

GRADUAAT IN HET
PROGRAMMEREN

Visual C#.NET met WPF Syllabus

Cursus

C# Essentials

Opleidingsonderdeel

Informatica | Programmeren

Afdeling

Sander De Puydt
Koen Bloemen
Wim Bervoets
Kristof Palmaers

Auteur



DE HOGESCHOOL
MET HET NETWERK

Overzicht

INLEIDING	3
HOOFDSTUK 1 PROBLEEMOPLOSSEND DENKEN	12
HOOFDSTUK 2 OPERATOREN	17
HOOFDSTUK 3 INFORMATIE OPSLAAN IN VARIABELEN EN CONSTANTEN	20
HOOFDSTUK 4 CONVERSIES	24
HOOFDSTUK 5 CONTROLESTRUCTUREN	28
HOOFDSTUK 6 VEEL GEBRUIKTE KLASSES	32
HOOFDSTUK 7 METHODEN EN PARAMETERS	41
HOOFDSTUK 8 EVENTPROCEDURES EN PARAMETERS	45
HOOFDSTUK 9 WPF BESTURINGSELEMENTEN	48
HOOFDSTUK 10 GEGEVENS BEHEREN MET EEN ARRAY	62
HOOFDSTUK 11 GENERIEKE LISTCOLLECTIE	69
HOOFDSTUK 12 FOUTEN OPSPOREN EN DE GESTRUCTUREERDE FOUTAFHANDELING	73
HOOFDSTUK 13 CONSOLE TOEPASSINGEN	80
BIJLAGEN	83

Inleiding

Dankwoord

Met dank aan Patricia Briers voor de vele jaar inzet in het beheren van deze cursus.

1 Microsoft Visual Studio

Microsoft Visual Studio is een **programmeerontwikkelomgeving** van Microsoft. Het biedt **een complete set ontwikkelingstools** om computerprogramma's in diverse programmeertalen voor Windows-omgevingen te ontwikkelen. Momenteel biedt het ondersteuning aan 36 computertalen.

De talen Visual Basic, Visual C#, Visual C++, Visual F#, Java en Python worden in de standaardeditie meegeleverd. Het wordt gebruikt om ASP.Net-webapplicaties, XML-webservices, desktopapplicaties en mobiele toepassingen te ontwerpen. Het is uitermate geschikt voor het ontwerpen van Windows-applicaties omdat Visual Studio het mogelijk maakt om op een eenvoudige wijze Windows-kenmerken zoals vensters en keuzemenu's aan een programma te geven.

Kenmerken

Het pakket bestaat naast een **editor** om de programmacode in te typen uit de **compilers** voor de verschillende talen, een **debugger**, een **visuele ontwerper** voor (dialoog) vensters en een **profiler** (analyseren van de uitvoersnelheid en het geheugengebruik). Verder wordt er ook een gehele versie van de MSDN-library meegeleverd. Visual Studio is ontworpen om het **.NET-framework** te ondersteunen.

Geschiedenis

2008

Op 19 november 2007 presenteerde Microsoft Visual Studio 2008, die de **.NET-filosofie** van Microsoft (versie 2.0, 3.0 en 3.5) ondersteunt. De geïntegreerde (.NET-)talen Visual Basic .NET, Visual C# .NET, Visual C++ .NET alsmede de eigenschappen van Visual Web Developer (ontwikkeling webapplicaties) maken gebruik van dezelfde **IDE (Integrated Development Environment, geïntegreerde ontwikkelomgeving)**. Het hele pakket wordt met dezelfde IDE bediend. Naast consoletoepassingen (programma's die gebruikmaken van het tekstvenster dat ook door MS-DOS-programma's gebruikt wordt), Windows-toepassingen, webtoepassingen en servertoepassingen kan men eigen ActiveX-besturingselementen en Microsoft SQL Server-databanken ontwerpen.

2010

Visual Studio 2010 werd geïntroduceerd op 12 april 2010. Belangrijkste vernieuwingen zijn het **.NET-framework 4**, ASP.NET 4, de toevoeging van de taal F#, ondersteuning voor Microsoft Silverlight (Web- en mobiele toepassingen) en IntelliTrace. Er is ondersteuning ingebouwd voor Windows 7, Windows Phone 7 en Windows Azure. De IDE is compleet herschreven in WPF wat resulteert in een rustigere, overzichtelijkere interface. Het is uitgebracht in 5 verschillende versies; Express, Professional, Premium, Ultimate en Test Professional.

2012 e.v.

In september 2011 werd een eerste versie van Visual Studio 2012 voorgesteld, de bèta was namelijk aanwezig in de Developer Preview van **Windows 8**. De Release Candidate kwam uit op 31 mei 2012. Visual Studio 2013 verscheen op 17 oktober 2013 en op 20 juli 2015 verscheen Visual Studio 2015 en .Net 4.6. De grootste wijziging in Visual Studio 2015 is dat ontwikkelaars niet alleen apps kunnen maken voor Windows Phone, maar ook voor iOS en Android.

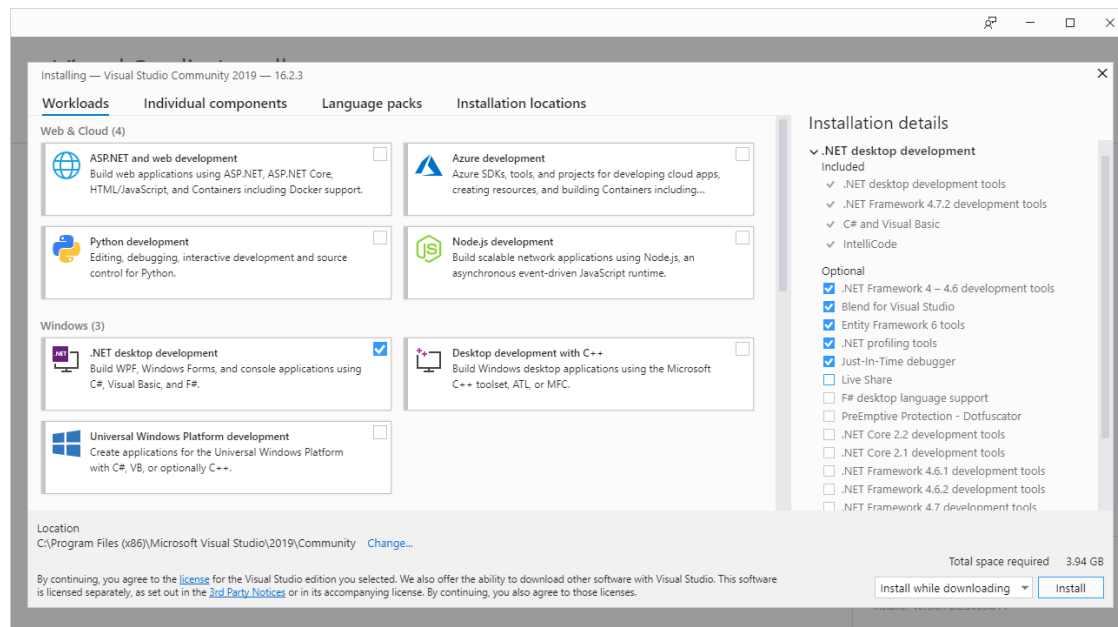
2015.

Met Visual Studio 2015 zal de Community Edition de Express-serie vervangen.

Installatie

Via de website van Visual Studio: <https://www.visualstudio.com/downloads/>
of via Azure Dev. Tools for Teaching: <https://aka.ms/devtoolsforteaching>

Selecteer Visual Studio Community 2022 en kies **.NET Desktop Development** (3,94 GB) (volstaat voor het 1^{ste} jaar). Je kan altijd later je platform uitbreiden met andere opties.



Versies

- **Community**
Visual Studio Community is een gratis, volledig functionele en uitbreidbare IDE voor individuele ontwikkelaars, open source projecten, wetenschappelijk onderzoek, onderwijs en kleine professionele teams. Maak applicaties voor Windows, Android en iOS als web applicaties en cloud-diensten.
- **Enterprise**
Visual Studio Enterprise is een geïntegreerde, end-to-end oplossing voor teams van elke omvang met veeleisende kwaliteit en omvang behoeften.
- **Professional**
Visual Studio Professional levert professionele developer tools en diensten aan individuele ontwikkelaars en kleine teams.
- **Test Professional**
Visual Studio Test Professional is ideaal voor testers, business analisten, product managers en andere belanghebbenden die team collaboration tools nodig hebben, maar niet een volledige ontwikkeling IDE.
- **Team Explorer**
Volledige versie voor ontwikkeling voor teams.

2 .NET programmeertalen

- **Visual Basic (sinds 2002)**

Visual Basic (VB) is de naam van een reeks programmeeromgevingen, later programmeertalen, uitgebracht door Microsoft. Het doel van Visual Basic is de ondersteuning van het bouwen van grafische applicaties op een visuele manier, dat wil zeggen, zo veel mogelijk via directe grafische manipulatie van elementen in plaats van het expliciet invoeren van programmacode.

- **Visual C# (# muzikale term voor een halve toon hoger) (sinds 2002)**

C# is object georiënteerd en lijkt qua syntaxis en semantiek sterk op Java. Het is een combinatie van C++ (objectgeoriënteerde taal van C), Delphi (vroegere Pascal) en Java. C# wordt beschouwd als de belangrijkste taal voor het .NET platform.

Naast *desktopapplicaties* en *serverapplicaties* (in combinatie met ASP.NET) wordt de taal ook gebruikt voor *mobiele* apparaten als tablets en smartphones, in combinatie met het .NET Compact Framework.

- **Visual F# (sinds 2005)**

F# (F sharp) is een mix van een functionele en een objectgeoriënteerde programmeertaal voor het .NET-platform van Microsoft. Omdat F# ook gebruik maakt van dezelfde CLR (Common Language Runtime) zoals C# en VB.NET, is het met F# direct mogelijk om samen te werken met Microsofts gamestudio XNA, en hiermee op erg eenvoudige wijze *games* te ontwikkelen.

- **Visual J#**

J# (uitgesproken als J sharp) is Microsofts implementatie van Suns Java. J# is in feite een soort *kopie van Java*, zodat het voor Java programmeurs eenvoudig is hun bestaande Java applicaties over te zetten naar het .NET framework.

3 De C programmeertaal

De programmeertaal C is gebaseerd op de programmeertaal B, die zelf weer op BCPL was gebaseerd. Het is een zeer praktische programmeertaal die meer op Algol (ALGOrithmic Language) lijkt dan op andere voorlopers, zoals - in historische volgorde - Fortran, Lisp, COBOL en BASIC.

Ook Pascal is een versimpeling van Algol, maar dan in een andere richting. Terwijl Pascal meer afstand neemt van de machine waar het op moet werken, ligt C juist dicht tegen de machine aan; het is betrekkelijk *'low-level'*.

De invloed van C is zo groot dat de meeste nieuwe talen zoals C++ (object georiënteerde versie van C), Objective-C, Java, JavaScript, C# en PHP grotendeels de syntaxis van C gebruiken.

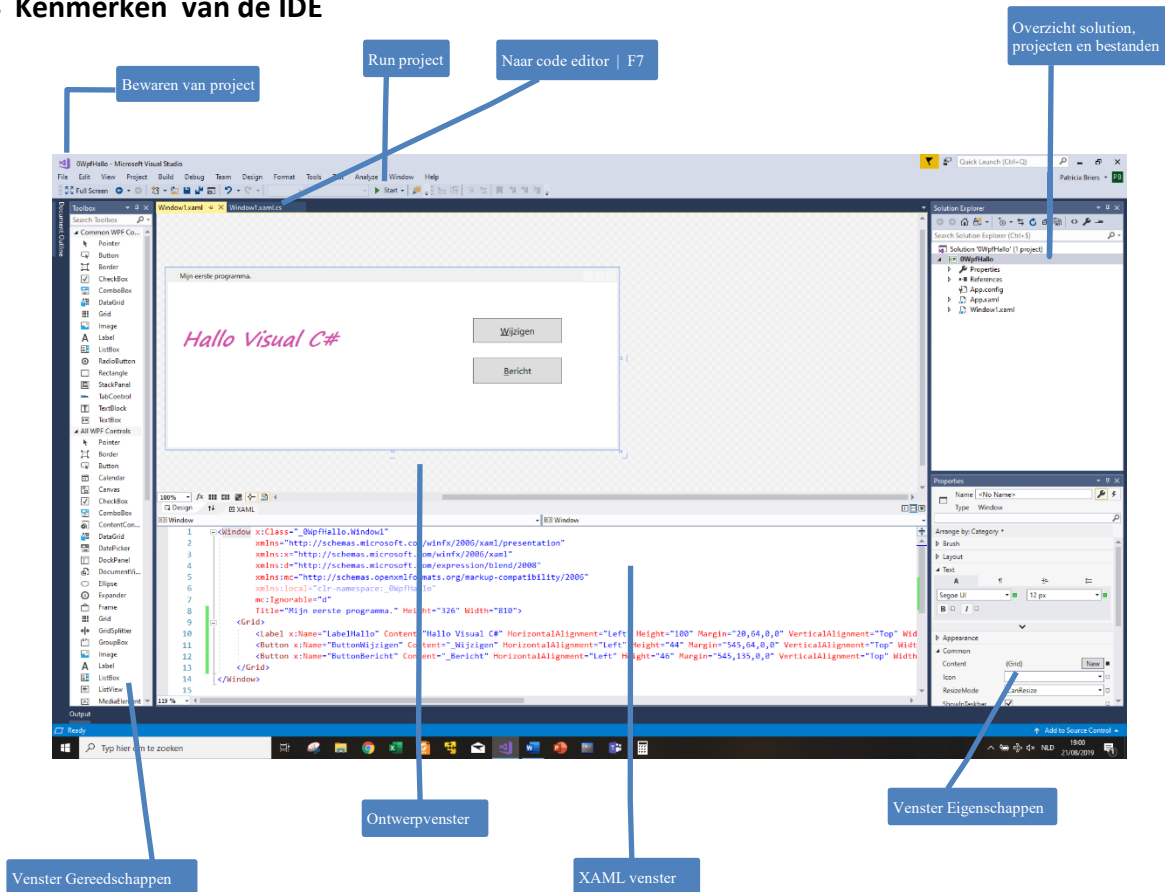
Voor- en nadelen van C

Zoals gezegd is C een taal die tamelijk dicht aansluit bij de hardware. Voordelen zijn dat C voor van *alles gebruikt* kan worden en *relatief snel* is.

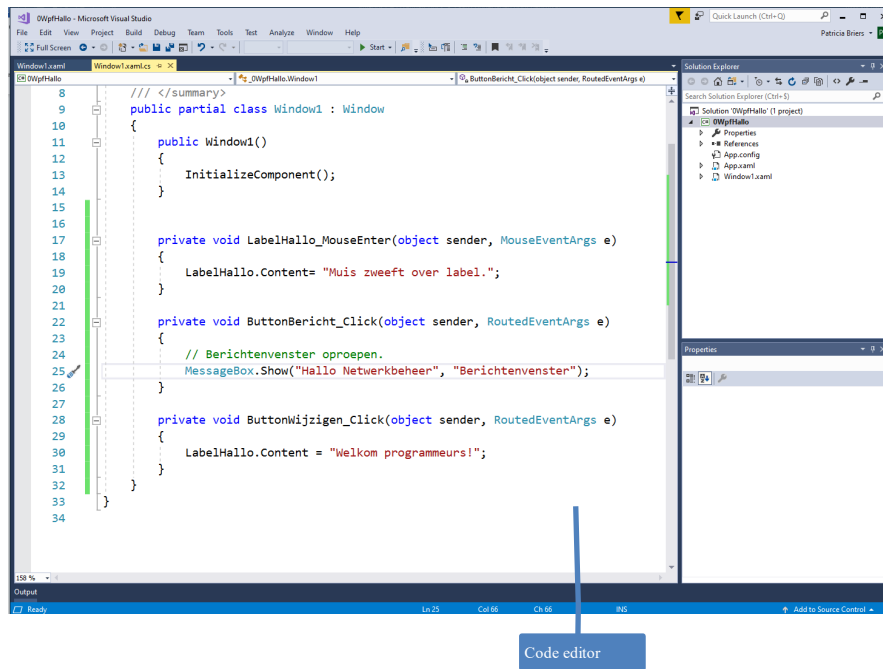
Nadelen zijn dat de taal *moeilijk* is en niet erg vergevingsgezind is met betrekking tot *fouten* en dat de C-compiler weinig controles uitvoert (hoewel die controles met diverse tools alsnog uitgevoerd kunnen worden). Een C-programma dat door de C-compiler correct wordt bevonden, hoeft niet per definitie goed te functioneren. Een oorzaak van vele problemen hier is het gebruik van *pointers*; dit is dan ook de reden waarom deze in afgeleide talen zoals Java en C# achterwege gelaten zijn.

Besturingssystemen als Unix en Windows zijn grotendeels in C geschreven. Vaak wordt er een combinatie gebruikt van C en C++, zoals bij Windows, waarbij C wordt gebruikt voor de kernel en C++ voor de overige componenten. Een ander veelgebruikt alternatief is een combinatie van C en (ingebodde) Assembler, dit komt dan terug voor bij de Linux-kernel, deze is quasi volledig in C geschreven met enkele *lowlevel* zaken in Assembler. Voor vrijwel iedere processor en microcontroller is een C-compiler beschikbaar. Daardoor kent de taal C een hoog percentage professionele programmeurs.

4 Kenmerken van de IDE



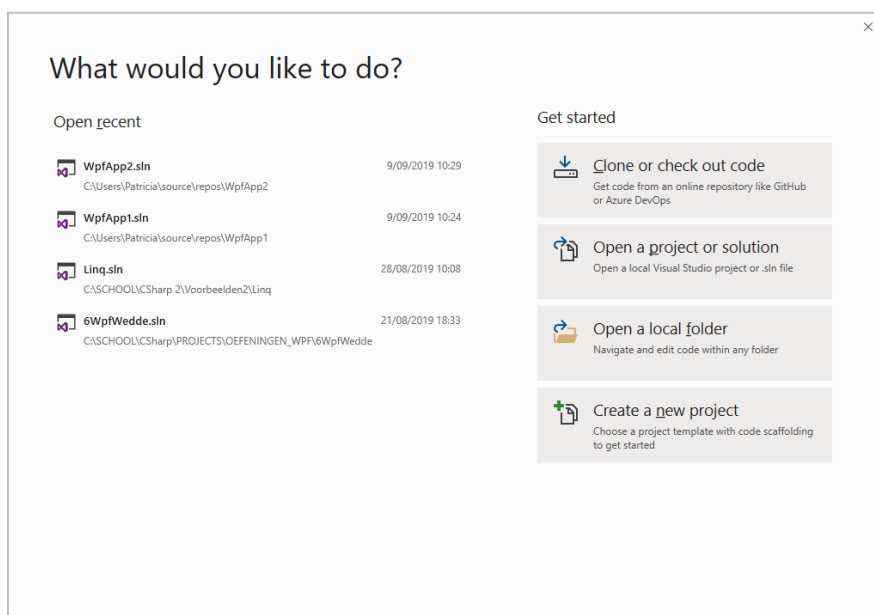
De Windows Presentation Foundation (of WPF) is het grafische subsysteem dat een onderdeel is van het .Net Framework. WPF maakt een ontwerp op basis van het nieuw programmeermodel XAML (uitgesproken als zammel – in het Engels zammel) waarmee op een nieuwe manier gebruikersinterfaces kunnen worden gebouwd. XAML (Extensible Application Markup Language) is een opmaaktaal voor de gebruikersinterface, om elementen, gebeurtenissen en andere onderdelen daarvan te definiëren.



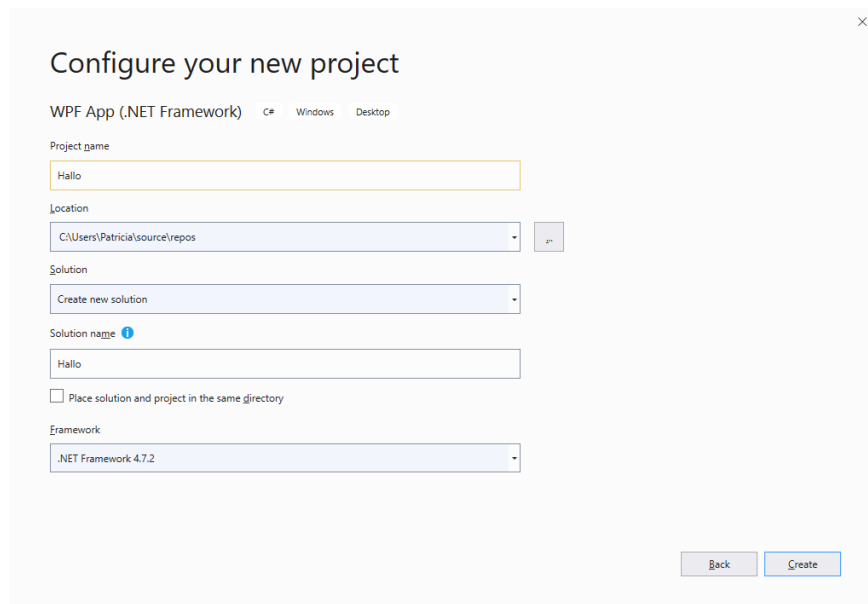
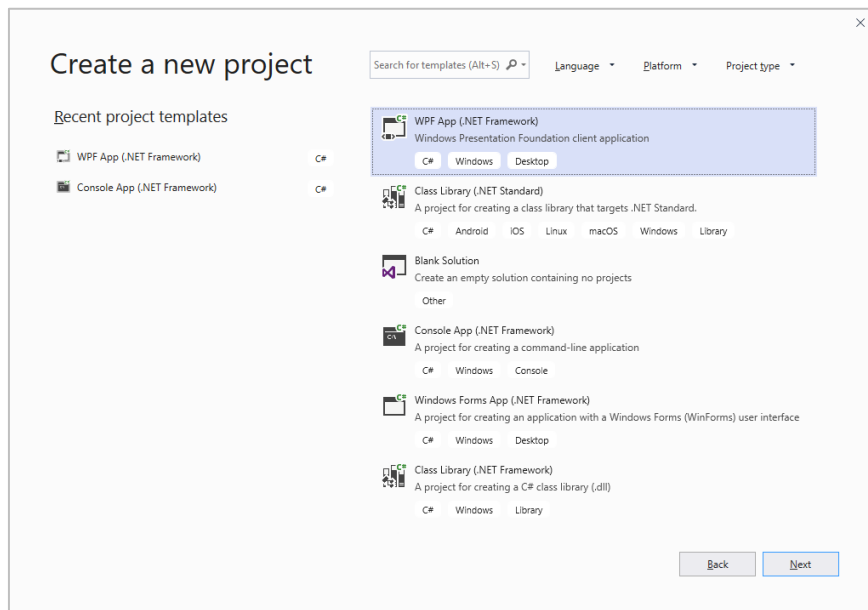
- **Solution Explorer** (rechtsboven) laat je je codebestanden bekijken, navigeren en beheren. Solution Explorer kan helpen bij het indelen van uw code door de bestanden te groeperen in Solutions en projecten.
- **Editorvenster** (midden), waar u waarschijnlijk het grootste deel van uw tijd zult doorbrengen, geeft de inhoud van het bestand weer. Hier kunt u de code bewerken of een gebruikersinterface ontwerpen, zoals een venster met knoppen en tekstvakken.

