb.Variables.Operators.Wpf.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Prb.Variables.Operators.Wpf"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <ListBox x:Name="lstOperations"/>
    </Grid>
</Window>
```

1. Selecteer het Window MainWindow en klik het tabblaadje **Events**.
2. Dubbelklik naast het event Loaded:



De methode `Window_Loaded` zal uitgevoerd worden wanneer het Window volledig geladen is. De programmacode die we schrijven voor het event `Loaded` van een Window zal dus meteen uitgevoerd worden als het Window geladen (opgestart) wordt. We gebruiken ze als we code willen schrijven die besturingselementen aanstuurt (opvult, wijzigt ...).

1. In `MainWindow.xaml.cs` werd nu volgende code automatisch voorzien:

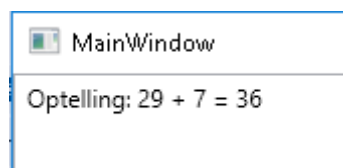
```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
}
```

De methode `Window_Loaded`.

2. Voeg volgende code toe. Gebruik waar mogelijk intellisense van Visual Studio:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    lstOperations.Items.Add("Optelling: 29 + 7 = " + (29 + 7));
}
```

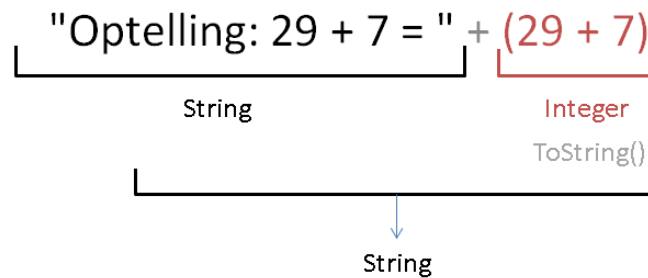
3. Voer de toepassing uit:



De naam van de `ListBox` die je in het Window hebt geplaatst, is `lstOperations`. Een `ListBox` kan een lijst van elementen op het scherm tonen. In de code kan je hem aanspreken/manipuleren met zijn naam. De eigenschap `Items` van de `ListBox` `lstOperations` bevat de elementen die in de `ListBox` aanwezig zijn (`ItemCollection`). Met de methode `Add` van de `ItemCollection` kun je een element aan de `ListBox` toevoegen. Dit is hetgeen gebeurt met de opdracht `lstOperations.Items.Add(...)`.

Uit dit voorbeeld kan je leren dat de operator + voor getallen voorziet in de som, en voor strings tekenreeksen samenvoegt (concatenering):

- "Optelling: 29 + 7 = " is een string
- (29 + 7) is de optelling van twee gehele getallen 29 en 7 en levert 36.
- De eerste tekenreeks wordt nu met de operator + samen gevoegd met het gehele getal 36; hiervoor wordt het gehele getal achter de schermen omgezet in een string (methode ToString()-zie later), de operator + voor twee tekenreeksen leidt tot een samen voeging van de twee tekenreeksen.



4. Voeg nog een reeks strings toe aan de ListBox:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    lstOperations.Items.Add("Optelling: 29 + 7 = " + (29 + 7));
    lstOperations.Items.Add("Aftrekking: 29 - 7 = " + (29 - 7));
    lstOperations.Items.Add("Vermenigvuldiging: 29 * 7 = " + (29 * 7));
    lstOperations.Items.Add("Deling: 29 / 7 = " + (29F / 7F));
    lstOperations.Items.Add("Modulo : 29 % 7 = " + (29 % 7));

    int number = 0;
    lstOperations.Items.Add("Waarde van getal: " + number);
    lstOperations.Items.Add("Verhogen met 1 : ++getal = " + ++number);
    lstOperations.Items.Add("Verlagen met 1 : --getal = " + --number);
    lstOperations.Items.Add("2 tot de 3de macht: " + Math.Pow(2, 3));
    lstOperations.Items.Add("De vierkantswortel van 16: " + Math.Sqrt(16));
}
```

Bij de deling is het interessant op te merken dat minstens één van de leden van de bewerking een kommagetal moet zijn (**suffix F**). Anders wordt een gehele deling uitgevoerd en gaan cijfers achter de komma dus verloren.

Wanneer meerdere bewerkingen in één statement worden opgenomen, worden deze als volgt uitgevoerd:

- Rekening houden met eventuele haakjes.
- Rekening houden met de volgorde der bewerkingen (vb. eerst vermenigvuldigen / delen, daarna optellen of aftrekken).
- Uitvoering van links naar rechts.

6 SAMENGESTELDE TOEKENNING

Je weet reeds dat je een waarde aan een variabele kan toekennen met de toekenningoperator `=`

```
int numberOfStudents;  
numberOfStudents = 10;
```

Je kunt de wiskundige hoofdbewerkingen uitvoeren:

```
numberOfStudents = 10 + 5;
```

Op de onderstaande manier kan je de originele waarde van een variabele aanpassen:

```
numberOfStudents = numberOfStudents + 8;  
numberOfStudents = numberOfStudents - 2;  
numberOfStudents = numberOfStudents * 3;  
numberOfStudents = numberOfStudents / 3;  
numberOfStudents = numberOfStudents % 4;
```

Dit kan korter door gebruik te maken van een samengestelde toekenningoperator:

```
numberOfStudents += 8;  
numberOfStudents -= 2;  
numberOfStudents *= 3;  
numberOfStudents /= 3;  
numberOfStudents %= 4;
```


Je kunt dit ook toepassen op andere types variabelen dan getallen.

```
string name = "Kenji";  
string lastName = "Minogue";  
name += " " + lastName;
```



Code repository

De volledige broncode van deze applicatie is te vinden op

 `git clone https://github.com/howest-gp-prb/cu-h2-variabelen-wiskundige-operatoren.git`