5 Maak een programma

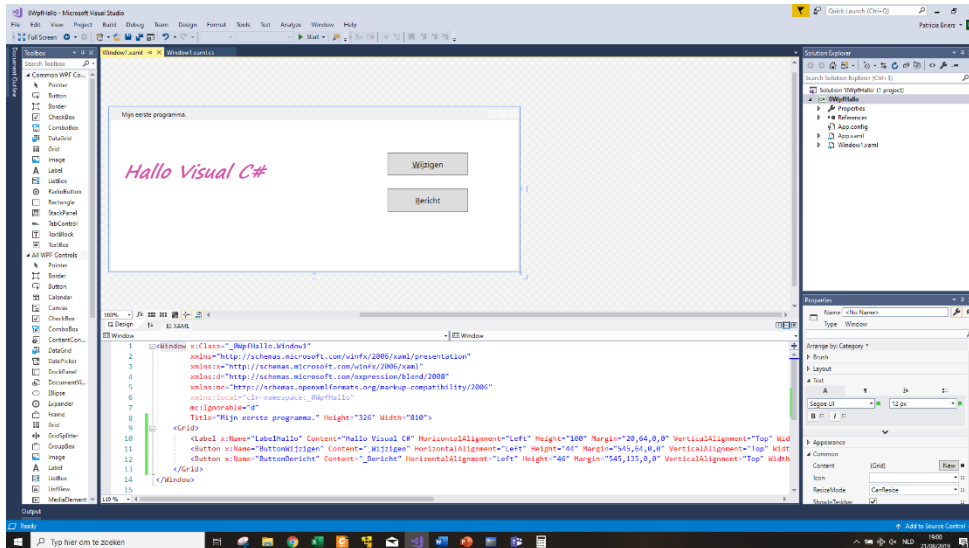
1. Open Visual Studio. Kies **New Project** (WPF - Windows Presentation Foundation App).



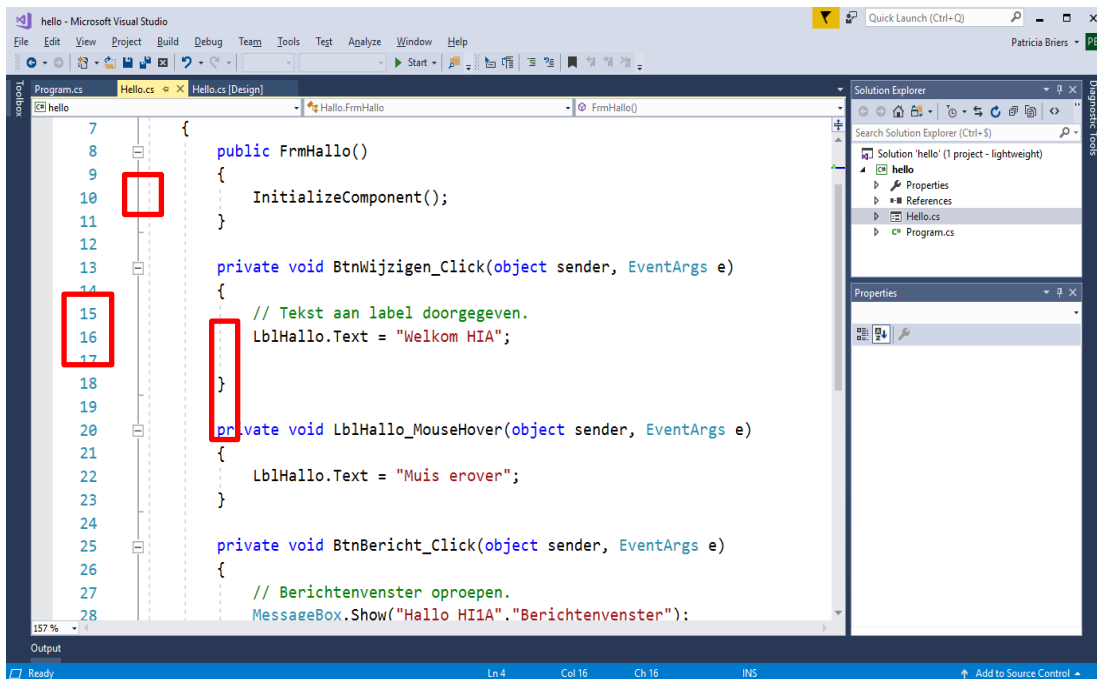
2. Het dialoogvenster **Nieuw project** toont verschillende projectsjablonen. Een sjabloon bevat de basisbestanden en instellingen die nodig zijn voor een bepaald projecttype. Kies een WPF App en typ **Hallo**.



3. Ontwerp je formulier. De verschillende besturingselementen/controls vind je terug in de Toolbox (links). Je kan de controls op verschillende manier ontwerpen:
- Selecteer een control en sleep naar je Form.
 - Klik 2x op een control en je control plaatst zich automatisch in je Form.
 - Klik op een control en druk op Ctrl en je kan de controls blijven toevoegen aan je WPF Form.

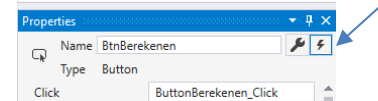


4. De C#-code voor uw toepassing wordt weergegeven in het editorvenster, dat het grootste deel van de ruimte in beslag neemt. U ziet dat de tekst automatisch wordt ingekleurd om verschillende delen van de code aan te geven, zoals zoekwoorden en typen. Bovendien geven kleine, verticale stippellijnen in de code aan welke accolades met elkaar overeenkomen, en regelnummers helpen u later de code te lokaliseren. U kunt kiezen voor de kleine min-tekenen in het vak om de codeblokken samen te vouwen of uit te breiden. Met deze functie voor het markeren van codes kunt u de code verbergen die u niet nodig heeft, zodat het rommel op het scherm tot een minimum wordt beperkt. De projectbestanden worden aan de rechterkant weergegeven in een venster met de naam Solution Explorer.



Je kan de gebeurtenis (vb. Click) op 3 verschillende manieren aan je object koppelen:

- Dubbel klik op button en je krijgt het standaardevent. Vb Click bij Button, TextChanged bij TextBox,...
- Selecteer in het Property-venster de juiste gebeurtenis.
- Typ Click in de XAML-tag van het object (XAML venster) en kies vervolgens op <New Event Handler>



5. Start nu de app. U kunt dit doen door te kiezen voor **Start Without Debugging** van de **Debug** menu op de menubalk. U kunt ook op **Ctrl + F5** drukken .

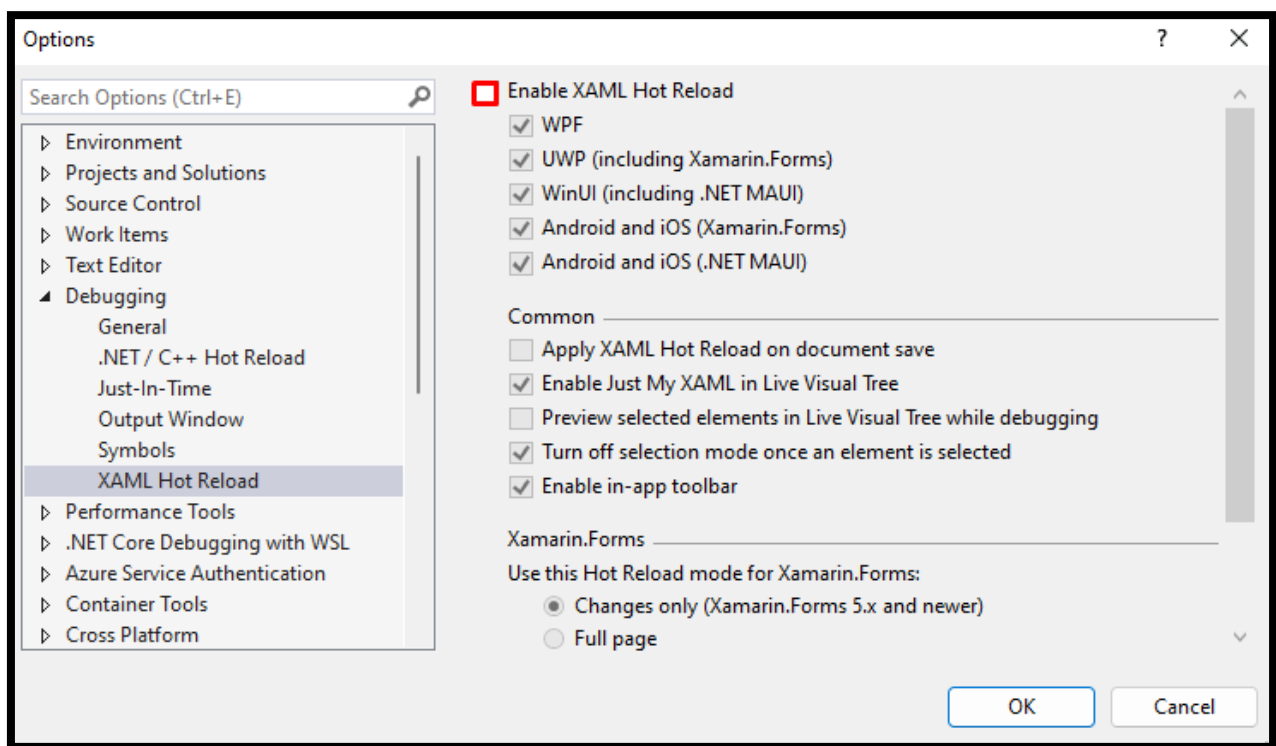
6 Visual Studio aanpassen

U kunt de gebruikersinterface van Visual Studio personaliseren, inclusief het standaardkleurenthema wijzigen. Om over te schakelen naar het thema **Dark** :

Ga naar de menubalk **Tools > Opties** en selecteer **Environment – General** en kies in **Color Theme** je voorkeur. Het kleurenthema voor de volledige IDE verandert in **Dark** .

De regelnummers in de teksteditor kan je eveneens best inschakelen. Ga naar de menubalk **Tools > Opties** en selecteer **Tekst Editor – C#** en schakel **Line Numbers** in.

Als je onderstaand Live Visual Tree hinderlijk vindt, kan je dit uitschakelen via de menu-optie Tools-Options. Vink onderstaande optie uit.



Hoofdstuk 1 Probleemoplossend denken

Het leren programmeren bestaat in de eerste plaats uit het leren een **probleem te ontleden** en er stapsgewijze een **oplossingsmethode** of algoritme op te bouwen.

Het volstaat immers niet een programmeertaal te kennen om goed te kunnen programmeren.

Programma's worden niet geschreven voor eenmalig gebruik, maar moeten een hele tijd operationeel zijn. Gedurende die tijd moeten, ten gevolge van gewijzigde omstandigheden of een vraag naar meer informatie, soms kleine of grote aanpassingen aangebracht worden. Om dat "onderhoud" van programma's, ook door iemand anders dan de programmeur die het programma oorspronkelijk maakte, gemakkelijk en vlot te kunnen laten uitvoeren, moeten de programma's **goed gedocumenteerd** en **duidelijk gestructureerd** zijn.

1 Gestructureerd denken

Met gestructureerd denken bedoelen we de systematische aanpak om via een stapsgewijze verfijning vanuit de probleemstelling een **goede oplossing** te ontwikkelen die onder vorm van één of meerdere programma's kan uitgevoerd worden. Deze aanpak vraagt het doorlopen van verschillende fasen.

fase 1 Probleemanalyse

In deze fase staat het analyseren van het eigenlijke probleem centraal:

Wat moet er gebeuren? Hoe pak ik dit aan? Wat is het probleem?

Over welke gegevens beschik ik?

Welk resultaat wordt gevraagd?

fase 2 Oplossingsstrategie

Hier proberen we een gestructureerde oplossing te formuleren van het probleem.

De basisstructuren voor het programma worden hier gemaakt.

Hoe los ik het probleem op?

Welke verfijnde technieken kan ik hiervoor gebruiken?

fase 3 Algoritme

Een algoritme is een voorschrift, opgebouwd uit één of meer stappen, dat precies aangeeft hoe men vanuit, een gegeven beginsituatie een vooraf beschreven resultaat kan bereiken.

Een algoritme moet volgende **eigenschappen** bezitten:

- 1 De opdrachten van een algoritme moeten **duidelijk, ondubbelzinnig en volledig gedefinieerd** zijn. Geen enkele stap uit het algoritme mag voor verschillende interpretaties vatbaar zijn.
- 2 Elke opdracht van een algoritme moet **vrij zijn van redundantie**, dwz ze mag niet te gedetailleerd zijn, ze moet vrij zijn van overbodige details.
- 3 Een algoritme moet vermelden **welke gegevens** nodig zijn en welke het vooropgestelde **doel** is. Men noemt het uitschrijven van deze gegevens het opstellen van de invoer- en uitvoerspecificaties van het algoritme.
- 4 Een goed algoritme moet een **structuur** hebben. Een gestructureerd algoritme is nl. goed leesbaar, eventueel gemakkelijk te wijzigen en handig bij het opsporen van fouten.
- 5 Een algoritme moet eindigen.

Er bestaan verschillende schematische voorstellingen voor dit programmeeralgoritme.

Vb. voorstellingswijze: **pseudo-code**

fase 4 : Programma

Na de schematische voorstelling wordt het eigenlijke programma geschreven in C#, Visual Basic, Visual Basic for Applications, C++, Java, ...

fase 5 : Testen van het programma

Door middel van testgegevens wordt het eigenlijke programma getest of het gewenste resultaat bereikt wordt.

fase 5 : Documentatie

Dit is vooral van nut om het programma later te kunnen aanpassen. In de **documentatiemap** zitten o.a. de documenten m.b.t. de probleemanalyse en de schema's die werden getekend en uitgewerkt. Meestal moet ook een gebruikershandleiding geschreven worden. Je programmeercode moet ook veelvuldig voorzien zijn van **commentaarregels**!

Voorbeeld

Fase 1 Probleemanalyse

Gegeven: /

Gevraagd: ☐ Wijzig tekst in "Welkom programmeurs"

- Berichtenvenster "Hallo netwerkbeheer"
- Wijzig tekst in "Muis zweeft over label"

Fase 2 Oplossingsstrategie

- Klik op 'Wijzigen': druk tekst "Welkom programmeurs" in label
- Klik op 'Bericht': berichtenvenster "Hallo netwerkbeheer"
- Muis over label: druk tekst "Muis zweeft over label" in label

Fase 3 Pseudocode

Begin *BtnWijzigen_*

druk "Welkom programmeurs" (*LblHallo*)

Einde

Begin *BtnBericht_Click*

druk "Hallo netwerkbeheer" (*berichtenvenster*)

Einde

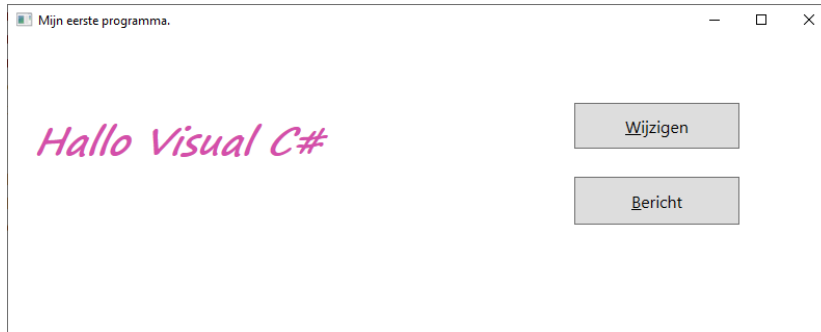
Begin *LblTekst_MouseEnter*

druk "Muis zweeft over label" (*LblHallo*)

Einde

Fase 4 Programmeren

4.1 Ontwerpfase/ Visual programming



4.2 Programmeerfase / Code programming

```
private void ButtonWijzigen_Click(object sender, RoutedEventArgs e)
{
    LabelHallo.Content = "Welkom programmeurs!";
}
```

```
private void LabelHallo_MouseEnter(object sender, MouseEventArgs e)
{
    LabelHallo.Content= "Muis zweeft over label.";
}
```

```
private void ButtonBericht_Click(object sender, RoutedEventArgs e)
{
    // Berichtenvenster oproepen.
    MessageBox.Show("Hallo 1PRO C/D", "Berichtenvenster");
}
```

Fase 5 Testen

Gebruik testgegevens om fouten op te sporen.

Fase 6 Documentatie

Ontwerp, probleemanalyse, algoritme, helpfuncties,... alle informatie over de toepassing.

2 Het gestructureerd schema

Het leren programmeren bestaat in de eerste plaats uit het leren een **probleem te ontleden** en stapsgewijze een **oplossingsmethode** of algoritme hier rond op te bouwen.

Het volstaat immers niet een programmeertaal te kennen om goed te kunnen programmeren.

Programma's worden niet geschreven voor eenmalig gebruik, maar moeten een hele tijd operationeel zijn. Gedurende die tijd moeten, ten gevolge van gewijzigde omstandigheden of een vraag naar meer informatie, soms kleine of grote aanpassingen aangebracht worden. Om dat "onderhoud" van programma's, ook door iemand anders dan de programmeur die het programma oorspronkelijk maakte, gemakkelijk en vlot te kunnen laten uitvoeren, moeten de programma's **goed gedocumenteerd** en **duidelijk gestructureerd** zijn.

2.1 Bepaling

Een gestructureerd schema is een **gedetailleerde voorstelling van de opeenvolging van operaties** die naar de oplossing van een gesteld probleem leiden, gebruik makend van structuurelementen die slechts één ingang en één uitgang hebben.

2.2 Doel

Gestructureerde schema's worden opgesteld om het **verloop van de redenering** die moet gevolgd worden ten einde een bepaald probleem op te lossen, op een duidelijke en ondubbelzinnige manier weer te geven. Zij leggen een band tussen de probleemdefinitie en het uiteindelijke programma.

Tot de belangrijkste kenmerken van systematische oplossingsmethodes kunnen we rekenen:

- Het op te lossen probleem wordt in **kleine delen gesplitst**, waardoor het mogelijk wordt in teamverband een oplossing uit te werken. Dit vraagt duidelijke omschrijvingen van de gebruikte structuren en modules.
- Men maakt gebruik van een reeks **structuren** die op een goed afgebakende manier gedefinieerd worden.
- Een probleem wordt **losgekoppeld** van een **programmeertaal** zelf.

3 De pseudo-code

De "pseudo-code" is een **fictieve programmeertaal**, die toelaat de uit te voeren opdrachten in eigen taal te formuleren. Dit biedt volgende voordelen:

- Men kan zich beter concentreren op de oplossing van het eigenlijke probleem zelf. Er moet immers geen rekening worden gehouden met de syntaxregels van een bepaalde programmeertaal.
- Het programma is beter leesbaar, waardoor de communicatie met niet-informatici zoals bij voorbeeld de directie van een onderneming gemakkelijker wordt.

4 Basisprogrammastructuren

4.1 Sequentie

S1

S2

S3

S4

.....

```
Begin
  lees F1
  lees F2
  lees F3
  SOM = F1 + F2 + F3
  druk SOM
Einde
```

4.2 Selectie

als-dan-anders

als V

dan S1

anders S2

eindals

```
Begin
  lees L
  als L < 16
    dan lees CODE
      als CODE = 1
        dan druk "Toestemming met ouders"
        anders druk "Geen toestemming"
      eindals
    eindals
  eindals
Einde
```

ingeval

ingeval l is

1: S1

2: S2

...

n: Sn

eindgeval

ingeval getal is

a: HG=10

b: HG=11

c: HG=12

eindgeval

4.3 Iteratie

Zolang-doe

zolang V

doe S

eindzolang

zolang (BK <= EK)

doe BK = BK * (1 + R/100)

JR = JR + 1

druk BK

eindzolang

Voor-doe

voor l van BW tot EW [stap]

doe S

einddoe

Begin

voor tel van 1 tot 10

doe lees F1, F2, F3

druk F1+F2+F3

einddoe

Einde

Hoofdstuk 2 Operatoren

1 Rekenkundige operatoren

1.1 Binaire operatoren (werken met 2 operators)

Operator	Betekenis	Voorbeeld
+ en -	getal positief of negatief maken	-3
* en /	vermenigvuldiging of deling	5/3=1,66 of 5*3=15
%	modulo of restdeling	5 % 3= 2
+ en -	optellen en aftrekken	5+3=8

Volgorde van bewerking: $() * / \% + -$

Vb.

$$13 - 2 * 6 - 3 / 3 = 0$$

$$(13 - 2) * 6 - 3 / 3 = 65$$

$$(((13 - 2) * 6 - 3) / 3 = 21$$

1.2 Unaire operatoren (werken met 1 operator)

Operator	Betekenis	Voorbeeld
++	verhoging met 1	$v1++ \quad \sim \quad v1 = v1 + 1$
		$v1=++v2 \quad \sim \quad v1 = (v2 + 1)$
		$v1 = v2++ \quad \sim \quad v1 = v2 \text{ en } v2 = v2 + 1$
--	vermindering met 1	$v1-- \quad \sim \quad v1 = v1 - 1$
		$v1=--v2 \quad \sim \quad v1 = (v2 - 1)$
		$v1 = v2-- \quad \sim \quad v1 = v2 \text{ en } v2 = v2 - 1$

2 Tekstoperator

Tekstoperator	Betekenis	Voorbeeld
+	aaneenschakeling van strings	"5" + "15" = "515"

3 Toewijzingsoperatoren

Toewijzingsoperator	Voorbeeld
=	$v1=10 \quad v1 = 10$
+=	$v1+=10 \quad v1=v1+10$
-=	$v1-=v2 \quad v1=v1-v2$
=	$v1=v2 \quad v1=v1*v2$
/=	$v1/=v2 \quad v1=v1/v2$
%=	$v1\%=v2 \quad v1=v1\%v2$

4 Relationale operatoren of vergelijkingsoperatoren

Vergelijkingsoperator	Betekenis
==	Is gelijk aan
!=	Is niet gelijk aan
>	Is groter dan
<	Is kleiner dan
>=	Is groter dan of gelijk aan
<=	Is kleiner dan of gelijk aan

5 Logische operatoren / Bitgewijze werking

Logische operator	Betekenis
&	Als beide voorwaardelijke expressies True zijn, is het resultaat True.
	Als minstens één voorwaardelijke expressie True is, is het resultaat True. (Inclusive Or)
!	Als de voorwaardelijk expressie False is, is het resultaat True. Als de voorwaardelijk expressie True is, is het resultaat False.
^	Als één van de voorwaardelijke expressies True is, is het resultaat True. Als beide voorwaardelijke expressies True zijn of als beide voorwaardelijke expressies False zijn, is het resultaat False. (Logische Xor of Exclusive Or)

Voorwaarde 1	Voorwaarde 2	And (&)	Or ()	Xor (^)
True	True	True	True	False
False	True	False	True	True
True	False	False	True	True
False	False	False	False	False

Conditionele AND en OR

&&	Werkt zoals de &, maar als de eerste voorwaarde False is, dan wordt de vergelijking afgebroken en levert False af.
	Werkt zoals de -operator, maar als de eerste voorwaarde True is, dan wordt de vergelijking afgebroken en levert True op.

Voorwaarde 1	Voorwaarde 2	AndAlso (&&)	OrElse ()
True	True	True	Kortsluiten /True
False	True	Kortsluiten / False	True
True	False	False	Kortsluiten /True
False	False	Kortsluiten / False	False

De conditionele AND en OR worden door de geringe performance enkel gebruikt in toepassingen waarin ze veel voorkomen.

6 Evaluatieprioriteit

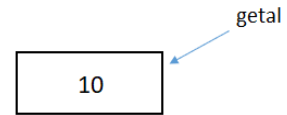
Categorie (van hoogste naar laagste prioriteit)	Operator
Rekenkundige operatoren	- (negatie), ++, -- (gebruikt als voorvoegsel) *, /, % +, -
Vergelijkingsoperatoren	>, <, <>, <=, >= ==, !=
Logische operatoren	! & & ^ &&
Toewijzingsoperatoren	=, *=, /=, %=, +=, -= ++, -- (gebruikt als achtervoegsel)

Hoofdstuk 3 Informatie opslaan in variabelen en constanten

Om efficiënt te kunnen programmeren, moet u informatie tijdelijk kunnen opslaan.

1 Wat zijn variabelen?

Een variabele is een tijdelijke opslaglocatie voor gegevens. Een **variabele verwijst naar een geheugenblok** dat informatie opslaat. De informatie die is opgeslagen in een variabele is van een bepaald gegevenstype. Het .NET Framework gebruikt het concept van gegevenstypen die voldoen aan de eisen van de Common Language Specification (CLS). Dus elke '.Net compiler' kent deze gegevenstypen.



2 Variabelen benoemen en toekennen

Elke variabele moet gedeclareerd en toegekend worden vooraleer dat je de variabele kan gebruiken in een toepassing.

- Variabelen declareren

Vb. `int getal;`

`string` voornaam;

`double` getal1, getal2; (2 variabelen gedeclareerd van het type double)

Het declareren van een variabele betekent dat u geheugenruimte reserveert voor informatie van een bepaald datatype (aangegeven door `int`, `single`, `string`,...).

- Variabelen toekennen

Vb. `string` naam = "Patricia Briers"; (declaratie met toekenning van waarde)

`int` getal1 = 4, getal2 = 5; (beide variabelen van het datatype integer)

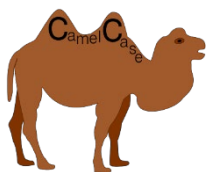
`int` getal1, getal2 = 5; (getal1 en getal2 zijn als integer gedeclareerd en krijgen respectievelijk de waarde 0 en waarde 5)

Naamgeving van variabelen:

- De naam kan met een letter, `_` of `@` beginnen, niet met een cijfer of ander teken.
- De rest van de naam mag uit letters, cijfers en `_` bestaan (geen spatie, punt en puntkomma).
- De naam mag niet langer dan 255 tekens zijn.
- Binnen het bereik van de variabele moet de naam uniek zijn.
- De naam mag niet een van de gereserveerde woorden van Visual C# zijn.
- C# is **hoofdlettergevoelig**: `voornaam`, `VOORNAAM`, `Voornaam`,... zijn 3 verschillende identifiers!!

Naamconventies

We hanteren de twee soorten conventies in het .NET Framework: *camelCasing* en *PascalCasing*



De naam `camelCase` verwijst, naar de hoofdletters van elk woord (middenin) in het samengestelde woord, naar de bulten van een kameel. We gebruiken het bij de naamgeving van variabelen en functieparameters.

Vb. `naamCursist`, `rekeningBank`, `studNaamPerAfdeling`,....

De `UppercamelCasing` of `Pascal Casing` wordt gebruikt voor overige benamingen. Vb. `Microsoft`, `BlackBerry`,....

3 Datatypes voor variabelen

Alias	.NET type	Doelgroep	Geheugenruimte	Waardenbereik
sbyte	System.SByte	Gehele getallen	1 byte	-128 tot 127
byte	System.Byte	Gehele getallen	1 byte	0 tot 255
short	System.Int16	Gehele getallen	2 bytes	-32 768 tot 32 767
ushort	System.UInt16	Gehele getallen	2 bytes	0 tot 65 535
int	System.Int32	Gehele getallen	4 bytes	- 2 147 483 648 tot 2 147 483 647
uint	System.UInt32	Gehele getallen	4 bytes	0 tot 4 294 967 295
long	System.Int64	Gehele getallen	8 bytes	-9 223 372 036 854 775 808 tot 9 223 372 036 854 775 807
ulong	System.UInt64	Gehele getallen	8 bytes	0 tot 18446744073709551615
float	System.Single	Decimale getallen	4 bytes	±1.401298E- 45 tot 3.402823E38
double	System.Double	Decimale getallen	8 bytes	±1.79769313486231E+308 tot 4.940656458424E-324
decimal	System.Decimal	Decimale getallen	16 bytes	+/- 79 228 x 1024
char	System.Char	Enkel karakter	2 bytes	Unicode karakter tussen 0 tot 65 535
string	System.String	Tekstinformatie	één byte per teken	Tot +/- 2 miljard Unicode karakters
bool	System.Boolean	Logische waarde	2 byte	True of False

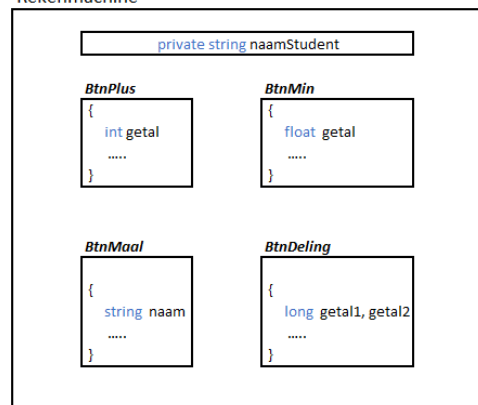
Voor de elementaire datatypes *Integer* en *Float* worden voor getallen gebruikt en voor alfanumerieke gegevens wordt *String* gebruikt. Vermits *Char* de karakters als getallen bewaart, behoort *Char* tot het numerieke datatype.

4 Bereik van variabelen

De toegankelijkheid van de variabelen binnen de applicatie heet *bereik* of *scope*. Afhankelijk van het gebied is de variabele toegankelijk in bepaalde delen van je applicatie. Daarbuiten heeft deze variabele géén invloed meer en kan dus niet meer geraadpleegd worden.

De variabelen binnen een procedure, for, switch of using gedeclareerd noemen we de **lokale** of **plaatselijke** variabelen. Ze zijn enkel bereikbaar binnen de structuur waarin ze gedeclareerd zijn.

Rekenmachine



Modulevariabelen of **klassevariabelen** zijn bereikbaar binnen alle procedures van een klasse. Gebruik *private* om je klassevariabele te declareren. Vb. `private string naam;`

Het gebruik van **globale** of **publieke** variabelen bestaat niet in C#. Je kan wel een publieke klasse aanmaken en daarin met statische variabelen werken. Zie verder hoofdstuk 'Werken met meerdere formulieren'.

```
public class Var
{
    public static int getal = 10;
}

...

Var.getal = 20;
```

5 Numerieke waarden

Wanneer een variabele een waarde toegekend krijgt, wordt er vooreerst gekeken of het een geheel getal is of een decimal getal. Bij een geheel getal wordt het kleinste datatype toegekend (*int*, *uint*, *long* of *ulong*). Bij een decimaal getal wordt het datatype automatisch *double*. Wens je een ander datatype aan een variabele van een ander datatype toe te kennen, gebruikt dan een achtervoegsel (suffix) om duidelijk te maken welke datatype je bedoelt.

Voorbeeld

```
float eenheidsprijs = 123.45;    // werkt niet
float eenheidsprijs = 123.45F;   // werkt
```

Type	Suffix	Example
uint	U or u	100U
long	L or l	100L
ulong	UL or ul	100UL
float	F or f	123.45F
decimal	M or m	123.45M

6 Alfnumerieke waarden

We onderscheiden 2 soorten alfanumerieke waarden. De **gewone tekenreeks** bestaat uit nul of meer tekens tussen dubbele aanhalingstekens, zoals in "module programmeren in c#", en kunnen zowel eenvoudige als speciale karakters/stuurcodes of escape sequences (zoals \t voor het tabblad karakter) en hexadecimale en Unicode-karakters bevatten.

Een **verbatim tekenreeks** 'letterlijke tekenreeks' bestaat uit een @ teken gevolgd door een dubbele aanhalingsteken. Een eenvoudig voorbeeld is @"hello". In een verbatim tekenreeks, worden de tekens tussen de scheidingstekens letterlijk geïnterpreteerd, met als enige uitzondering de dubbele aanhalingstekens (" "). Een woordelijke letterlijke tekenreeks kan over meerdere lijnen.

Escape Sequence	What it Represents
\'	Single Quotation Mark (character 39)
\"	Double Quotation Mark (character 34)
\?	Literal Question Mark
\\	Backslash (character 92)
\a	Alert/Bell (character 7)
\b	Backspace (character 8)
\f	Formfeed (character 12)
\n	New Line (character 10)
\r	Carriage Return (character 13)
\t	Horizontal Tab (character 9)
\v	Vertical Tab (character 11)
\ooo	ASCII character in octal notation
\xhh	ASCII character in hexadecimal notation

Voorbeelden

<code>string a = "hallo, C #";</code>	hallo, C #	--- gewone tekenreeks
<code>string b = @"hallo, C #";</code>	hallo, C #	--- verbatim tekenreeks
<code>string c = "hallo \t C #";</code>	hallo C #	--- gewone tekenreeks
<code>string d = @"hallo \t C #";</code>	hallo \t C #	--- verbatim tekenreeks
<code>string e = "Paul is \"lector\" in klas B";</code>	Paul is "lector" in klas B	--- gewone tekenreeks
<code>string f = @"Paul is ""lector"" in klas B";</code>	Paul is "lector" in klas B	--- verbatim tekenreeks
<code>string g = "\\server\\share\\file.txt";</code>	\\server\\share\\file.Txt	--- gewone tekenreeks
<code>string h = @"\\server\\share\\file.txt";</code>	\\server\\share\\file.Txt	--- verbatim tekenreeks
<code>string i = "Visual\r\nStudio\r\n2017";</code>	Visual	--- gewone tekenreeks
<code>string j = @"Visual</code>	Studio	--- verbatim tekenreeks
<code>Studio</code>	2017	over meerdere regels
<code>2017";</code>		

7 Constanten

Met een constante kan je een beschrijvende naam gebruiken voor de waarde van een variabele (read-only) die niet verandert tijdens de uitvoering van uw project. De constante wordt gedeclareerd met het keyword `const`.

Vb. `const float pi = 3.14159265f;`
`const string naam = "Patricia Briers";`
`private const int months = 12;`
`const int maanden = 12, weken = 52, dagen = 365;`
`const double dagenPerWeek = dagen / weken;`

Hoofdstuk 4 Conversies

4.1 Numerieke conversies

Visual C# kent allerlei *manieren* om waarden van het ene type om te zetten in waarden van een ander type. We kunnen de conversies in twee hoofdcategorieën splitsen in *verbredende* conversies en *versmallende* conversies. Verbredende conversies zijn conversies waarin gegevenverlies of onjuiste resultaten onmogelijk zijn. Vb. een conversie van een waarde van het type *Integer* naar een waarde van het type *Long* is een verbredende conversie, terwijl het omgekeerde (van type *Long* naar type *Integer*) een versmallende conversie is.

Visual C# voert verbredende conversies automatisch uit en wordt ook *impliciete* conversie genoemd. Om geen gegevensverlies bij versmallende conversie te hebben gebruiken we de *expliciete* conversie – dus concreet aangeven naar welke datatype moet geconverteerd worden - met behulp van de conversiefuncties.

Voorbeeld

```
short  shortResult, shortVal = 4;
int    integerVal = 67;
long   longResult;
float  floatVal = 10.5F;
double doubleResult, doubleVal = 99.999;
string stringResult, stringVal = "17";
bool   boolVal = true;

doubleResult = floatVal * shortVal; // impliciet: 10.5 * 4 = 42 (berekening in double)
shortResult = (short)floatVal;      // expliciete casting: 10
longResult = integerVal + Convert.ToInt64(stringVal); // mix: 67 + 17 = 84

stringResult = Convert.ToString(boolVal) + Convert.ToString(doubleVal); // string: True99.999
```

Expliciete numerieke conversietabel dmv casting

Van	Naar
sbyte	byte , ushort, uint, ulong, or char
byte	sbyte or char
short	sbyte , byte, ushort, uint, ulong, or char
ushort	sbyte , byte, short, or char
int	sbyte , byte, short, ushort, uint, ulong, or char
uint	sbyte , byte, short, ushort, int, or char
long	sbyte , byte, short, ushort, int, uint, ulong, or char
ulong	sbyte , byte, short, ushort, int, uint, long, or char
char	sbyte , byte, or short
float	sbyte , byte, short, ushort, int, uint, long, ulong, char, or decimal
double	sbyte , byte, short, ushort, int, uint, long, ulong, char, float, or decimal
decimal	sbyte , byte, short, ushort, int, uint, long, ulong, char, float, or double

Expliciete numerieke conversietabel met System.Convert klasse

Conversiefuncties	Converteert het argument naar
<code>Convert.ToBoolean(Val)</code>	boolean
<code>Convert.ToByte(Val)</code>	byte
<code>Convert.ToChar(Val)</code>	char
<code>Convert.ToDecimal(Val)</code>	decimal
<code>Convert.ToDouble(Val)</code>	double
<code>Convert.ToInt16(Val)</code>	short
<code>Convert.ToInt32(Val)</code>	integer
<code>Convert.ToInt64(Val)</code>	long
<code>Convert.ToSByte(Val)</code>	sbyte
<code>Convert.ToSingle(Val)</code>	single
<code>Convert.ToString(Val)</code>	string
<code>Convert.ToUInt16(Val)</code>	ushort
<code>Convert.ToUInt32(Val)</code>	uint
<code>Convert.ToUInt64(Val)</code>	ulong

Let op

`Convert.ToInt32(4.5)` geeft 4 omdat er naar het dichtstbijzijnde even getal afgerond wordt terwijl (int) 4.5 = 4 omdat na de komma afgekap wordt.

4.2 Formattering van getallen (opmaak)



In C# 6.0 werd de *string interpolation* voorgesteld. De oude samengestelde format werd stevig vereenvoudigd. Deze techniek converteert een string naar een numerieke waarde en bovendien wordt er een opmaak doorgevoerd.

- `string.Format("Geheel getal: {0} en floating point: {1}", 123456, 123.456)`
wordt: `$"Geheel getal: {123456} en floating point: {123.456}"`
- `string.Format("Macht {0:D2} van {1} is {2:n0}\n", getal1, getal2, macht);`
wordt: `$"Macht {getal1:D2} van {getal2} is {macht:n0}\n";`

Voorbeelden

```
Label1.Text = $"Geheel getal: {123456} en floating point: {123.456}"; // Geheel getal: 123456 en floating point: 123,456
Label1.Text = $"{123:D5}"; // 00123
Label1.Text = $"{123456:N}"; // 123 456,00
```

// Voorbeeld getallen uitgelijnd wegschrijven

```
string a = $"{1,5}|{20,5}|{300,5}|";           "|    1|   20|  300|"
string b = $"{1,-5}|{20,-5}|{300,-5}|";         "|1    |20   |300  |"
string b = $"{1,-5:d3}|{20,-5:d3}|{300,-5:d3}|"; "|001 |020 |300 |"
```

//Afdruk naar tekstvak

```
TxtResultaat.Text = $"LOONFICHE VAN {naam}\r\nBrutojaarwedde : {bruto:C}";
```

```
LOONFICHE VAN Ellen De Cooman
Brutojaarwedde : € 30 095,10
```

Karakter	Omschrijving	Voorbeeld	Resultaat
C or c	Currency	<code>(\$"{2.5:C}";</code> <code>(\$"{-2.5:C}";</code>	€ 2,50 € -2,50
D or d	Decimal	<code>(\$"{25:D5}";</code>	00025
E or e	Scientific	<code>(\$"{250000:E}";</code>	2,500000E+005
F or f	Fixed-point	<code>(\$"{25:F2}";</code> <code>(\$"{25:F0}";</code>	25,00 25
G or g	General	<code>(\$"{2458.512:G}";</code>	2458,512
N or n	Number	<code>(\$"{2500000:N}";</code>	2 500 000,00
P or p	Percent	<code>(\$"{2.5:P}";</code>	250,00%
X or x	Hexadecimal	<code>(\$"{250:X}";</code>	FA

Je kan ook zelf een format opmaken. Vb. `Label1.Text = $"{3100.5: #,##0.00}";` // 3 100,50 :

Format	Getal	Weergave
0000.00	100.5	0100.50
0	100.5	101
#,##0	3100.5	3 101
#,##0.00	3100.5	3 100,50
# ##0.00 \E\U\R	3100.5	3 100,50 EUR
0.00%	0.253	25,30%
yyyy-MM-dd	27/11/2021	2021-11-27
MMM, yy	27/11/2021	dec, 21
MMMM, hh:mm:ss zz	nov, 11:33:16	november, 11:33:16 +01 (wintertijd)

(kleine letters mm: minuten | hoofdletters MM = maanden)

4.3 String conversie

Je kan een string converteren naar een numerieke waarde dmv de klasse `String` of door rechtstreeks gebruik te maken van de method `Parse/TryParse`.

Voorbeeld

```
TxtGetal.Text = "120";

int i = int.Parse(TxtGetal.Text);           // geeft 120
MessageBox.Show(i.ToString());

float i = float.Parse(TxtGetal.Text) / 13; // geeft 9,230769
MessageBox.Show(i.ToString());
```

Parse

Parsing is het ontleden van uitdrukking. Het omzetten van string waarden naar numerieke waarden doe je door gebruik te maken van de methode **Parse** die gekoppeld is aan een numeriek gegevenstype.

Conversiefuncties	Parse method
<code>Convert.ToDecimal(Val)</code>	<code>decimal.Parse</code>
<code>Convert.ToDouble(Val)</code>	<code>double.Parse</code>
<code>Convert.ToInt32(Val)</code>	<code>int.Parse</code>
<code>Convert.ToInt64(Val)</code>	<code>long.Parse</code>
<code>Convert.ToSingle(Val)</code>	<code>float.Parse</code> ... (zoals bij de <code>Convert</code>)

TryParse

De `TryParse` geeft *True* weer als de conversie slaagt en *False* als de conversie mislukt. Hierdoor kan je testen of je te maken hebt met 'verdachte' gegevens (geen alfanumerieke getallen of getallen die groter zijn dan het maximale getal dat een datatype kan bevatten – overflow).

```
int cijfer
bool resultaat = int.TryParse(TxtGetal.Text, out cijfer);

of

bool resultaat = int.TryParse(TxtGetal.Text, out int cijfer);
```

Convert.String klasse

De klasse `String` gebruikt intern de `Parse` methode maar geeft geen foutmelding (maar 0 terug) wanneer de string `NULL` is. In tegenstelling tot de `Parse` methode die bij `NULL` een `ArgumentNullException` geeft). Je kan deze klasse in alle omstandigheden gebruiken.

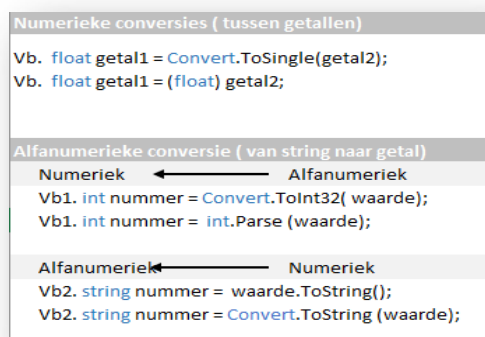
```
TxtGetal.Text = Convert.ToString(100); // geeft "100"
TxtGetal.Text = Convert.ToString(100.234f); // geeft "100,234"
```

ToString() methode

Merk ook op dat in het voorbeeld de `.ToString` methode wordt gebruikt ipv `Convert.String` klasse. Ook hier geeft de `ToString()`-methode bij corrupte gegevens meer foutmeldingen dan de `String`-klasse en is de klasse veiliger, maar de methode is gemakkelijker in gebruik. Bovendien is een methode altijd vlugger dan een klasse.

```
int getal = 10;
string s1 = getal.ToString();
string s2 = Convert.ToString(getal);
```

Besluit



Hoofdstuk 5 Controlestructuren

1 Selectie

1.1 If-opdracht

```
if (teller == 10)
{
    TxtResultaat.Text = "Ok";
}
```

De accolades mogen weggelaten worden als er 1 statement volgt.

```
if (teller == 10) TxtResultaat.text = "Ok";
```

1.2 If-else-opdracht

```
if (teller == 10)
{
    TxtResultaat.Text = "Ok";
    Console.WriteLine("Teller is Ok");
}
else
{
    TxtResultaat.Text = "verkeerd";
    Console.WriteLine("Teller is verkeerd");
}
```

```
if voorwaarde
{
    Instructies
}
[else
{
    Instructies
}]
```

Tip

De snippet is heel gemakkelijk in gebruik. Gebruik het veelvuldig!

```
if
Code snippet for if statement
Note: Tab twice to insert the 'if' snippet
```

1.3 Geneste Ifs

```
if (teller == 10)
{
    TxtResultaat.Text = "10";
}
else if (teller == 20)
{
    TxtResultaat.Text = "20";
}
else
{
    TxtResultaat.Text = "overige waarde";
}
```

```
if (aantal <= 1000)
{ prijs = 200; }
else if (aantal <= 2000)
{ prijs = 250; }
else if (aantal <= 5000)
{ prijs = 450; }
else if (aantal <= 10000)
{ prijs = 500; }
else
{ prijs = 750; }
```

	< 1 000
1 001 -	2 000
2 001 -	5 000
5 001 -	10 000
> 10 000	

```
If voorwaarde
{
    [instructies]
}
else if voorwaarde
{
    [instructies]
}
else
{
    [instructies]
}
```

1.4 If Ternary operator

```
string resultaat = (cijfer >= 6) ? "voldoende" : "onvoldoende";
string resultaat = ((cijfer < 5) ? "onvoldoende" : ((cijfer > 7) ? "goed" : "voldoende"));
```

```
voorwaarde ? resultaatBijWaar : resultaatBijNietWaar;
```

1.5 Meervoudige selectie

```
switch (land)
{
    case "au":
    case "uk":
    case "us":
        taal = "English";
        break;
    case "be-noord":
    case "nl":
        taal="Nederlands";
        break;
    default: taal="overige taal";
        break;
}
```

```
switch (leeftijd)
{
    case 18: boodschap = "Meerderjarig!";
        break;
    case 30: boodschap = "Stilaan volwassen?";
        break;
    case 55: boodschap = "Bijna senior!";
        break;
    default: boodschap = "Niet van toepassing.";
        break;
}
```

switch (uitdrukking)

case testgrens:

instructies

[break]/[go to testgrens]

case testgrens:

instructies

[break] / [go to testgrens]

[default:

instructies]

Tip

Maak gebruik van de snippet **switch**



2 Iteratie *Zie handboek hoofdstuk 8 p201*

2.1 Het gebruik van for-lussen

```
For (int i = 0; i <= 5; i++)
{
    TxtResultaat.Text += $"{i}";
}
```

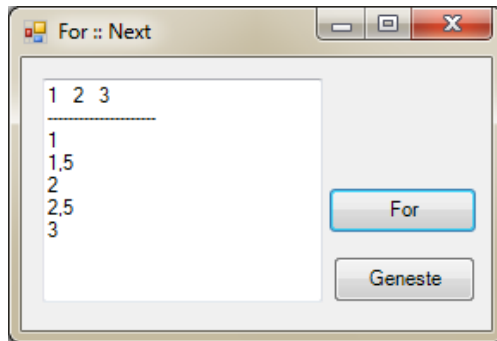
0 1 2 3 4 5

```
for ( int i = 1; i <= 8; i++)
{
    for (float j = 8; j >= 1; j -= 1.5f)
    {
        TxtResultaat.Text += $"{i+j}";
    }
}
```

9	7,5	6	4,5	3
10	8,5	7	5,5	4
11	9,5	8	6,5	5
12	10,5	9	7,5	6
13	11,5	10	8,5	7
14	12,5	11	9,5	8
15	13,5	12	10,5	9
16	14,5	13	11,5	10

```
for (beginwaarde; voorwaarde; stapgrootte)
{
    instructies
}
```

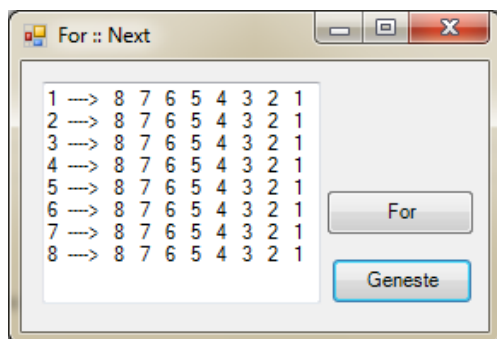
Voorbeelden



```
for (int i = 1; i <= 3; i++)
{
    TxtFor.Text += i.ToString() + "  ";
}

TxtFor.Text += Environment.NewLine + "-----";

for (float i = 1; i <= 3; i += 0.5f)
{
    TxtFor.Text += "\n" + i.ToString() ;
}
```



```
for (int i = 1; i <= 8; i++)
{
    TxtFor.Text += i.ToString() + " ----> ";

    for (int j = 8; j >= 1; j--)
    {
        TxtFor.Text += j.ToString() + "  ";
    }

    TxtFor.Text += "\n";
}
```

2.2 Het gebruik van de while-lussen

```
int i = 0;
while (i <= 10)
{
    TxtResultaat.Text += i.ToString();
    i++;
}
```

```
int i = 0;
do
{
    TxtResultaat.Text += i.ToString();
    i++;
} while (i <= 10);
```

```
0 1 2 3 4 5 6 7 8 9 10
```

```
while (voorwaarde)
{
    [instructies]
}
```

```
do
{
    [instructies]
} while (voorwaarde)
```

Hoofdstuk 6 Veel gebruikte klassen

6.1 System.Math klasse

PI EN E		
Math.PI	-->	3,14159265358979
Math.E	-->	2,71828182845905
ABSOLUTE WAARDEN		
Math.Abs(-12.34)	-->	12,34
MINIMUM EN MAXIMUM		
Math.Min(-12.34, 10)	-->	-12,34
Math.Max(-12.34, 10)	-->	10
AFRONDEN		
Math.Round(10.5)	-->	10
Math.Round(10.51)	-->	11
Math.Round(11.5)	-->	12
Math.Round(-12.51)	-->	-13
Math.Round(-12.3456, 2)	-->	-12,35
Math.Ceiling(-11.5)	-->	-11
Math.Ceiling(12.5)	-->	13
Math.Floor(-11.5)	-->	-12
Math.Floor(12.5)	-->	12
MACHTSVERHEFFING, LOGARITME, VIERKANTSWORTEL		
Math.Pow(2, 3)	-->	8
Math.Log(8, 2)	-->	3
Math.Sqrt(16)	-->	4

Bij 5 achter de komma wordt er afgeronden naar het dichtstbijzijnde even getal! Vb. 10.5 a 10

6.2 System.String klasse

Compare()	vergelijkt de inhoud van strings (A<B = -1 , A=B = 0, A>B = 1)
Concat()	voegt strings samen
Equals()	vergelijken van de waarden in strings
Insert()	voegt een string in een andere string in
IndexOf()	geeft positie waar string begint
Join()	voegt een array naar een string met het opgegeven separator
IsNullOrEmpty	test of een string Null is of leeg ("")
Remove()	verwijdert een gedeelte van een string
Replace()	vervangt een letterteken door een ander teken
PadLeft()/PadRight	vult links/rechts met opgegeven karakters aan
Split()	splitst een string in een array van strings
StartsWith()/EndsWith	geeft true of false als een string begint/eindigt met opgegeven string
Substring()	leest een substring uit een string
ToLower()	zet een string om in kleine letters
ToUpper()	zet een string om in hoofdletters
Trim()	verwijdert een bepaald teken links en recht van een string
TrimEnd()	verwijdert een bepaalde teken aan einde van string
TrimStart()	verwijdert een bepaalde teken aan begin van string

Voorbeelden

<code>string.Concat("Briers - ", "Dox");</code>	"Briers - Dox"
<code>string.Compare("aa", "AA");</code>	-1 (kleiner) / 0 (gelijk)
<code>string.Compare("aa", "AA", true);</code>	0 (gelijk) True: negeert hoofdletters
<code>string.CompareOrdinal("A", "a");</code>	-32 (vergelijkt ASCII-code : 32 kleiner) A=65 en a=97
<code>string.Equals("abcd", "Abcd");</code>	false
<code>"abcd".Equals("Abcd");</code>	false
<code>string.Equals("A", "a", StringComparison.OrdinalIgnoreCase)</code>	(hoofdlet. en kleine letters gelijk)
<code>"Visual CSharp".Substring(2);</code>	"sual CSharp"
<code>"Visual CSharp".Substring(2,5);</code>	"sual "
<code>"Visual CSharp".PadLeft(20);</code>	"Visual CSharp"
<code>"Visual CSharp".PadLeft(20, '*');</code>	"*****Visual CSharp"
<code>"Visual CSharp".Contains("Visual")</code>	true
<code>"Visual CSharp".Contains("Visual C#")</code>	false
<code>"Visual CSharp".IndexOf("*");</code>	-1 (Niet gevonden)
<code>"Visual CSharp".IndexOf("CSh");</code>	7
<code>"Visual CSharp ".Trim();</code>	"Visual CSharp"
<code>"Visual CSharp".Remove(4);</code>	"Visu"
<code>"Visual CSharp".Remove(4,3);</code>	"VisuCSharp"
<code>"Visual C#".StartsWith("Visual")</code>	true
<code>"Visual CSharp".Insert(2, "PATRICIA");</code>	"ViPATRICIASual CSharp "
<code>"Visual CSharp".Length;</code>	13
<code>"Visual CSharp".Replace ("a", "o");</code>	Visuol CShorp
<code>"Visual CSharp".ToUpper();</code>	VISUAL CSHARP
<code>"Visual CSharp".ToLower();</code>	visual csharp

Voorbeeld Join() - Split()

```
string[] element = {"PCVO Limburg", "Campus PXL Hasselt", "Elfde-Liniestraat 26", "3500 Hasselt"};

Console.WriteLine("\n === Afdruk lus === ");
for (int i = 0; i < element.Length; i++)
{
    Console.Write(element[i] + ", ");
}

Console.WriteLine("\n\n === Afdruk string.Join() === ");

Console.WriteLine(string.Join(", ", element));

Console.WriteLine("\n\n === Afdruk string.Split() ===");
string adres = "PCVO Limburg,Campus PXL Hasselt,Elfde-Liniestraat 26,3500 Hasselt";
string[] delen = adres.Split(',');

for (int i = 0; i < delen.Length; i++)
{
    Console.WriteLine($"Element {i}: {delen[i]}");
}
```

```
=== Afdruk lus ===
PCVO Limburg, Campus PXL Hasselt, Elfde-Liniestraat 26, 3500 Hasselt,

=== Afdruk string.Join() ===
PCVO Limburg, Campus PXL Hasselt, Elfde-Liniestraat 26, 3500 Hasselt

=== Afdruk string.Split() ===
Element 0: PCVO Limburg
Element 1: Campus PXL Hasselt
Element 2: Elfde-Liniestraat 26
Element 3: 3500 Hasselt
```

NIET doen:

```
sTemp == "";
sTemp == "Test";
if (sTemp > "Test") ...
```

Wel Doen:

```
sTemp.Length == 0;
sTemp.Equals("Test");
if (sTemp.CompareTo("Test") > 0)
```

In tegenstelling tot de '=='-operator geeft Equals een foutmelding (dat opgevangen kan worden) als een string=null.

6.3 System.Text klasse

Met een **StringBuilder** kan je een object bouwen zonder een object meermaals te openen zoals bij het samenvoegen van strings of een tekst toevoegen aan een stringobject.

Voorbeeld

```
StringBuilder sb = new StringBuilder();
int j = 10, i=0;
for (; i < 10; i++)
{
    j += i;
    sb.Append("punt: ").Append(i.ToString()).AppendLine();
}
sb.Insert(0, "StringBuilder:").Insert(14, Environment.NewLine); // vooraan 14 kar toevoegen
sb.Replace(':', '-', 15, sb.Length-15); // vanaf positie 15 ':' door '-' vervangen
sb.AppendFormat("{0}, {1}", i, j); // i en j worden onderaan toegevoegd
Console.Write(sb);
sb.Clear(); //lengte = 0, dus leeg
sb.Remove(0,14); // eerste 14 karakters (StringBuilder:) verwijderd

// Afdruk
StringBuilder:                                punt- 6
punt- 0                                       punt- 7
punt- 1                                       punt- 8
punt- 2                                       punt- 9
punt- 3                                       10, 55
punt- 4
punt- 5
```

6.4 System.DateTime en TimeSpan klasse

Een instantie van de klasse DateTime representeert een datum tussen 1 januari 0001 en 31 december 9999 en een tijd tussen 0:00:00 (middernacht) en 23:59:59.

Eigenschappen

DateTime datum //variabele van datatype DateTime

DateTime.Today	21/10/2020 0:00:00
DateTime.Now	21/10/2020 15:54:10
datum.Date	21/10/2020 15:54:10;5545904
datum.Day	21
datum.DayOfWeek	Wednesday
datum.DayOfYear	297
datum.Hour	15
datum.Minute	54
datum.Month	10
datum.TimeOfDay	15:54:10.5545904
datum.Year	2020

Methoden

<code>datum.AddDays(36)</code>	voegt dagen toe of trek dagen af
<code>datum.AddMonths(12)</code>	voegt maanden toe of trek maanden af
<code>datum.AddYears(2)</code>	voegt jaren toe of trek jaren af
<code>datum.Subtract(datum)</code>	geeft verschil in dagen, uren en minuten (geen maanden, jaren)
<code>datum.Subtract(datum).Days</code>	geeft verschil in dagen tussen de opgegeven datums
<code>DateTime.Parse (string)</code>	zet string om naar datum
<code>datum.ToLongDateString()</code>	geeft lange datum: dddd d mmmm yyyy
<code>datum.ToShortDateString()</code>	geeft korte datumnotatie: d/mm/yyyy
<code>datum.ToLongTimeString()</code>	geeft lange tijdsnotatie: hh:mm:ss
<code>datum.ToShortTimeString()</code>	geeft korte tijdsnotatie: hh:mm

Voorbeeld

```
// declareren en initialiseren van variabele datum
DateTime vandaag = DateTime.Today; // huidige systeemdatum
DateTime datum = new DateTime(2020, 10, 3); // 3 oktober 2020
DateTime conversieDatum = DateTime.Parse("2020-10-3"); // converteert string naar datum
Console.WriteLine(DateTime.Now); //3/10/2020 16:32:47
Console.WriteLine(vandaag.AddYears(2)); //3/10/2022 16:32:47
Console.WriteLine(vandaag.TimeOfDay); //16:32:47.2911001
Console.WriteLine(vandaag.DayOfWeek); // Saturday
Console.WriteLine((int)vandaag.DayOfWeek); // 6
Console.WriteLine(vandaag.Subtract(gbDatum).Days); // geeft dagen tussen vandaag en gbDatum
Console.WriteLine(vandaag.AddMonths(12)); //3/10/2021 16:32:47
Console.WriteLine(vandaag.AddDays(-6)); //27/9/2020 16:32:47
Console.WriteLine(vandaag.ToLongDateString()); //zaterdag 3 oktober 2020
Console.WriteLine(vandaag.ToShortDateString()); //3/10/2020
Console.WriteLine(vandaag.ToLongTimeString()); //16:32:47
Console.WriteLine(vandaag.ToShortTimeString()); //16:32
```

Voorbeeld tijdsverloop

```
// Initialiseren van tijdsverloop
TimeSpan interval = new TimeSpan(2, 14, 18); // geeft "02:14:18"

// Tijdsverloop berekenen.
DateTime vertrek = new DateTime(2017, 7, 2, 18, 32, 0);
DateTime aankomst = new DateTime(2017, 7, 12, 22, 47, 10);
TimeSpan reistijd = aankomst - vertrek;
Console.WriteLine($"{aankomst} - {vertrek} = {reistijd}");
// geeft: 12/07/2017 22:47:10 - 2/07/2017 18:32:00 = 10.04:15:10 dus 10 dagen, 4 uur, 15 min en 10 sec

// Tijd tussen tijdseenheden berekenen.
DateTime tijd = new DateTime(1999, 1, 1);
TimeSpan dagen = new TimeSpan(DateTime.Today.Ticks - tijd.Ticks);
Console.WriteLine($"Sinds de invoering van de euro zijn {dagen.Days} dagen verstreken.");
// geeft: Sinds de invoering van de euro zijn 6879 dagen verstreken.

DateTime tijd = new DateTime(2017, 11, 1, 0, 0, 0);
TimeSpan duur = new TimeSpan(DateTime.Now.Ticks - tijd.Ticks); // 10 miljoen ticks in een sec.
Console.WriteLine($"Duur: {Math.Truncate(duur.TotalMinutes)}:{Math.Truncate(duur.TotalSeconds)}");
// sinds 1 november zijn er 699 minuten en 41980 sec verstreken.
```

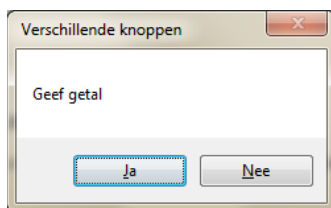
6.4 System.Windows.Forms.MessageBox klasse *Zie eveneens handboek p 29 en 455*

Met het berichtenvenster kan je in de vorm van een dialoogvenster een mededeling doorgeven waardoor de gebruiker deze informatie moet bekijken.

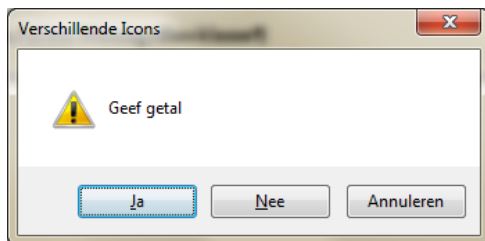
6.4.1 Berichtenvenster gebruiken.



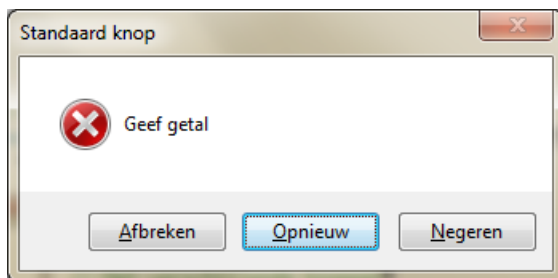
```
// Standaard MessageBox  
MessageBox.Show("Geef een getal in", "Standaard");
```



```
// Met verschillende knoppen  
MessageBox.Show("Geef getal", "Verschillende knoppen",  
MessageBoxButtons.YesNo);
```



```
// Met verschillende icons  
MessageBox.Show("Geef getal", "Verschillende Icons",  
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
```



```
//Standaard knop  
MessageBox.Show("Geef getal", "Standaard knop",  
MessageBoxButtons.AbortRetryIgnore, MessageBoxIcon.Stop,  
MessageBoxResult.Retry);
```

6.4.2 Dialoogwaarden opvangen van het berichtenvenster

```
// Resultaatwaarden  
MessageBoxResult antw = MessageBox.Show("Wil je echt afsluiten?", "Project afsluiten.",  
MessageBoxButton.YesNo, MessageBoxImage.Question, MessageBoxResult.No);  
  
if (antw == MessageBoxResult.Yes)  
{  
    e.Cancel = false;  
}  
else  
{  
    // Cancel het Closing-event en windows wordt niet gesloten.  
    e.Cancel = true;  
}
```

```

if (antw == DialogResult.Retry) TxtResultaat.Text = "Opnieuw";
if (antw == DialogResult.Abort) TxtResultaat.Text = "Afbreken";
if (antw == DialogResult.Ignore) TxtResultaat.Text = "Negeren";

```

6.5 Microsoft.VisualBasic.Interaction klasse

In Visual C# bestaat het dialoogvenster `InputBox` niet. Niets belet ons om de klasse `Visual Basic` te importeren opdat we gebruik kunnen maken van deze functionaliteit.



```
string InputBox("Prompt","Title","DefaultResponse",intXpos,intYPos)
```

Voorbeeld

// Voeg bij References 'Microsoft.VisualBasic' toe en voeg daarna in je code de library via `USING` toe.

```

using Microsoft.VisualBasic;

string antwoord = Interaction.InputBox("Geef een getal", "Invoer", "50", 500);
// Testen tot er een invoer is.
while (string.IsNullOrEmpty(antwoord)) // Cancel, Sluiten of lege string
{
    MessageBox.Show("Geef een getal in", "Foutieve invoer", MessageBoxButtons.OK,
                    MessageBoxIcon.Exclamation);
    antwoord = Interaction.InputBox("Geef een getal", "Invoer", "50", 500);
};

```

NB. De positiecoördinaten voor het invoervenster worden weergegeven in pixels. Wanneer je geen coördinaten opgeeft dan wordt het invoervenster automatisch gecentreerd.

Een pixel (samentrekking **p**ictures en **e**lement) is een punt van een digitaal scherm. Het aantal pixels is bepalend voor de resolutie (vb. 1280 × 1024 pixels) van het beeldscherm.

6.5 System.Random klasse *Zie eveneens handboek 6.7 p 151*

Gebruik deze klasse om willekeurige getallen te genereren. Instanties van deze klasse *Random* bieden ons een 'stroom' van willekeurige getallen die we één voor één kunnen verkrijgen via de `Next`-methode.

`Random()` is nodig om het algortime binnen de klasse *Random* te initialiseren. Indien de parameter van `Random()` leeg is, wordt de systeemtijd hiervoor gebruikt en daardoor heb je telkens andere gegevens.

Gebruik je echter een parameter (Int32) dan krijg je steeds dezelfde willekeurige getallenreeks wanneer je opstart.

Voorbeeld

```
// Modulevariabelen
private Random rndGetal1 = new Random();
private Random rndGetal2 = new Random(1);

private void BtnRandom1_Click(object sender, EventArgs e)
{
    // >=0 getal <int32.MaxValue (2 147 483 647)
    TxtResultaat.Text += rndGetal1.Next();           // 974985579 2063543068 179342757,...

    // >=0 getal <100
    TxtResultaat.Text += rndGetal1.Next(100);        // 37 22 42 20 83,...

    // >=10 getal <20
    TxtResultaat.Text += rndGetal1.Next(10, 20);      // 13 16 14 10 12,...

    // >=0.0000000000000000 getal < 0.9999999999999978
    TxtResultaat.Text += rndGetal1.NextDouble();     // 0,317507218251707 0,236007541993636,...
}

private void BtnRandom2_Click(object sender, EventArgs e)
{
    TxtResultaat.Text += rndGetal2.Next();           // 534011718 237820880 1002897798 1657007234
    TxtResultaat.Text += rndGetal2.Next(100);        // 24 11 46 77 65
    TxtResultaat.Text += rndGetal2.Next(10, 20);      // 12 11 14 17 16
    TxtResultaat.Text += rndGetal2.NextDouble();     // 0,248668584157093 0,110743977181029
}
```

Wanneer je de volgende keer opnieuw met BtnRandom2 begint, krijg je dezelfde reeks van getallen!

6.5 Klasse DispatcherTimer *Zie eveneens handboek 6.8 p 154*

Vermits er geen XAML-notatie is voor een DispatcherTimer-object moet je het object volledig coderen.

```
using System.Windows.Threading; //Namespace toevoegen!!

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Installeren van timer dmv de klasse aan te spreken.
    DispatcherTimer wekker= new DispatcherTimer();

    // Timer laten aflopen om de seconde.
    wekker.Tick += new EventHandler(DispatcherTimer_Tick);
    wekker.Interval = new TimeSpan(0, 0, 1); //uren, minuten, seconden

    // Timer laten starten
    wekker.Start();

    // TIJD instellen.
    tijd = DateTime.Now;
    LblTijd.Content = $"{tijd.ToLongDateString()} {tijd.ToLongTimeString()}";
}
```

```
private void DispatcherTimer_Tick(object sender, EventArgs e)
{
    LblTijd.Content = $"{DateTime.Now.ToLongDateString()} {DateTime.Now.ToLongTimeString()}";
}
```

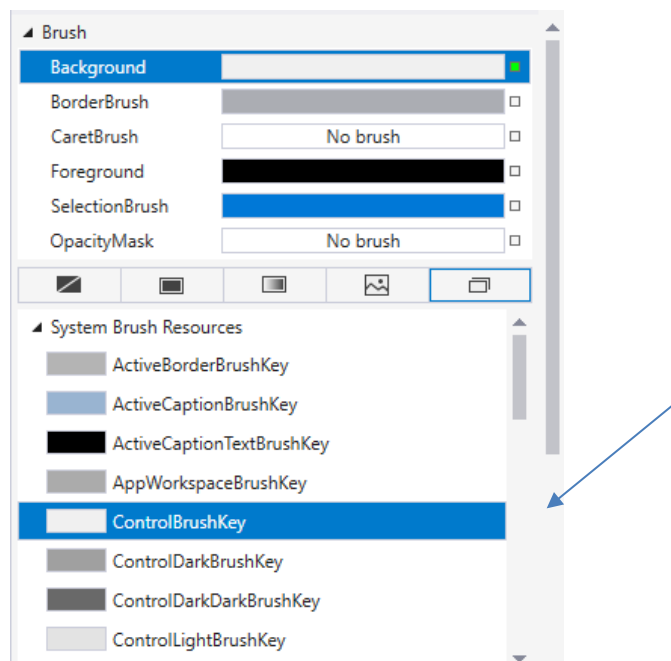
6.6 Char structure

Visual studio bevat een aantal methods om te werken met karakters:

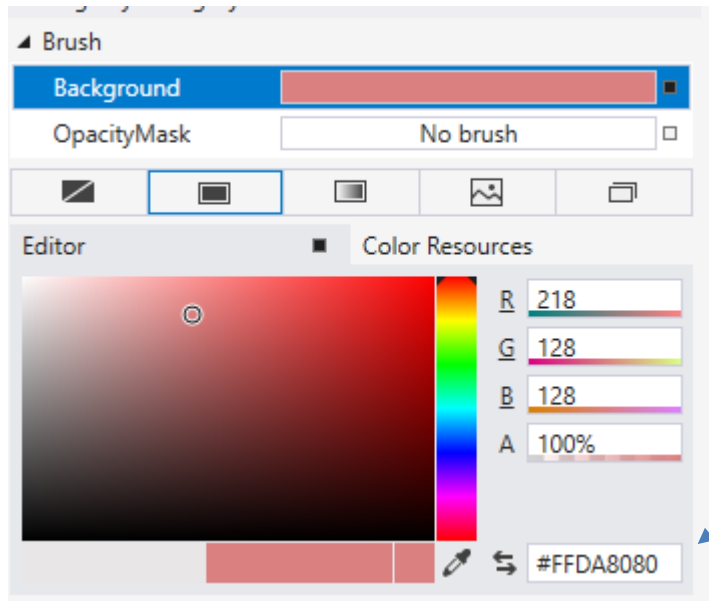
```
char karakter1 = 'A';
char karakter2 = '½'; // ALT+171
char karakter3 = '9';
Console.WriteLine(karakter1.CompareTo('B')); // Output: "-1" ('A' < 'B')
Console.WriteLine(karakter1.Equals('A')); // Output: "True"
Console.WriteLine(char.GetNumericValue(karakter2)); // Output: "0,5"
Console.WriteLine(char.GetNumericValue(karakter3)); // Output: "9"
Console.WriteLine(char.IsDigit(karakter2)); // Output: "False"
Console.WriteLine(char.IsDigit(karakter3)); // Output: "True"
Console.WriteLine(char.IsLetter(', ')); // Output: "False"
Console.WriteLine(char.IsLower('u')); // Output: "True"
Console.WriteLine(char.IsNumber(karakter2)); // Output: "True"
Console.WriteLine(char.IsNumber(karakter3)); // Output: "True"
Console.WriteLine(char.IsPunctuation('.', ')); // Output: "True"
Console.WriteLine(char.Parse("S")); // Output: "S"
Console.WriteLine(char.ToLower('M')); // Output: 'm'
```

6.7 Werken met kleuren

XAML



```
< ... Background="{DynamicResource {x:Static SystemColors.ControlBrushKey}}" />
```



< ... Background="#FFDA8080">

PROGRAMMEERCODE

1 Kleurentabel gebruiken (zie Bijlage 3)

```
// Achtergrondkleur zetten.  
TxtSpeler.Background = Brushes.LightGreen;
```

2 Kleurenallet met hexadecimale waarde gebruiken

```
using System.Windows.Media;  
// Kleurenallet met hexadecimale waarde zetten.  
var bc = new BrushConverter();  
TxtSpeler.Background = (Brush)bc.ConvertFrom("#FFE8E6E6");
```

3 Systeemkleuren gebruiken

```
using System.Windows.Media;  
// Systeemkleuren zetten.  
TxtProgramma.Background = SystemColors.ControlBrush; // Niet ControlBrushKey gebruiken
```


Hoofdstuk 7 Methoden en parameters

Zie eveneens handboek hoofdstuk 5 p 89.

We onderscheiden 3 soorten methoden/procedures:

- Eventprocedures: vb. Click(), KeyDown(), MouseEnter(),...
- Voidprocedures: vb. `MessageBox.Show()`, `Focus()`,...
- Functieprocedures : vb. `Round()`, `int.Parse()`, `ToString()`...

In de technologie van het object georiënteerd programmeren noemen we een procedure een **methode**.

We kunnen zelf methoden definiëren omdat ze de volgende voordelen hebben:

- geen herhaling van statements (programmacode);
- programma's zijn gemakkelijk te lezen;
- programma's zijn gemakkelijker te maken (in groep);
- kan ook in andere projecten gebruikt worden (public);

7.1 Voidmethode

7.1.1 ByVal (waarde wordt doorgegeven)

```
private void BtnByVal_Click(object sender, EventArgs e)
{
    int getal1 = 1, getal2 = 2;
    // --- OPROEPEN VAN METHOD ---
    TestByVal(getal1, getal2);

    Console.WriteLine($"Waarde na oproep: {getal1} {getal2}");
}

private void TestByVal(int getal1, int getal2)
{
    Console.WriteLine($"Waarde in test voor wijziging: {getal1} {getal2}");
    getal1 += 10;
    getal2 += 10;
    Console.WriteLine($"Waarde in test na wijziging: {getal1} {getal2}");
}

// Afdruk
Waarde in TestByVal voor wijziging: 1 2
Waarde in TestByVal na wijziging: 11 12
Waarde na oproep: 1 2
```

NB. Je kan ook meteen aan je parameters een waarde toekennen:

```
// --- OPROEPEN VAN METHOD ---
TestByVal(getal1: 1, getal2: 2); of TestByVal(getal1: var1, getal2: var2);
```

17.1.2 ByRef (geheugenplaats wordt doorgegeven)

```
private void BtnByRef_Click(object sender, EventArgs e)
{
    int getal1 = 1;
    int getal2 = 2;
    Console.WriteLine($"Waarde voor oproep: {getal1} {getal2}");
    TestByRef(ref getal1, ref getal2);
    Console.WriteLine($"Waarde na oproep: {getal1} {getal2}");
}
```

```
private void TestByRef(ref int getal1, ref int getal2)
{
    Console.WriteLine($"Waarde in test voor wijziging: {getal1} {getal2}");
    getal1 += 10;
    getal2 += 10;
    Console.WriteLine($"Waarde in test na wijziging: {getal1} {getal2}");
}

// Afdruk
Waarde voor oproep: 1 2
Waarde in TestByRef voor wijziging: 1 2
Waarde in TestByRef na wijziging: 11 12
Waarde na oproep: 11 12
```

7.1.3 Out

In tegenstelling tot *ref* moet *out* altijd een beginwaarde hebben in de geschreven method. Je kan *out* dus enkel gebruiken om met (lege) waarden te starten. Het sleutelwoord *out* verwijst naar een uitgaande parameter en zorgt ervoor dat de argumenten zonder voorafgaande initialisatie worden doorgestuurd.

```
private void BtnOut_Click(object sender, EventArgs e)
{
    int getal1 = 1;
    TestOut( getal1, out int getal2, out int getal3);
    Console.WriteLine($"Waarde na oproep: {getal1} {getal2} {getal3}");
}

private void TestOut(int getal1, out int getal2, out int getal3)
{
    getal1 = 10; // heeft geen zin want de waarde kan toch niet gebruikt worden.
    getal2 = 10; //beginwaarden
    getal3 = 10;
    getal3++;
}

// Afdruk
Waarde na oproep: 1 10 11
```

7.1.4 Overloading

// Methoden met dezelfde naam met verschillende parameters.

```
private void TestByRef(ref int getal1, ref int getal2)
{
    Console.WriteLine($"Waarde in test voor wijziging: {getal1} {getal2}");
    getal1 += 10;
    getal2 += 10;
    Console.WriteLine($"Waarde in test na wijziging: {getal1} {getal2}");
}

private void TestByRef(ref int getal1, ref int getal2, ref int getal3)
{
    Console.WriteLine($"Waarde in test voor wijziging: {getal1} {getal2} {getal3}");
    getal1 += 10;
    getal2 += 10;
    getal3 += 10;
    Console.WriteLine($"Waarde in test na wijziging: {getal1} {getal2} {getal3}");
}
```

```
private void BtnProcOptional_Click(object sender, EventArgs e)
{
    int getal1 = 1, getal2 = 2, getal3 = 3;
    TestByRef (ref getal1, ref getal2);
    Console.WriteLine($"Waarde na oproep: " {getal1} {getal2}");
    TestByRef (ref getal1, ref getal2, ref getal3);
    Console.WriteLine($"Waarde na oproep: {getal1} {getal2} {getal3}");
}
```

// Afdruk

Waarde in TestByRef voor wijziging: 1 2

Waarde in TestByRef na wijziging: 11 12

Waarde na oproep: 11 12

Waarde in TestByRef voor wijziging: 11 12 3

Waarde in TestByRef na wijziging: 21 22 13

Waarde na oproep: 21 22 13

7.2 Functiemethode en resultaten

7.2.1 ByVal

```
private void BtnByValFunctie_Click(object sender, EventArgs e)
{
    int getal1 = 10;
    int getal2 = 25;
    int getal3 = Som(getal1, getal2);
    Console.WriteLine($"Waarde na oproep {getal1} {getal2} {getal3}");
}
```

```
private int Som(int getal1, int getal2)
{
    getal1++;
    getal2++;
    return getal1 + getal2;
}
```

// Afdruk

Waarde na oproep 10 25 37

7.2.2 ByRef

```
private void BtnRefFunctie_Click(object sender, EventArgs e)
{
    int getal1 = 10;
    int getal2 = 25 ;
    Console.WriteLine($"Waarde voor oproep: {getal1} {getal2}");
    int getal3 = SomRef(ref getal1, ref getal2);
    Console.WriteLine($"Waarde na oproep: {getal1} {getal2} {getal3}");
}

private int SomRef(ref int getal1, ref int getal2)
{
    Console.WriteLine($"Waarde in SomRef voor wijziging: {getal1} {getal2}");
    getal1 += 10;
    getal2 += 10;
    Console.WriteLine($"Waarde in SomRef na wijziging: {getal1} {getal2}");
    return getal1 + getal2;
}
```

```
// Afdruk
Waarde voor oproep: 10 25
Waarde in SomRef voor wijziging: 10 25
Waarde in SomRef na wijziging: 20 35
Waarde na oproep: 20 35 55
```

7.2.3 Overloading

```
// Functiemethoden met 2 of 3 parameters
```

```
private void BtnOptionalFunctie_Click(object sender, EventArgs e)
{
    int getal1 = 10;
    int getal2 = 25;
    int getal3 = 30;
    int getal4 = SomRef(ref getal1, ref getal3);
    Console.WriteLine($"Waarde na 1ste oproep: {getal1} {getal2} {getal3} {getal4}");
    getal4 = SomRef(ref getal1, ref getal2, ref getal3);
    Console.WriteLine($"Waarde na 2de oproep: {getal1} {getal2} {getal3} {getal4}");
}

private int SomRef(ref int getal1, ref int getal2)
{
    Console.WriteLine($"Waarde in SomRef voor wijziging: {getal1} {getal2}");
    getal1 += 10;
    getal2 += 10;
    Console.WriteLine($"Waarde in SomRef na wijziging: {getal1} {getal2}");
    return getal1 + getal2;
}

private int SomRef(ref int getal1, ref int getal2, ref int getal3)
{
    Console.WriteLine($"Waarde in SomRef voor wijziging: {getal1} {getal2} {getal3}");
    getal1 += 10;
    getal2 += 10;
    getal3 += 10;
    Console.WriteLine($"Waarde in SomRef na wijziging: {getal1} {getal2} {getal3}");
    return getal1 + getal2 + getal3;
}

// Afdruk
Waarde in SomRef voor wijziging: 10 30
Waarde in SomRef na wijziging: 20 40
Waarde na 1ste oproep: 20 25 40 60
Waarde in SomRef voor wijziging: 20 25 40
Waarde in SomRef na wijziging: 30 35 50
Waarde na 2de oproep: 30 35 50 115
```

Hoofdstuk 8 Eventprocedures en parameters

8.1 Toetsaanslagen controleren

De eventprocedure heeft altijd 2 parameters (sender,e).

- *sender* geeft het actief object aan (event of source)
- *e* geeft eventdata (KeyEventArgs voor KeyPress en EventArgs voor Click)

Door deze parameters kan je de eigenschappen van de objecten ophalen en bewerken. Deze parameters mogen verwijderd worden wanneer ze niet gebruikt worden.

Vb. `private void BtnSluiten_Click()`

KeyDown

KeyDown wordt gebruikt wanneer je een toets op je toetsenbord **indrukt**.

Eigenschappen van KeyEventArgs:

- e. Key geeft het karakter
- e. Handled geeft aan of KeyDown wordt afgehandeld

Voorbeeld

```
private void TextBox_KeyDown(object sender, KeyEventArgs e)
{
    // Test op numeriek toetsenbord.
    if (e.Key >= Key.NumPad0 && e.Key <= Key.NumPad9)
    {
        TxtResultaat.Text = "Numeriek";
    }
    else if (e.Key == Key.K )
    {
        TxtResultaat.Text = "Kleine letter k of hoofdletter K";
    }
    else if (Keyboard.IsKeyDown(Key.LeftShift) || Keyboard.IsKeyDown(Key.RightShift))
    {
        TxtResultaat.Text = " Shift";
    }
    else if (e.Key == Key.Back)
    {
        TxtResultaat.Text = "Backspace";
    }
    else if (e.Key == Key.Return)
    {
        TxtResultaat.Text = "Enter";
    }
    else
    {
        //KeyPress wordt niet aanvaard, dus geen invoer van het karakter.
        e.Handled = true;
    }
}
```


Je kan ook testen op de speciale toetsen ALT, CONTROL, SHIFT en WINDOWS-toets dmv Modifiers.

```
private void TextBox_KeyDown(object sender, KeyEventArgs e)
{
    // Test op numerieke toetsen bovenaan je toetsenbord. De cijfers worden getriggerd door
    // combinatie met de shift. Op eenzelfde toets kunnen immers andere tekens staan.

    if ((e.Key >= Key.D0 && e.Key <= Key.D9) && e.KeyboardDevice.Modifiers == ModifierKeys.Shift)
    {
        Txt2.Text = "numerieke toets";
    }

    if (e.Key == Key.K && e.KeyboardDevice.Modifiers == ModifierKeys.Shift)
    {
        Txt2.Text = "Hoofdletter K";
    }

    if (e.Key == Key.K && e.KeyboardDevice.Modifiers == ModifierKeys.None)
    {
        Txt2.Text = "kleine k";
    }
}
```

```
private void TextBox_KeyDown(object sender, KeyEventArgs e)
{
    // De verschillende toetsaanslagen registreren op de toets 0-à-} 
    if ((e.Key >= Key.D0 && e.Key <= Key.D9) && e.KeyboardDevice.Modifiers == ModifierKeys.Shift)
    {
        TxtDecimaal.Text = "numerieke toets met shift: 0";
    }
    if ((e.Key >= Key.D0 && e.Key <= Key.D9) && e.KeyboardDevice.Modifiers == ModifierKeys.None)
    {
        TxtDecimaal.Text = "gewone aanslag: à";
    }

    // Alt Gr is een combinatie van Alt + Ctrl
    if ((e.Key >= Key.D0 && e.Key <= Key.D9)
        && e.KeyboardDevice.Modifiers == (ModifierKeys.Alt | ModifierKeys.Control))
    {
        TxtDecimaal.Text = "aanslag met Alt Gr: }";
    }
}
```

KeyUp

KeyUp wordt gebruikt wanneer je een toets terug *loslaat*.

Eigenschappen van KeyEventArgs:

- e. Key geeft het karakter
- e. Handled geeft aan of KeyDown wordt afgehandeld

Je kan deze gebeurtenis ook gebruiken om combinatietoetsen te testen. Vermits je vaak niet binnen één seconde 2 toetsen tegelijk kan indrukken, gebruik je best de KeyUp om 2 combinatietoetsen te testen.

Voorbeeld

```
private void TextBox_KeyUp(object sender, KeyEventArgs e)
{
    if (Keyboard.IsKeyDown(Key.LeftShift) || Keyboard.IsKeyDown(Key.RightShift) && e.Key == Key.K)
    {
        TxtResultaat.Text = " Shift + K";
    }
}
```

Hoofdstuk 9 WPF besturingselementen

De **Windows Presentation Foundation** of **WPF** is een grafische onderdeel van het .Net Framework. Vermits het gebruik maakt van een vectorieel grafisch systeem maakt WPF een beter ontwerp van besturingselementen (zoals dialoogvensters, knoppen en keuzelijsten) mogelijk. Daarnaast introduceert WPF een nieuw programmeermodel XAML (uitgesproken als zammel), waarmee op een nieuwe manier gebruikersinterfaces kunnen worden gebouwd. WPF is een combinatie van XAML (markup) en C# / VB.NET / andere .NET taal.

Het belangrijkste verschil tussen Windows Forms en WPF is het feit dat standaard Windows-controles (bijvoorbeeld een TextBox) bovenop een Windows Forms wordt gebouwd. WPF wordt vanuit het niets opgebouwd en is niet afhankelijk van standaard Windows instellingen.

Het nadeel van deze flexibiliteit is dat je soms harder moet werken dan Windows Forms.

XAML, wat staat voor **eXtensible Application Markup Language** dat wordt gebruikt voor het beschrijven van een GUI. In vorige GUI frameworks, zoals Windows Forms, werd een GUI gemaakt in C# of VB.

Basis XAML

Een knop wordt gemaakt door:

```
<Button>
```

XAML-tags moet worden beëindigd, hetzij door het schrijven van de eind-tag of door een slash op het einde van de start-tag:

```
<Button> </ Button>
```

Of

```
<Button />
```

Tussen de begin- en eind-tag kan je ook gegevens opnemen:

```
<Button Margin="50,45,342,234"> A-knop </Button>
```

```
<Button FontWeight = "Bold" Content = "Een knop" Margin="50,115,347,174" />
```

Events in XAML

Drie manieren om een gebeurtenis te koppelen aan een object:

- Dubbelklik op button en je krijgt het standaardevent.
- Selecteer in het Property-venster de juiste gebeurtenis.
- Typ MouseDown en vervolgens op <New Event Handler> en IntelliSense doet de rest.


```

mples.XAML.EventsSample"
microsoft.com/winfx/2006/xaml/presentation"
;.microsoft.com/winfx/2006/xaml"
:ight="300" Width="300">
>useUp="pnlMainGrid_MouseUp" Background="LightBlue" MouseDown="">

```



9.1 Windows

```

<Window x:Class="_51WpfTryCatchDatum.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:_51WpfTryCatchDatum"
        mc:Ignorable="d"
        Title="Correcte datum controleren..." Height="333.515" Width="536.376" FontSize="18" Loaded="Window_Loaded">
    <Window.Background>
        <ImageBrush ImageSource="Afbeeldingdatum.jpg"/>
    </Window.Background>

```

- Eigenschappen: Name: naam van formulier
 Title: weergave van tekst in titelbalk
 Width: breedte van formulier in pixels
 Height: hoogte van formulier in pixels
 Backcolor: achtergrondkleur van formulier
 FontFamily: lettertype voor alle objecten op het formulier
 WindowsState: geeft venstergrootte (Maximized, Minimized of Normal)
- Method: Close(): formulier wordt gesloten
- Events: Loaded(): treedt op wanneer het formulier geladen wordt

9.2 Label

```

<Label Content="Personeelslid" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="60,51,0,0"
        HorizontalContentAlignment="Right"/>

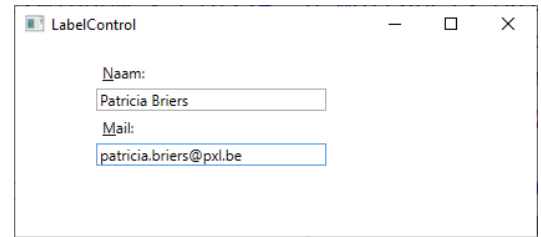
```

- Eigenschappen: Name: naam van label
 Content: weergave van tekst
 BackGround: achtergrondkleur
 ForeGround: voorgrondkleur
 FontFamily: lettertype en puntgrootte
 HorizontalContentAlignment: plaats van tekst in label (links, rechts...)
 Visibility: Hidden/Visibility dan label zichtbaar is of niet

```

<!--Klik op ALT en N en de focus gaat naar txtName-->
<Label Content="_Naam:" Target="{Binding ElementName=TxtNaam}" />
<TextBox x:Name="TxtNaam" />
<!--Klik op ALT en M en de focus gaat naar txtMail-->
<Label Content="_Mail:" Target="{Binding
ElementName=TxtMail}" />
<TextBox x:Name="TxtMail" />

```



9.3 TextBlock

Zoals een label wordt een TextBlock gebruikt om tekst weer te geven. Er zijn echter wel een paar belangrijke verschillen tussen het label en het TextBlock. Met het TextBlock kan je alleen tekst weergeven maar met een label kan je ook:

- een rand maken;
- disabled maken;
- kan ook via content een afbeelding bevatten;
- een sjablooninhoud via de eigenschap ContentTemplate gebruiken;
- sneltoetsen toewijzen gekoppeld aan gerelateerde bedieningselementen.

Gebruik een TextBlock wel als je enkel tekst wenst weer te geven want het is een veel lichter besturingselement dan een label.

9.4 Tekstvak (TextBox)

```

<TextBox x:Name="TxtResultaat" HorizontalAlignment="Left" Height="200" Margin="65,190,0,0"
TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="395" IsEnabled="False"
Background="#FFFAE5E5" FontFamily="Consolas"/>

```

- Eigenschappen:

Name:	naam van tekstvak
Text:	inhoud die getoond wordt in het tekstvak
FontFamily/FontWeight:	lettertype, lettergrootte...
Foreground:	voorgroundkleur
BackGround:	achtergrondkleur
IsEnabled:	True/False beschikbaarheid van het tekstvak
IsReadOnly:	True/False inhoud kan enkel gelezen worden
MaxLength:	maximaal aantal karakters in het tekstvak
MaxLine:	maximaal aantal lijnen in tekstvak
VerticalScrollbarVisibility:	schuifbalk (verticaal)
HorizontalScrollbarVisibility:	schuifbalk (horizontaal) – enkel actief als TextWrapping=NoWrap
TextWrapping:	tekstterugloop of WordWrapping instellen
Visibility:	True/False tekstvak zichtbaar stellen
- Method:

Focus():	tekstvak krijgt de focus
Clear():	inhoud van tekst wordt gewist
- Events:

Click():	treedt op wanneer op tekstvak wordt geklikt
TextChanged():	treedt op bij wijziging van inhoud
GotFocus():	tekstvak krijgt de focus
LostFocus():	tekstvak verliest de focus

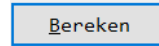
9.5 Opdrachtknop (Button)

```
<Button x:Name="ButtonBerekenen" Content="_Berekenen" HorizontalAlignment="Left"
Margin="355,35,0,0" VerticalAlignment="Top" Width="105" Height="35" IsDefault="True"
Click="ButtonBerekenen_Click"/>
```

- Eigenschappen:

Name:	naam van knop
Text:	tekst op knop Vb. _Bereken (Alt+B) om Click() te activeren
Width:	breedte in pixels
Height:	hoogte in pixels
FontFamily:	lettertype, lettergrootte...
FontStyle:	Italic,...
FontWeight:	Thin, Bold, SemiBold,.....
Foreground:	voorgroondkleur
Background:	achtergrondkleur
IsDefault:	standaardknop
IsEnabled:	True/False beschikbaarheid van het tekstvak
Visibility:	tekstvak zichtbaar stellen
- Method: Focus(): knop krijgt de focus
- Events:

Click():	treedt op wanneer op tekstvak wordt geklikt
GotFocus():	knop krijgt de focus
LostFocus():	knop verliest de focus
MouseEnter():	muis komt over de knop

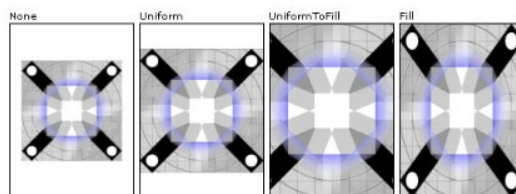


9.6 Afbeeldingsvak (Image)

```
<Image x:Name="ImgDuimen" Source="duimenOmhoog.jpg" Stretch="Fill"/>
```

- Eigenschappen:

Name:	naam van object
Width:	breedte in pixels
Height:	hoogte in pixels
Source:	afbeelding toekennen
IsEnabled:	beschikbaarheid van het object
Visibility:	True/False object zichtbaar stellen
Stretch:	het formaat van de afbeelding



Als je wilt werken met afbeeldingen besteed je best even aandacht aan de werking van een image.

- Afbeelding toekennen een image in XAML

```
<Image HorizontalAlignment="Left" Height="61" Margin="837,337,0,0" VerticalAlignment="Top" Width="54" Source="Telaat.png"/>
```

- Afbeelding toekennen aan knop in XAML

Als je over een knop met de muis beweegt, verdwijnt je afbeelding op de knop doordat de content van de knop wordt getoond. Als je in de Designer je **Background** en je **Content** met dezelfde afbeelding opgeeft, blijft je afbeelding netjes staan.

```
<Button.Background>
    <ImageBrush ImageSource="Telaat.png" Stretch="UniformToFill"/>
</Button.Background>
```

```
<Button.Content>
    <Image Source="Telaat.png"/>
</Button.Content>
```

- Afbeelding aan een image programmeren

```
ImgAfb.Source = new BitmapImage(new Uri(@"Afbeeldingen\Telaat.png", UriKind.RelativeOrAbsolute));
```

- Nieuwe afbeelding op een knop programmeren

```
// Afbeelding
Image afb = new Image();
afb.Source = new BitmapImage(new Uri(@"Afbeeldingen\Telaat.png", UriKind.RelativeOrAbsolute));
afb.Stretch = Stretch.Fill;
Button.Content = afb;
```

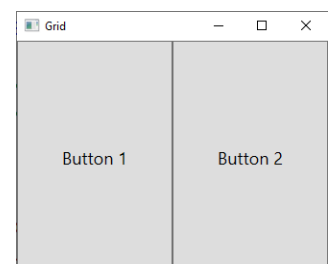
9.7 WPF-panelen

Panelen zijn één van de belangrijkste soorten WPF besturingselementen. Ze fungeren als containers voor andere besturingselementen en bepalen de lay-out van uw vensters. Denk bijvoorbeeld maar aan het Grid waarin we onze besturingselementen hebben ondergebracht.

Grid

Het raster is waarschijnlijk het meest complexe paneeltype. Een raster kan meerdere rijen en kolommen bevatten. Gebruik het raster wanneer je meerdere kolommen/rijen nodig hebt en vaak in combinatie met de andere panelen. De besturingselementen binnen een grid zijn volledig schaalbaar.

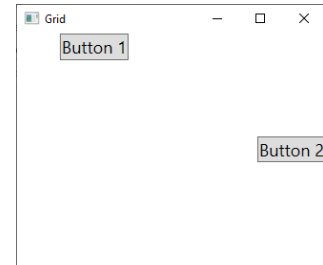
```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" /> * → proportionele grootte
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Button Content="Button 1"/>
    <Button Grid.Column="1" Content="Button 2"/>
</Grid>
```



```

<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Button VerticalAlignment="Top"
    HorizontalAlignment="Center">Button 1 </Button>
  <Button Grid.Column="1" VerticalAlignment="Center"
    HorizontalAlignment="Right">Button 2</Button>
</Grid>

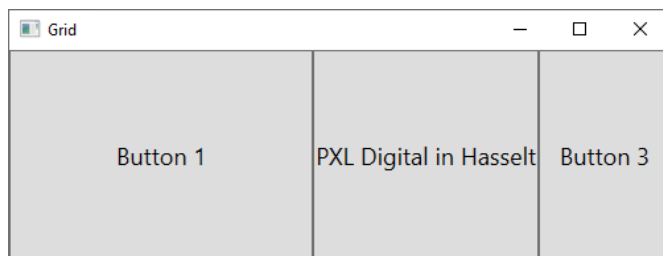
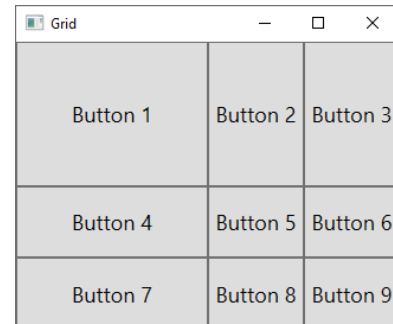
```



```

<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="2*" />
    <ColumnDefinition Width="1*" />
    <ColumnDefinition Width="1*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="1*" />
    <RowDefinition Height="1*" />
  </Grid.RowDefinitions>
  <Button>Button 1</Button>
  <Button Grid.Column="1">Button 2</Button>
  <Button Grid.Column="2">Button 3</Button>
  <Button Grid.Row="1">Button 4</Button>
  <Button Grid.Column="1" Grid.Row="1">Button 5</Button>
  <Button Grid.Column="2" Grid.Row="1">Button 6</Button>
  <Button Grid.Row="2">Button 7</Button>
  <Button Grid.Column="1" Grid.Row="2">Button 8</Button>
  <Button Grid.Column="2" Grid.Row="2">Button 9</Button>
</Grid>

```



```

<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="1*" />
    <ColumnDefinition Width="Auto" /> <!--Breedte past zich aan de content van button aan-->
    <ColumnDefinition Width="100" />
  </Grid.ColumnDefinitions>
  <Button>Button 1</Button>
  <Button Grid.Column="1">PXL Digital in Hasselt</Button>
  <Button Grid.Column="2">Button 3</Button>
</Grid>

```

```

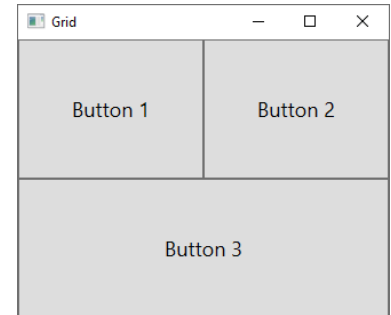
<Grid>

```

```

<Grid.ColumnDefinitions>
    <ColumnDefinition Width="1*" />
    <ColumnDefinition Width="1*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
<Button>Button 1</Button>
<Button Grid.Column="1">Button 2</Button>
<Button Grid.Row="1" Grid.ColumnSpan="2">Button 3</Button>
<!--Spanwijdte over 2 kolommen-->
</Grid>

```



```

<Grid >
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Label>Naam:</Label>
    <TextBox Grid.Column="1" Margin="0,0,0,10" />
    <Label Grid.Row="1">E-mail:</Label>
    <TextBox Grid.Row="1" Grid.Column="1" Margin="0,0,0,10" />
    <Label Grid.Row="2">Opmerkingen:</Label>
    <TextBox Grid.Row="2" Grid.Column="1" AcceptsReturn="True" />
</Grid>

```



Canvas

Het canvas is waarschijnlijk het eenvoudigste paneel van allemaal. Standaard doet het niet echt iets, je kunt er alleen besturingselementen in plaatsen en ze vervolgens zelf positioneren met behulp van expliciete coördinaten. Het paneel zal indien je de positie van het venster wijzigt dan ook niets doen.

```

<Canvas x:Name="CanTotaal" HorizontalAlignment="Left" Height="329" Margin="59,105,0,0"
VerticalAlignment="Top" Width="387" Background="{DynamicResource {x:Static
SystemColors.ControlDarkBrushKey}}">
    <Label Content="Totaal 2" HorizontalAlignment="Left" Height="35" VerticalAlignment="Top"
        Width="67" HorizontalContentAlignment="Right" Canvas.Left="12" Canvas.Top="18"/>
    <TextBox HorizontalAlignment="Left" Height="35" TextWrapping="Wrap" VerticalAlignment="Top"
        Width="57" Canvas.Left="95" Canvas.Top="18"/>
</Canvas>

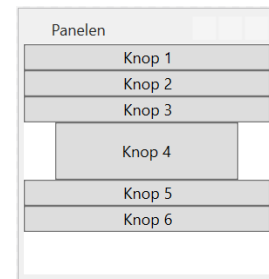
```

Stackpanel

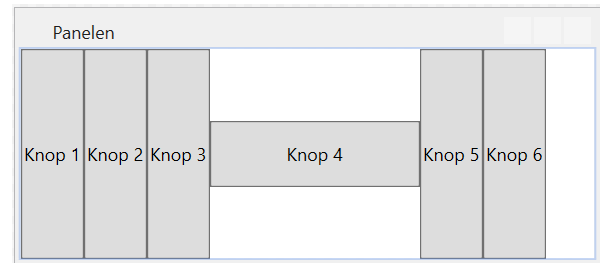
Het StackPanel rekt elk van de onderliggende besturingselementen over de volledige breedte of hoogte. De objecten krijgen bij een horizontale oriëntatie dezelfde hoogte (op basis van het hoogste item). Bij een verticale richting krijgen de objecten dezelfde breedte (op basis van het breedste item).

Gebruik het StackPanel als u een lijst met bedieningselementen wilt dat alle beschikbare ruimte in beslag neemt.

```
<StackPanel>
  <Button>Knop 1</Button>
  <Button>Knop 2</Button>
  <Button>Knop 3</Button>
  <Button Width="140" Height="44">Knop 4</Button>
  <Button>Knop 5</Button>
  <Button>Knop 6</Button>
</StackPanel>
```



```
StackPanel Orientation="Horizontal">
  <Button>Knop 1</Button>
  <Button>Knop 2</Button>
  <Button>Knop 3</Button>
  <Button Width="140" Height="44">Knop 4
</Button>
  <Button>Knop 5</Button>
  <Button>Knop 6</Button>
</StackPanel>
```

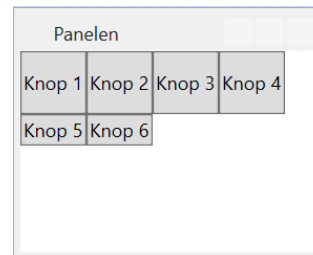


Wrappanel

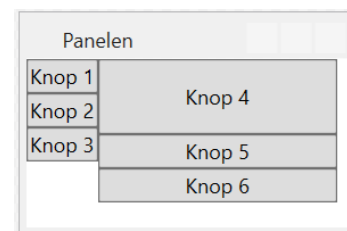
Het WrapPanel zal elk van zijn onderliggende besturingselementen naast elkaar plaatsen, horizontaal (standaard) of verticaal. Wanneer er geen plaats meer is dan wordt het volgende object naar de volgende regel verplaatst.

De objecten krijgen bij een horizontale oriëntatie dezelfde hoogte (op basis van het hoogste item). Bij een verticale richting krijgen de objecten dezelfde breedte (op basis van het breedste item).

```
<!--Standaard horizontale -->
<WrapPanel>
  <Button>Knop 1</Button>
  <Button>Knop 2</Button>
  <Button>Knop 3</Button>
  <Button Height="40">Knop 4</Button>
  <Button>Knop 5</Button>
  <Button>Knop 6</Button>
</WrapPanel>
```



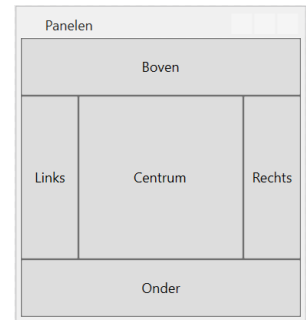
```
<WrapPanel Orientation="Vertical">
  <Button>Knop 1</Button>
  <Button>Knop 2</Button>
  <Button>Knop 3</Button>
  <Button Width="140" Height="44">Knop 4</Button>
  <Button>Knop 5</Button>
  <Button>Knop 6</Button>
</WrapPanel>
```



Dockpanel

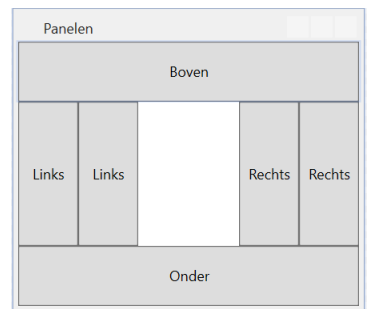
Met het DockPanel kan je objecten boven, onder, links of rechts tegen de rand koppelen. Het laatste element in het DockPanel vult de rest van de ruimte (midden). Standaard wordt het eerste object links gekoppeld en neemt het laatste object de resterende ruimte in.

```
<DockPanel>
    <Button DockPanel.Dock="Top" Height="50" Content="Boven"/>
    <Button DockPanel.Dock="Bottom" Height="50" Content="Onder"/>
    <Button DockPanel.Dock="Left" Width="50" Content="Links"/>
    <Button DockPanel.Dock="Right" Width="50" Content="Rechts"/>
    <Button Content="Centrum"/>
</DockPanel>
```



Als je de eigenschap LastChildFill uitschakelt, wordt het laatste object niet opgevuld.

```
<DockPanel LastChildFill="false">
    <Button DockPanel.Dock="Top" Height="50" Content="Boven"/>
    <Button DockPanel.Dock="Bottom" Height="50" Content="Onder"/>
    <Button DockPanel.Dock="Left" Width="50" Content="Links"/>
    <Button DockPanel.Dock="Right" Width="50" Content="Rechts"/>
    <Button DockPanel.Dock="Right" Width="50" Content="Rechts"/>
    <Button Width="50" Content="Links"/>
</DockPanel>
```



Viewbox

De ViewBox is een erg handig besturingselement in WPF. Hiermee kan je de inhoud aan de beschikbare grootte aanpassen. Het formaat van de inhoud wordt niet enkel aangepast maar het wordt ook geschaald (=transformatie).

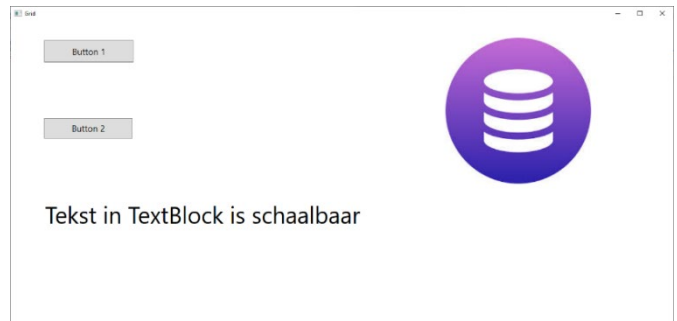
Hoewel het kan worden gebruikt voor elk type besturingselement, wordt het vaak gebruikt voor 2D-afbeeldingen of om een schaalbaar deel van een gebruikersinterface in een venster aan te passen.

Voorbeeld 1 Viewbox bevat het volledig grid

```
<Viewbox>
<Grid >
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="20" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
        <RowDefinition Height="20" />
    </Grid.RowDefinitions>
    <Label>Naam:</Label>
    <TextBox Grid.Column="1" Margin="0,0,0,10" MinWidth="300"/>
    <Label Grid.Row="1">E-mail:</Label>
    <TextBox Grid.Row="1" Grid.Column="1" Margin="0,0,0,10" />
    <Label Grid.Row="2">Opmerkingen:</Label>
    <TextBox Grid.Row="2" Grid.Column="1" AcceptsReturn="True" MinHeight="250" />
</Grid>
</Viewbox>
```


Voorbeeld 2 Viewbox voor bepaalde controls binnen het grid

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="98*" />
    <RowDefinition Height="97*" />
    <RowDefinition Height="138*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="316*" />
    <ColumnDefinition Width="271*" />
  </Grid.ColumnDefinitions>
  <Button Content="Button 1" HorizontalAlignment="Left" Height="45" Margin="69,39,0,0"
  VerticalAlignment="Top" Width="183" />
  <Button Content="Button 2" HorizontalAlignment="Left" Height="42" Margin="69,17,0,0"
  VerticalAlignment="Top" Width="181" Grid.Row="1" />
  <Viewbox Margin="16,34,25,27" Grid.Column="1" Grid.RowSpan="2">
    <Image Source="DB.jpg" />
  </Viewbox>
  <Viewbox VerticalAlignment="Center" Grid.Row="2" Grid.RowSpan="2">
    <TextBlock Text="Tekst in TextBlock is schaalbaar" />
  </Viewbox>
</Grid>
```



9.8 Selectievakje (Checkbox)

Het selectievakje stelt je in staat om verschillende keuzes aan te vinken.

```
<CheckBox x:Name="ChkWerkzoekend" Content="Werkzoekend" HorizontalAlignment="Left" Height="26"
Margin="55,150,0,0" VerticalAlignment="Top" Width="173" Click="ChkWerkzoekend_Click" />
```

- Eigenschappen: IsChecked: specificeert of het vakje geselecteerd is
FlowDirection: plaats van vakje (links of rechts)
- Events: Checked(): treedt op wanneer de eigenschap IsChecked *true* is
Click(): treedt op wanneer op object wordt geklikt
Unchecked(): treedt op wanneer de eigenschap IsChecked *false* is

9.9 Keuzerondje (Radiobutton)

Alle keuzerondjes op een formulier vormen automatisch een groep. Van de reeks keuzerondjes wordt 1 keuze geaccepteerd en ze sluiten elkaar wederzijds uit. Worden ze in een container gebruikt, dan vormen ze een aparte logisch groep.

```
<RadioButton x:Name="RadJongen" Content="Jongen" IsChecked="True" Checked="RadJongen_Checked"
Width="243" Canvas.Left="10" Canvas.Top="33" HorizontalAlignment="Center" VerticalAlignment="Center"
VerticalContentAlignment="Center" />
```

- Eigenschappen: IsChecked : specificeert of het keuzerondje geselecteerd is
FlowDirection: keuzerondje voor of na de tekst plaatsen.

- Events: Checked(): treedt op wanneer de eigenschap Checked verandert
Click(): treedt op wanneer op object wordt geklikt

9.10 Keuzelijst (Listbox)

Hiermee kan je de invoer van gegevens via een lijst voorzien.

```
<ListBox x:Name="LbxNamen">
    <ListBoxItem Content="Sander" />
    <ListBoxItem Content="Kristof"/>
    <ListBoxItem Content="Tom"/>
    <ListBoxItem Content="Paul" />
    <ListBoxItem Content="Patricia"/>
</ListBox>
```

- Eigenschappen: Items : items van de lijst invoeren
SelectionMode: *Single*: 1 item selecteren
Multi: meerdere items per keer selecteren (klikken op item)
Extended: meerdere items per keer (+Ctrl) of per groep (+Shift)
- Events: SelectedChanged(): opvragen van geselecteerde items (standaard event)
- Methods van ListBox
 - Add: voegt element toe aan einde van de listbox
 - Clear: wist alle elementen in de listbox
 - Contains: onderzoekt of het element in de listbox voorkomt
 - IndexOf: zoekt object in listbox
 - Insert: voegt element toe
 - Remove: verwijdert element
 - RemoveAt: verwijdert specifiek element

Voorbeelden

Opvragen van geselecteerde items (enkelvoudige selectie)

```
((ListBoxItem) ListBox1.SelectedItem).Content.ToString() Æ "Patricia"
LbxNamen.SelectedIndex Æ 1
```

Opvragen van geselecteerde items (meervoudige selectie)

```
foreach (ListBoxItem item in ListBox1.SelectedItems)
{
    TxtResultaat.Text += $"{item.Content}\n";
}
```

Items toevoegen

```
// Achteraan toevoegen.
ListBoxItem item = new ListBoxItem();
item.Content = TxtToevoegen.Text;
ListBox1.Items.Add(item);
```

```
// Op een positie toevoegen.
ListBox1.Items.Insert(index,item);
```

Items verwijderen

```
ListBox1.Items.Remove("element1")
ListBox1.Items.RemoveAt(2)
```

Listbox wissen

```
ListBox1.Items.Clear()
```

Zoeken naar index/item

```
TxtResultaat.Text = ListBox1.Items.Contains("Patricia"); // True: item bevindt zich in listbox
```

```
// Positie van element in listbox zoeken.
```

```
for (int i = 0; i < Listbox1.Items.Count; i++)
```

```
{
```

```
    if (((ListBoxItem)Listbox1.Items[i]).Content.ToString().Equals(TxtZoeken.Text))
```

```
    {
```

```
        index = i;
```

```
        //break;
```

```
    }
```

```
}
```

```
// ==== OF =====
```

```
int j = 0;
```

```
foreach (ListBoxItem item in LbxNamen.Items)
```

```
{
```

```
    if (item.Content.Equals(TxtZoeken.Text))
```

```
    {
```

```
        index = j;
```

```
        break;
```

```
    }
```

```
    j++;
```

```
}
```

9.11 Vervolgkeuzelijst (ComboBox)

Hiermee kan je de invoer van gegevens via een vervolgkeuzelijst voorzien.

```
<ComboBox x:Name="CboOpleiding" SelectionChanged="CboOpleiding_SelectionChanged">
    <ComboBoxItem Content="Programmeren"/>
    <ComboBoxItem Content="Netwerkbeheer"/>
    <ComboBoxItem Content="Internet of Things"/>
    <ComboBoxItem Content="Digitale vormgever"/>
    <ComboBoxItem Content="Drone opleiding"/>
</ComboBox>
```

- Eigenschappen: Items : items van de lijst invoeren via Enter
- MaxDropDownHeight: hoogte instellen waarbij aantal items zichtbaar worden
- IsEditable: invoer mogelijk en lijst uitschuiven (met klik of Alt+↓)
- IsReadOnly: enkel lijst uitschuiven
- IsTextSearchEnabled: tekst automatisch aanvullen Vb. D geeft dadelijk alle items die beginnen met een D... (niet samen met IsReadOnly = true)

Opvragen van geselecteerde items

```
((ComboBoxItem)CboOpleiding.SelectedItem).Content.ToString(); Æ "Netwerkbeheer"
```

```
CboOpleiding.SelectedIndex; Æ 1
```

NB. Keuzelijst leegmaken: CboOpleiding.SelectedIndex = -1;

Items toevoegen

```
ComboBoxItem item = new ComboBoxItem();  
item.Content = "Game designer";  
CboOpleiding.Items.Add(item);
```

Items verwijderen

```
CboOpleiding.Items.RemoveAt(CboOpleiding.SelectedIndex);
```

Combobox openen

```
CboOpleiding.IsDropDownOpen = true;
```

9.12 Schuifbalk (ScrollBar) /Schuifregelaar (Slider)

```
<Slider x:Name="Schuifregelaar" HorizontalAlignment="Left" Height="35" Margin="85,115,0,0"  
VerticalAlignment="Top" Width="395" TickPlacement="BottomRight" AutoToolTipPlacement="TopLeft"  
Maximum="100000" TickFrequency="10000" ValueChanged="Slider_ValueChanged" IsSnapToTickEnabled="True"  
SmallChange="100" LargeChange="5000"/>
```

- Eigenschappen: LargeChange: grote sprongen door klikken in schuifbalk/schuifregelaar
SmallChange: kleine sprongen door klikken op schuifregelaar/schuifbalk
Value: de waarde tussen minimum en maximum
Minimum: min. waarde
Maximum: max. waarde
Orientation: horizontale of verticale positie
TickFrequency: per eenheid het aantal streepjes op balk
TickPlacement: plaats van streepjes
AutoToolTipPlacement: plaats van automatische tooltip (geeft value)
IsSnapToTickEnabled: zorgt voor correcte value op eenheid (geeft geen waarde daartussen vb. 15.569853)
- Events: ValueChanged(): treedt op bij schuiven van de schuifbalk

9.13 Hoofdmenu (Menu)

```
<Menu x:Name="Hoofdmenu" HorizontalAlignment="Left" VerticalAlignment="Top" >
  <MenuItem Header="Bestand" >
    <MenuItem x:Name="MnuAfsluiten" Header="Afsluiten" Click="MnuAfsluiten_Click" />
  </MenuItem>

  <MenuItem x:Name="MnuAard" Header="Aard oefening" >
    <MenuItem x:Name="MnuOptellen" Header="Optellen" Click="MnuOptellen_Click" />
    <MenuItem x:Name="MnuAftrekken" Header="Aftrekken" Click="MnuAftrekken_Click" />
    <MenuItem x:Name="MnuVermenigvuldiging" Header="Vermenigvuldiging"
Click="MnuVermenigvuldiging_Click" />
    <MenuItem x:Name="MnuDeling" Header="Deling" Click="MnuDeling_Click" />
  </MenuItem>
```

- Eigenschappen: IsCheckable : vinkje blijft staan todat je er weer op klikt (geeft dan wel geen submenu weer)
ToolTip: tooltip bij menu-item
IsEnabled: actief of inactief (grijs) zetten
VerticalAlignment verticale positie van menu in windows/grid
HorizontalAlignment horizontale positie van menu in windows/grid

Hoofdstuk 10 Gegevens beheren met een array

Arrays worden gebruikt om grote hoeveelheden gegevens op te slaan of te verwerken. Vermits arrays niet anders zijn dan **geïndexeerde variabelen**, zijn de regels voor declaratie en bereik van variabele ook op arrays van toepassing.

1 Eindimensionale array *Zie eveneens handboek hoofdstuk 14 p351.*

1.1 Declaratie van arrays

```
int[] getallen = new int[5]; // 5 elementen van het type integer (index 0 tem index 4)
string[] namen = new string[6]; // 6 elementen van het type string (index 0 tem index 5)
int[] getallen; // Je kan er enkel meewerken als je later de grootte toekent.
```

1.2 Initialisatie

Arrays kunnen ook dadelijk geïnitieerd worden.

```
// 6 elementen van het type string (index 0 tem index 5)
string[] namen = new string[6] {"Patricia", "Silvia", "Paul", "Kathleen", "Ben", "Andreas"};
of string[] namen = new string[] {"Patricia", "Silvia", "Paul", "Kathleen", "Ben", "Andreas"};
of string[] namen = {"Patricia", "Silvia", "Paul", "Kathleen", "Ben", "Andreas"};
int[] getallen = {10, 20, 30, 40, 50};
```

```
for (int i = 0; i <= 4; i++)

{ getallen[i] = 10; } // 10 10 10 10 10
```

Let op!

```
//Declaratie
int[] getallen;
```

```
// Later initialiseren.
```

```
getallen = new int[] { 1, 3, 5, 7, 9 }; // Goed (zonder grootte dan expliciet met NEW werken)
getallen = { 1, 3, 5, 7, 9 }; // Fout
```

	getallen
[0]	10
[1]	20
[2]	30
[3]	40
[4]	50

1.3 Afdruk

```
for (int i = 0; i <= 5; i++)
{
    TxtResultaat.AppendText(namen[i]);
}
```

1.4 Ondergrens/bovgrens en lengte van een array bepalen

```
int[] getallen = new int[10]; à getallen.GetUpperBound(0) = 9 en getallen.GetLowerBound(0) = 0
getallen.GetLength(0) = 10
```

```
// Afdruk
for (int i = 0; i <= getallen.GetUpperBound(0); i++)
{
    Console.WriteLine(getallen[i]);
}
```

```
of for (int i = 0; i < getallen.GetLength(0); i++) // of getallen.Length-1 (aantal items in array)
{
    Console.WriteLine(getallen[i]);
}
```

1.5 Foreach (Enumeratielus)

De enumeratielus wordt gebruikt om items uit collection te **doorlopen/lezen**.

```
// Doorlopen van For Each
foreach (int item in getallen)
{ Console.WriteLine(item); }           // 1 3 5 7 9

foreach (int item in getallen)
{   i = item + 2;                       // NIET: item += 2; Je kan niet rechtstreeks toekennen.
    TxtResultaat.Text += $"{i} ";       // 3 5 7 9 11
}
```

2 Tweedimensionale array Zie eveneens handboek hoofdstuk 15 p379.

2.1 Declaratie

```
// Declaratie van 5 rijen en 2 kolommen
int[,] getal1 = new int[5,2];

// Declaratie van 2 rijen en 5 kolommen
int[,] getal2 = new int[2,5];
```

getal1	
	0 1
0	3 4
1	2 8
2	1 9
3	5 6
4	7 7

getal2					
	0	1	2	3	4
0	3	4	2	8	1
1	9	5	6	7	7

2.2 Initialisatie

```
// Initialisatie van 5 rijen en 2 kolommen
int[,] getal1 = new int[5,2] {{3, 4}, {2, 8}, {1, 9}, {5, 6}, {7, 7}};
of int[,] getal1 = new int[,] {{3, 4}, {2, 8}, {1, 9}, {5, 6}, {7, 7}};
of int[,] getal1 = {{3, 4}, {2, 8}, {1, 9}, {5, 6}, {7, 7}};

// Initialisatie van 2 rijen en 5 kolommen
int[,] getal2 = {{3, 4, 2, 8, 1}, {9, 5, 6, 7, 7}};
```

```
for (int r = 0; r <= 4; r++)
{
    for (int k = 0; k <= 1; k++)
    {
        getal1[r, k] = r * k;
    }
}
```

getal1	
	0 1
0	0 0
1	0 1
2	0 2
3	0 3
4	0 4

2.3 Afdruk

```
for (int r = 0; r <= 4; r++)
{
    for (int k = 0; k <= 1; k++)
    {
        TxtResultaat.AppendText($"{getal1[r, k]} ");
    }
    TxtResultaat.AppendText(" - ");           //=> 3 4 - 2 8 - 1 9 - 5 6 - 7 7 -
}
```

```
for (int r = 0; r <= 1; r++)
{
    for (int k = 0; k <= 4; k++)
    {
        TxtResultaat.AppendText($"{getal2[r, k]} ");
    }
    TxtResultaat.AppendText(" - ");           // = > 3 4 2 8 1 - 9 5 6 7 7 -
}
```

2.4 Ondergrens/bovgrens en lengte van een array bepalen

```
int[,] getallen = new int[10,20];
getallen.GetUpperBound(0) = 9   en getallen.GetLowerBound(0) = 0
getallen.GetUpperBound(1) = 19  en getallen.GetLowerBound(1) = 0
getallen.GetLength(0) = 10
getallen.GetLength(1) = 20
! getallen.Length = 200 (totaal aantal elementen in array)
```

3 Arrays en procedures

```
short[] cijfers = new short[10];
int res;
// Een functie geeft een array terug.
cijfers = IngelezenArray();

// Array wordt aan de funciemethode doorgegeven.
res = Som(cijfers);

// Array wordt aan een voidmethode doorgegeven.
ToonGetallenInLijst(cijfers);
```

```
private short[] IngelezenArray()
{
    short[] getallen = new short[10];
    string getalInput;

    for (int i = 0; i <= 9; i++)
    { getalInput = Interaction.InputBox("Geef een cijfer tussen 0 en 10: ");
      getallen[i] = short.Parse(getalInput);
    }
    return getallen;
}
```

```
private int Som(short[] arrGetal)
{
    int intSom = 0;
    // Som van de array-elementen
    foreach (short getal in arrGetal)
    {
        intSom += getal;
    }
    return intSom;
}

private void ToonGetallenInLijst(short[] arrGetal)
{
    // Toont de getallen in de keuzelijst lstCijfers
    foreach (short getal in arrGetal)
    {
        lstCijfer.Items.Add(getal);
    }
}
```


6 Members van de klasse System.Array

BinarySearch()	Zoekt met behulp van een 'binary' vergelijking naar een waarde in een 1-dimensionele array. De array moet wel gesorteerd zijn.
Clear()	Wist de inhoud van de array.
Copy()	Kopieert (eventueel een deel) van de array naar een andere array.
CreateInstance()	Maak een nieuwe instantie van de Array-class. Hiermee zou men eventueel de lowerbound van de array kunnen laten afwijken van 0; dit is echter af te raden om in de toekomst verwarring te voorkomen.
IndexOf()	Retourneert de index van de gezochte waarde die het eerst voorkomt in de array.
LastIndexOf()	Deze method lijkt erg op de IndexOf() method, het retourneert echter de laatst voorkomende index van een gezochte waarde.
Reverse()	Keert alle waarden in de array om. Wanneer u deze method direct aanroept na een Sort(), dan kan men dus ook aflopend sorteren.
Sort()	Sorteert (een deel van) de elementen in de array.

Voorbeelden

```
string[] namen = new string[] { "Patricia", "Silvia", "Paul", "Kathleen", "Ben", "Andreas", "Paul"};

// Eerste voorkomen zoeken.
Console.WriteLine(Array.IndexOf(namen, "Paul")); //2

// Laatste voorkomen zoeken.
Console.WriteLine(Array.LastIndexOf(namen, "Paul")); //6

// Array kopiëren.
string[] kopie = new string[namen.Length];
Array.Copy(namen, kopie, namen.Length);

// Array kopiëren: 3 elementen vanaf index 0 in NAMEN naar index 1 in KOPIE
string[] kopie2 = new string[4];
Array.Copy(namen, 0, kopie2, 1, 3); => null "Patricia" "Silvia" "Paul"

foreach (string element in kopie)
{
    Console.WriteLine(element); =>"Patricia" "Silvia" "Paul" "Kathleen" "Ben" "Andreas" "Paul"
}

// Array omkeren.
Array.Reverse(namen); //"Paul" "Andreas" "Ben" "Kathleen" "Paul" "Silvia" "Patricia"
```

```
// Array sorteren.
Array.Sort(kopie); //"Andreas" "Ben" "Kathleen" "Silvia" "Patricia" "Paul" "Paul"

// Een element zoeken in GESORTEERDE array. VLUIG!
Console.WriteLine(Array.BinarySearch(kopie, "Paul")); //5
```

```

// Array groter of kleiner maken. C# maakt intern een nieuwe array en doet copy.
int[] getal = new int[] {10, 5, 20};
Array.Resize(ref getal, 11); => 10 5 20 0 0 0 0 0 0 0 0
Array.Resize(getal, 11);      => 0 0 0 0 0 0 0 0 0 0 0
getal = new int[11];          => 0 0 0 0 0 0 0 0 0 0 0

// 1D Array wissen
int[] getal = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
// Vanaf index 2 de volgende 5 elementen wissen.
Array.Clear(getal, 2, 5); => 1 2 0 0 0 0 0 8 9

// 2D Array wissen
int[,] getallen = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
// Vanaf index 2 de volgende 5 elementen wissen.
Array.Clear(getallen, 2, 5); => 1 2 0 0 0 0 0 8 9

// Volledig wissen.
Array.Clear(getallen, 0, getallen.Length); => 0 0 0 0 0 0 0 0 0

// Array maken. Interessant wanneer je het type tijdens het compileren niet weet
// Een dimensionale array van type String (index: 0 ... 9) toekennen bij run-time
Array EenDimArray = Array.CreateInstance(typeof(string), 10);

// 2-dimensionale array van type Short (3 rijen: - 2 kolommen)
Array TweeDimArray = Array.CreateInstance(typeof(short), 3, 2);

```

```

// FIND (zoekt element) of EXIST (geeft True or False als element al dan niet bestaat)

string[] planeten = {"Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"};
//
// Zoekt eerste element beginnend met M.
//
string value1 = Array.Find(planeten, element => element.StartsWith("M")); → Mercury
//
// Zoekt eerste element dat 4 karakters lang is.
//
string value2 = Array.Find(planeten, element => element.Length == 4); → Mars
//
// Zoekt alle elementen groter dan 4 karakters lang en het wordt toegekend aan een array.
//
string[] array2 = Array.FindAll(planeten, element => element.Length >= 4);

Console.WriteLine("Is Pluto een planeet? {0}",
    Array.Exists(planeten, element => element == "Pluto")); → False

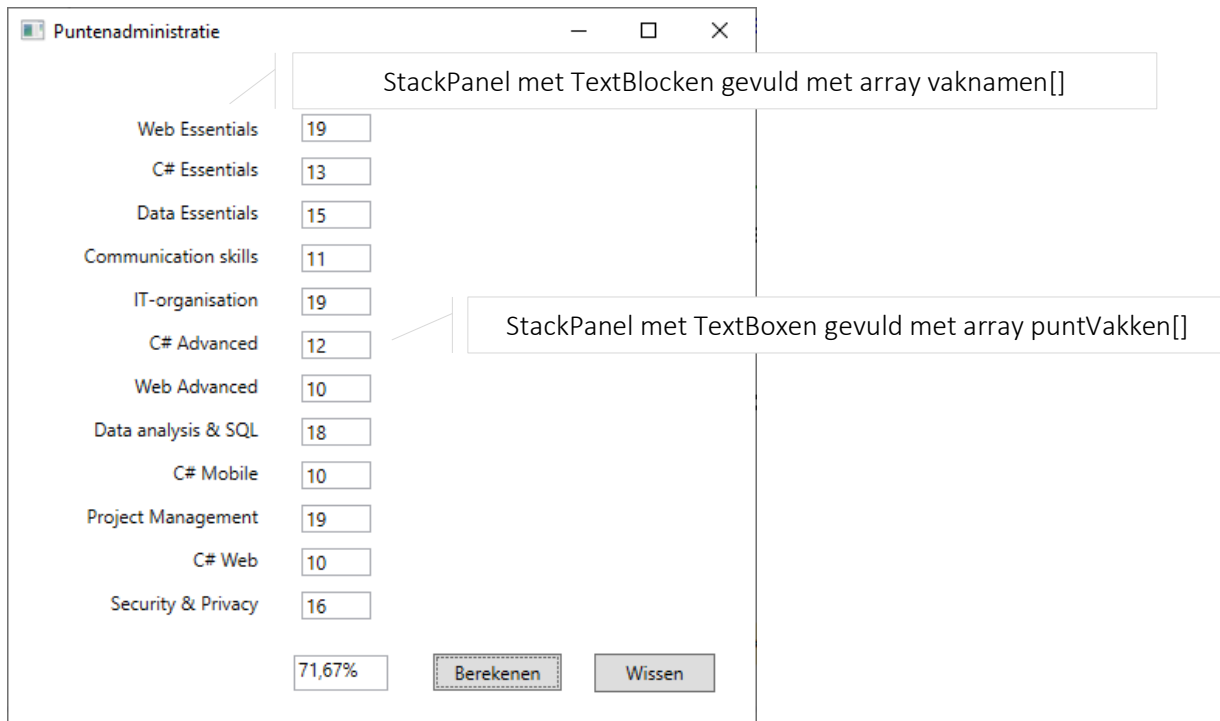
// string.Join geeft de array weer met de opgegeven seperator ","
Console.WriteLine(string.Join(",", array2)); → Mercury,Venus,Earth,Mars,
Jupiter,Saturn,Uranus,Neptune

```

7 Array van besturingselementen

Er kunnen eveneens arrays van controls gemaakt worden om met de verschillende elementen te werken.

In het voorbeeld hieronder wordt een array van tekstvakken aangemaakt in een StackPanel om de punten van de verschillende tekstvakken te kunnen berekenen. Vemits de volledige collectie van het StackPanel bestaat uit de tekstvakken die we gebruiken voor de array van tekstvakken *puntVakken[]*, wordt de collectie bij het opstarten van de toepassing eerst toegekend aan *puntVakken[]*. Later kan je *puntVakken[]* vlot overlopen om het behaald totaal te berekenen of tekst te wissen in alle elementen van de array.



```
private TextBox[] puntVakken = new TextBox[12];
private string[] vaknamen = { "Web Essentials", "C# Essentials", "Data Essentials", "Communication
skills", "IT-organisation", "C# Advanced", "Web Advanced", "Data analysis & SQL", "C# Mobile",
"Project Management", "C# Web", "Security & Privacy" };
```

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    Random rnd = new Random();

    // TextBlocken vullen met de namen van de vakken.
    int i = 0;
    foreach (TextBlock item in StackPanelVakken.Children)
    {
        item.Text = vaknamen[i];
        i++;
    }

    i = 0;
```

```

// Array van textboxes (met punten) vullen.
foreach (TextBox item in StackPanelPunten.Children)
{
    puntVakken[i] = item;
    puntVakken[i].Text = rnd.Next(8, 21).ToString();
    i++;
}

private void BtnBerekenen_Click(object sender, RoutedEventArgs e)
{
    int totaal=0;

    // === Totaal maken van alle punten. ===
    foreach (TextBox item in StackPanelPunten.Children)
    {
        totaal += int.Parse(item.Text);
    }

    TxtTotaal.Text = $"{totaal/240f:p2}";
}

private void BtnWissen_Click(object sender, RoutedEventArgs e)
{
    // Alle tekstvakken op "0" zetten.
    foreach (TextBox item in puntVakken)
    {
        item.Text = "0";
    }

    // Tesktvak Totaal leegmaken.
    TxtTotaal.Clear();
}

```

Hoofdstuk 11 Generieke listcollectie

List<T>

Arrays zijn dus een handig middel om elementen een waarde te geven maar er zijn ook *nadelen* aan.

- Zo kan je een array niet van grootte veranderen (tenzij je werkt met de klasse Array).
- Wil je een waarde verwijderen of toevoegen, dan moet je een kopie maken en elke waarde opschuiven.

Door gebruik te maken van List<> uit de klasse **System.Collections.Generic**, kan je deze nadelen omzeilen.

Een generieke constructie maakt het mogelijk eenzelfde functionaliteit toe te passen op verschillende datatypes. Indien je een generieke klasse of method definieert, hoef je geen aparte versie te maken voor elk datatype waarop dezelfde functionaliteit moet worden toegepast.

Methods

Add	voegt element toe aan einde List
AddRange	List/Array toevoegen aan een andere List
Clear	wist alle elementen van List
Count	telt het aantal elementen in de List
ToArray	kopieert List naar een ééndimensionale array
GetRange	specificeert een bereik
IndexOf	zoekt de positie van een item in de List
Insert	voegt element toe
Remove	verwijdert element
RemoveAt	verwijdert specifiek element
Reverse	draait de volgorde van de elementen om
Sort	sorteert de elementen in de List

Voorbeeld1

```
using System.Collections.Generic;

int[] getallen = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };
List<int> waarden = new List<int>(); // bekhaakjes gebruiken!

// Waarden toevoegen
foreach (int item in getallen)
{ waarden.Add(item); } // voegt de waarden toe aan de List (List verhoogt Count automatisch)

// === OF === Array toevoegen (resultaat idem als aparte elementen toevoegen)
waarden.AddRange(getallen);

// Afdruk in ListBox
ListBox1.Items.Clear();
for (int i=0; i < waarden.Count; i++)
{
    ListBox1.Items.Add(waarden[i].ToString()); // 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
}

// voegt een waarde op de 4de plaats (index 3) toe
waarden.Insert(3, 200); // 10, 20, 30, 200, 40, 50, 60, 70, 80, 90, 100
```

```
//wis de waarde met index 8
waarden.RemoveAt(8); // 10, 20, 30, 200, 40, 50, 60, 70, 90, 100

// OF wis het item met waarde 80
waarden.Remove(80);

// List toekennen aan een array.
int[] array = waarden.ToArray(); // 10,20,...,100
int[] output = waarden.GetRange(2, 3).ToArray(); // 30,200,40

// List toevoegen aan een andere List (achteraan)
List<int> priemgetallen = new List<int>(new int[] { 2, 3, 5, 7, 11, 13, 17 });
waarden.AddRange(priemgetallen); // list toevoegen (achteraan)
waarden.AddRange(getallen); // array toevoegen (achteraan)
```

Voorbeeld2

```
string[] voornamen = { "Wouter", "Paul", "Andreas", "Niels", "Kathleen", "Paul", "Silvia",
"Patricia" };
List<string> waarden = new List<string>(voornamen); // Grootte is vast, bij Add: Range vergroten!
// zonder initialisatie: automatisch vergroten
// (zie eerste voorbeeld)

// waarden sorteren
waarden.Sort(); // Andreas, Kathleen, Niels, Patricia, Paul, Paul, Silvia, Wouter

//eerste 3 elementen gesorteerd
waarden.Sort(0, 3, null); // Andreas, Paul, Wouter, Niels, Kathleen, Paul, Silvia, Patricia
// null geeft default comparer aan (QuickSort, HeapSort,...)

//Zoekt naar positie.
waarden.IndexOf("Lise"); // -1 niet gevonden
waarden.IndexOf("Paul", 3); // vanaf positie 3 zoeken = 5 eerste voorkomen
waarden.IndexOf("Paul", 2, 2); // -1 niet gevonden
```

N.B. Oudere versies werkten met een ArrayList die dezelfde werking/mogelijkheden heeft als een List, maar heeft de problematiek van boxing (impliciete conversie van value types zoals int, float,.. naar een *object*). Best vermijden!

Dictionary<T>

De Dictionary klasse slaat de elementen in de vorm van een *sleutelpaar* bestaande uit **2 elementen** op. Met de sleutel (key) wordt de waarde(value) toegevoegd en met de sleutel kan je waarden ook terug opzoeken of verwijderen. De sleutels en waarden kunnen een willekeurig datatype krijgen.

Voorbeeld

```
Dictionary<string, int> punten = new Dictionary<string, int>();

// Toevoegen van elementen aan dictionary bestaande uit de Key en Value.
punten.Add("Anse", 17);
punten.Add("Lise", 12);
punten.Add("Sander", 15);
punten.Add("Ayko", 9);
punten.Add("Bram", 16);

// Collectie overlopen.
Console.WriteLine($"Punten overzicht\r\n");
```

```

foreach (var item in punten)
{
    Console.WriteLine($"{item.Key}: {item.Value}/20");
}

// Element opzoeken.
Console.WriteLine("\r\nZoek het resultaat van: ");
string naam = Console.ReadLine();
Console.WriteLine($"{naam} behaalde {punten[naam]}"); // Bram behaalde 16

// Of een element opzoeken met de TryGetValue().
punten.TryGetValue(naam, out int resultaat); // geeft eveneens 16

// Element verwijderen.
Console.WriteLine();
Console.WriteLine("\r\nVerwijder het resultaat van: "); // Lise
naam = Console.ReadLine();
punten.Remove(naam);

// Onderzoeken of element bestaat.
string resultaat=( punten.ContainsKey(naam))?"niet verwijderd":"verwijderd";

Console.WriteLine($"{naam} is {resultaat}");
Console.ReadLine();

```

```

--- Afdruk ---

Punten overzicht

Anse: 17/20
Lise: 12/20
Sander: 15/20
Ayko: 9/20
Bram: 16/20

Zoek het resultaat van: Bram
Bram behaalde 16

Verwijder het resultaat van: Lise
Lise is verwijderd

```

```

// === Omzetten naar List.===
List<KeyValuePair<string, int>> list = dictionary.ToList(); // ToList() → using System.Linq;

// Afdruk list.
foreach (KeyValuePair<string, int> item in list)
{
    Console.WriteLine($" {item.Key} behaalde {item.Value}");
}

```

```

Anse behaalde: 17
Lise behaalde: 12
Sander behaalde: 15
Ayko behaalde: 9
Bram behaalde: 16

```

```

// Omzetten naar Dictionary.

```

```

// Nieuwe list.
List<string> nLijst = new List<string>();
nLijst.Add("Programmeren");
nLijst.Add("Netwerkbeheer");
nLijst.Add("Internet of things");
nLijst.Add("Digitale vormgeving");

// List naar Dictionary.
Dictionary<string, int> nDict = new Dictionary<string, int>();
int i = 750;
foreach (string item in nLijst)
{
    nDict.Add(item, i);
    i += 100;
}

// Afdruk dictionary.
foreach (var item in nDict)
{
    Console.WriteLine(item);
}

```

```

[Programmeren, 750]
[Netwerkbeheer, 850]
[Internet of things, 950]
[Digitale vormgeving, 1050]

```

Tip

Gebruik een Dictionary<> als je vaak gegevens moet **opzoeken**. Vooral bij veel gegevens werkt de List<> trager. Moet je echter gewoon gegevens doorlopen dan gebruik je best de List<>.

Hoofdstuk 12 Fouten opsporen en de gestructureerde foutafhandeling

Bij het maken van een programma worden fouten gemaakt.

Er zijn verschillende soorten fouten of bugs:

- Syntaxisfouten
- Logische fouten
- Runtime-fouten

De *syntaxisfouten* worden grotendeels opgevangen tijdens het programmeren (gekleurde golfjes).

De *logische fouten* kunnen opgevangen worden door het programma één per één te bekijken tijdens de uitvoering.

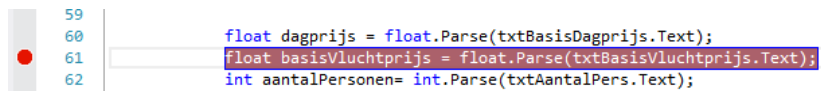
De *runtime-fouten* worden opgevangen door de gestructureerde foutafhandeling (Try Catch).

14.1 Debuggen in onderbrekingsmodus

Een breakpoint kan je gebruiken om in je code even te pauzeren zodat je kan zien welke waarden het programma heeft. Het is een efficiënte manier om je code te testen en analyseren.

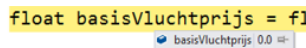
- **Breakpoint instellen**

Zet een breakpoint in je code door op de grijze balk (voor het regelnummer) te klikken of kies in het menu Debug voor 'Toggle Breakpoint'.



Start je toepassing of doorloop je programma met F11. Het punt waar de debugger is aangekomen wordt getoond met een rode bol met een gele pijl.

Als je met je muis boven eender welke variabele even blijft hangen dan krijg je te zien welke waarde de variabele momenteel heeft.



Merk op dat bovenaan in je Visual Studio een balkje is bijgekomen.




- **Programma doorlopen**

De gele pijl in de grijze balk geeft aan waar je in je code bent, maar je kan met de gele pijl ook terug in je code gaan of verder (maar overgeslagen code wordt niet uitgevoerd).

Step Into	F11
Step Over	F10
Step Out	Shift+F11

Met *Step Over* (F10) kan je het programma stap voor stap doorlopen. De knoppen *Step Into* en *Step Out* worden gebruikt voor methods. Met *Step Out* kan je uit de method springen en met *Step Over* wordt de procedure volledig uitgevoerd.

Met Run To Cursor  (Ctrl+F10) kan je een stuk code uitvoeren tot de huidige positie van de cursor.

- **Vensters Autos en Watch**



Met het venster Watch kan je de inhoud van de *zelf ingestelde* lokale variabelen en eigenschappen bekijken gedurende de volledige uitvoering van het programma. De waarde in het rood geeft de laatste wijziging aan.

Name	Value	Type
vluchtprijs	640.0	float
dagprijs	60.0	float
aantalDagen	7	int
verblijfsprijs	1176.0	float
aantalPersonen	4	int
teBetalen	0.0	float



Het venster Autos geeft de waarde en de eigenschappen van het huidig statement en het vorig statement. Rood is de laatste waardeverandering in de actieve procedure.

Name	Value	Type
korting	90.8	float
reispreis	1816.0	float
teBetalen	0.0	float
this	{reiskost2.frmReiskost2, Text: }	reiskost2.frmReiskost2
txtKortingspercentage	{System.Windows.Forms.TextBox, Text: 5}	System.Windows.Forms.TextBox



Het venster Locals geeft alle lokale variabelen weer gedurende de uitvoering van het programma.

Name	Value	Type
this	{WpfReiskost2.MainWindow}	WpfReiskost2.MainWindow
sender	{System.Windows.Controls.Button: Berekenen}	object {System.Windows.Controls.Button}
e	{System.Windows.RoutedEventArgs}	System.Windows.RoutedEventArgs
vluchtprijs	640	float
verblijfsprijs	1176	float
reispreis	1816	float
teBetalen	1725.2	float
testBasis	true	bool
dagprijs	60	float
testVlucht	true	bool
basisVluchtprijs	200	float
testPersonen	true	bool
aantalPersonen	4	int
testDagen	true	bool
aantalDagen	7	int
testKorting	true	bool
korting	90.8	float



Het venster Immediate kan je tijdens het debuggen gebruiken om expressies of waarden van variabelen op te vragen. Zoals je in onderstaande venster kan zien, kan je tijdens de uitvoering van een programma ook de waarde van expressies, variabelen en functies wijzigen.

```
Immediate Window
? reiskost
error CS0103: The name 'reiskost' does not exist in the current context

? reisprijs
1816

reisprijs = 1500
1500

? reisprijs
1500
```

14.2 Exceptions

Een *runtime-fout* is een software- of hardware probleem dat tot gevolg heeft dat een programma niet goed meer werkt. Als er een runtime-fout optreedt, kunnen er gegevens verloren gaan in het bestand waarmee u werkt. Runtime-fouten kunnen ook fouten in het bestand veroorzaken, waardoor het bestand beschadigd raakt, zodat u er niet meer mee kunt werken.

De *runtime-fouten* worden opgevangen door de gestructureerde foutafhandeling (Try Catch).

Je kan je programmacode schrijven zodat die specifieke runtimefouten (**uitzonderingen of exceptions**) worden afgehandeld. Deze routines worden **gestructureerde errorhandlers of gestructureerde exceptionhandlers** genoemd.

```
try
{
    'Beschermd code'
    Statements die een runtimefout veroorzaken
}
catch
{
    'ErrorHandler of exceptionhandler'
    Statements die uitgevoerd worden bij een runtimefout
}
finally
{
    Optionele statements die altijd uitgevoerd worden
}
```

1 Try ... Catch

Errorhandlers worden in de eventprocedures geplaatst waarin mogelijk problemen kunnen optreden. Ze worden dus in de eventprocedure rond de code gebouwd en worden ook beschermende code genoemd omdat het niet leidt tot een programmacrash.

Vb. bij netwerkverbindingen, internetverbindingen, printer offline,,...

```
try
{
    ImgAfb.Source = new BitmapImage(new Uri(bestand, UriKind.Absolute));
}
catch (Exception) // foutafhandeling
{
    MessageBox.Show("De afbeelding staat niet in het aangegeven pad.");
}
```

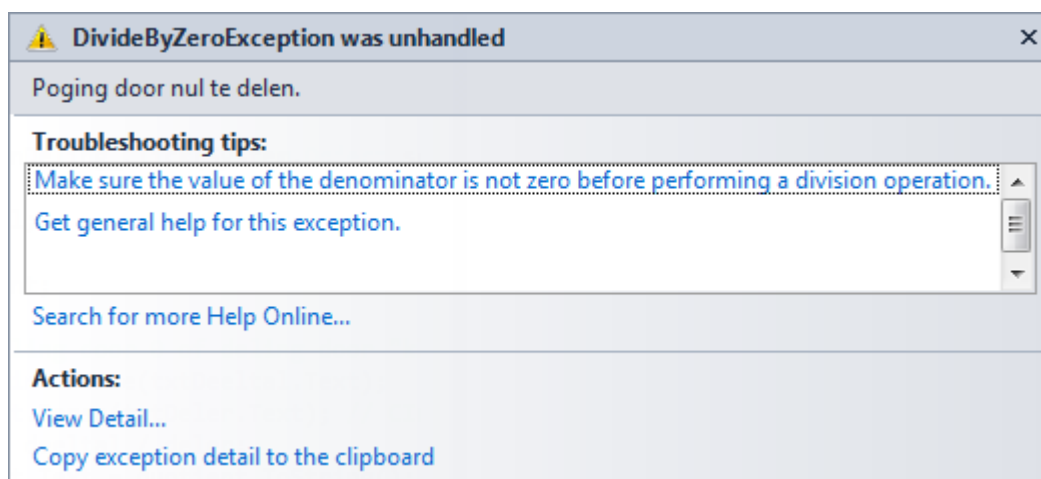
2 Finally

De finally- code wordt altijd uitgevoerd. Ze worden gebruikt om eigenschappen bij te werken, een berichtvenster te geven of andere 'opruimwerkzaamheden' te verrichten.

```
try
{
    ImgSeizoen.Source = new BitmapImage(new Uri(bestand, UriKind.Absolute));
}
catch (Exception) // foutafhandeling
{
    MessageBox.Show("De afbeelding staat niet in het aangegeven pad.");
}
finally
{
    MessageBox.Show("ErrorHandler is afgehandeld.");
}
```

3 Specifieke runtimefouten opvangen

Wanneer een exception optreedt worden de Catch-blokken één per één doorlopen. **Het Catch-blok waarvan het type exception overeenstemt met de exception wordt uitgevoerd.** Indien een exception voorkomt dat niet gelijk is aan één van deze types wordt het Catch-blok zonder type exception doorlopen. Het type exception wordt in de titelblak van het dialoogvenster weergegeven. *View Detail* geeft een gedetailleerde omschrijving van de foutmelding.



```

try
{
    deeltal = Convert.ToInt32(TxtDeeltal.Text);
    deler = Convert.ToInt32(TxtDeler.Text);
    quotient = deeltal / deler;
    TxtQuotient.Text = quotient.ToString();
}
catch (DivideByZeroException) // Deling door 0
{
    MessageBox.Show("De deler mag niet 0 zijn.", "FOUT BIJ DELING", MessageBoxButtons.OK,
        MessageBoxIcon.Stop);
}
catch (FormatException ex) // Deling door "k" of "1.6" CONVERSIEFOUT
{
    MessageBox.Show("Je moet 2 gehele getallen ingeven.\r\n\r\n" + ex.Message, "Conversiefout bij
        DELER/DELTAL", MessageBoxButtons.OK, MessageBoxIcon.Stop);
}
catch (OverflowException ex) // 9999999999 voor int te groot!
{
    MessageBox.Show(ex.Message, "Het getal moet tussen - 2 147 483 648 tot 2 147 483 647 liggen.");
}
catch (Exception ex) // Application fout
{
    MessageBox.Show("Er is een onverwachte fout opgetreden.\r\n\r\n" + ex.Message, "Conversiefout
        bij DELER/DELTAL", MessageBoxButtons.OK, MessageBoxIcon.Stop);
}
}

```

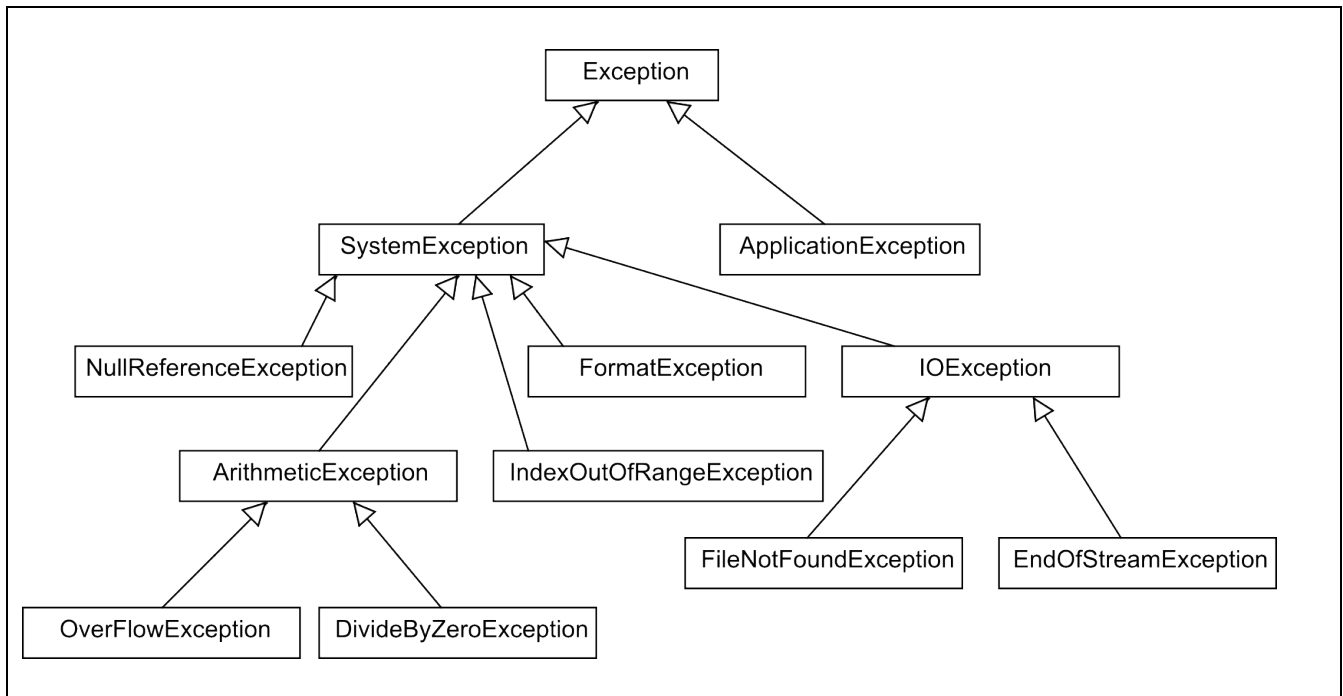


Door gebruik te maken van een variabele ex kan je de foutmelding gegenereerd door het systeem gebruiken. VB. *ex.Message* geeft 'De indeling van de invoertekenreeks is onjuist.'

Let op!

Eerst moet je de meest specifieke fouten *vóór* de algemene fout opvangen omdat je anders altijd een algemene fout bekomt.

Classificatie klassenhiërarchie



4 Geneste blokken Try ... Catch gebruiken

Door het gebruik van geneste blokken kan een tweede foutafhandeling vermeden worden. Vb. de gebruiker krijgt de kans om bij een foutmelding de disc in drive D te plaatsen.

```
try
{
    ImgAfb.Source = new BitmapImage(new Uri(bestand, UriKind.RelativeOrAbsolute));
}
catch (System.IO.FileNotFoundException)
{
    MessageBox.Show("Please insert the disc in drive D");
    try
    {
        // Gebruiker kan de disc in drive D leggen.
        picCSharp.Image = Image.FromFile("D:\\Mijn documenten\\CSharp.png");
    }
    catch (System.IO.FileNotFoundException)
    {
        MessageBox.Show("Knop Tonen uitgeschakeld.");
    }
}
```

5 Zelf een runtime-fout genereren

Je kan zelf een runtime-fout genereren om gegevens te testen of uitzonderlijke exceptions op te vangen.

```
private void BtnThrow_Click(object sender, EventArgs e)
{
    try
    {
        MessageBox.Show(Convert.ToString(WordToNumber("hXndred")));
    }
    catch (FormatException ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private int WordToNumber(string woord)
{
    int result = 0;
    if (woord == "ten")
    {
        result = 10;
    }
    else if (woord == "hundred")
    {
        result = 100;
    }
    else if (woord == "thousand")
    {
        result = 1000;
    }
    else
    {
        throw new FormatException("Verkeerde invoer: " + woord); } // gaat naar Catch
    return result;
}
```

Je kan verschillende vormen gebruiken:

- `throw new FormatException();`
- `throw new FormatException(string);`
- `throw new FormatException(string,exception);`

6 Defensief programmeren

De beste manier om te programmeren is echter het defensief programmeren. Dus zorgen dat een runtimefout niet kan voorkomen door gebruik te maken van zogenaamde **validatietechnieken** in het .NET framework. Als een uitzondering of runtimefout betrekkelijk zeldzaam is (bijvoorbeeld minder dan 25 %) is de errorhandler waarschijnlijk de beste aanpak.

```
if (File.Exists(pad))
{
    ImgAfb.Source = new BitmapImage(new Uri($"Afbeelding\\{afb}", UriKind.Absolute));
}
else
{
    throw new UriFormatException();
}
```

Hoofdstuk 13 Console toepassingen

Dit soort programma's kent geen vensters, labels, tekstvelden e.d., waardoor programma's kleiner zijn en je je beter kan concentreren op de syntax van de taal. Voor het testen van programma-onderdelen is deze toepassing uiterst geschikt om te gebruiken.

1 Opbouw van een console-toepassing

Een console-toepassing bestaat uit vier stappen, waarvan de eerste stap niet altijd hoeft voor te komen en de vierde stap alleen nodig is om het resultaat even te kunnen bekijken (sluit automatisch af).

- 1 invoeren van de gegevens
- 2 verwerken van de gegevens
- 3 tonen van het resultaat
- 4 wachten op de Enter-toets

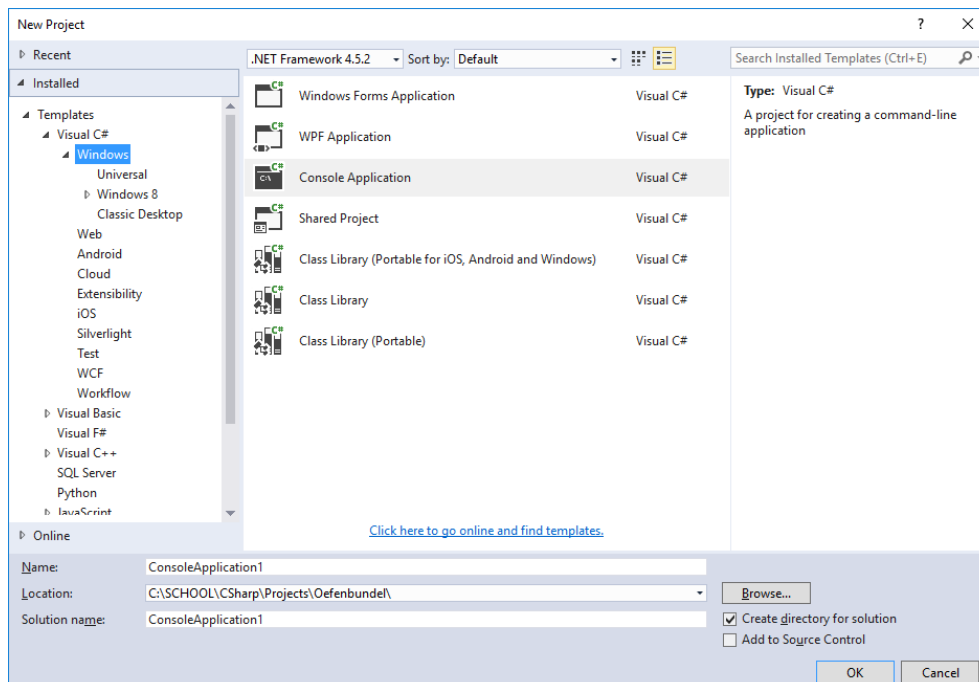
2 De klasse Console

De klasse Console kent zeer veel methoden, maar we gebruiken er meestal maar vier:

- Console.Read: gegevens van het toetsenbord inlezen.
- Console.ReadLine: gegevens van het toetsenbord inlezen en daarna verder gaan op de volgende regel.
- Console.Write: gegevens op het scherm afdrukken.
- Console.WriteLine: gegevens op het scherm afdrukken en verder gaan op de volgende regel.

3 Nieuw project maken

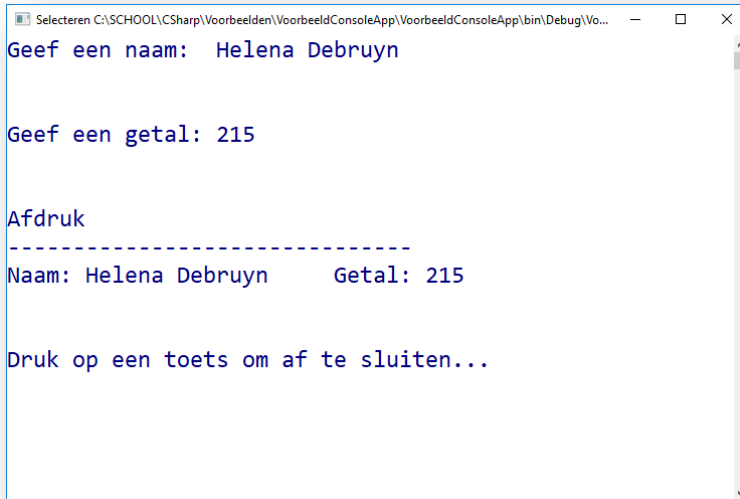
- Selecteer in de lijst met sjablonen de optie Console Application



3 Enkele voorbeelden

Consoletoepassing 1

Een naam en getal inlezen en afdrukken.



```
Geef een naam: Helena Debruyn

Geef een getal: 215

Afdruk
-----
Naam: Helena Debruyn    Getal: 215

Druk op een toets om af te sluiten...
```

```
static void Main(string[] args)
{
    string naam;
    int getal;

    // Inlezen.
    Console.Write("Geef een naam: ");
    naam = Console.ReadLine(); // leest tot ENTER
    Console.WriteLine();
    Console.WriteLine();
    Console.Write("Geef een getal:\t");
    getal = int.Parse(Console.ReadLine());
    Console.WriteLine();
    Console.WriteLine();

    // Afdruk.
    Console.WriteLine("Afdruk");
    Console.WriteLine("-----");
    Console.WriteLine($"Naam: {naam}\t Getal: {getal}");
    Console.WriteLine();
    Console.WriteLine();

    Console.WriteLine("Druk op enter om af te sluiten...");
    Console.ReadLine();
}
```

Consoletoepassing 2

Druk af op iemand meer of minder dan 10 000 dagen oud is.



```
static void Main(string[] args)
{
    DateTime geboortedatum, vandaag = DateTime.Now;
    long aantalDagen;

    Console.Write("Geef je geboortedatum (dd/mm/yyyy): ");
    geboortedatum = DateTime.Parse(Console.ReadLine());
    aantalDagen = vandaag.Subtract(geboortedatum).Days;
    Console.WriteLine();

    if(aantalDagen >= 10000 )
    {
        Console.WriteLine($"Je bent al meer dan 10 000 dagen op deze aardbol ({aantalDagen:n0} dagen).");
    }
    else
    {
        Console.WriteLine($"Je bent minder dan 10 000 dagen op deze aardbol ({aantalDagen:n0} dagen).");
    }

    Console.WriteLine();
    Console.WriteLine();
    Console.WriteLine("Druk op een toets om af te sluiten....");
    Console.ReadLine();
}
```

Bijlagen

B1 Naamconventies in C#

C #-naamgevingsconventies zijn een belangrijk onderdeel van de C #-coderingsstandaarden en best practices bij het ontwikkelen van .NET-toepassingen. .NET-naamgevingsconventies zijn standaarden voor het definiëren van de naamgeving van variabelen, methoden, klassen en andere code-elementen.

Er worden de volgende terminologieën gebruikt om C #- en .NET-naamgevingsstandaarden te declareren.

- **Camel Case (camelCase):** In deze standaard wordt de eerste letter van het woord altijd in kleine letters weergegeven en daarna begint elk woord met een hoofdletter.
- **Pascal Case (PascalCase):** hierin is de eerste letter van elk woord een hoofdletter

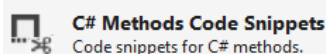
Identifier	Casing	Voorbeeld
Namespace	Pascal	namespace System.Text { ... }
Class	Pascal	public class LeesBestand { ... }
Interface	Pascal	public interface IEnumerable { ... }
Method	Pascal	private int Berekening() { ... }
Property	Pascal	public string Naam { get { ... } }
Event	Pascal	public event EventHandler Click ;
Public variable	Pascal	private int Salaris ;
Public Const	Pascal	public const int Min = 0;
Private variable	Camel	private string naam ;
Enum Value	Pascal	public enum FileMode { Append , ... }
Parameter	Camel	private int Converteer(string waarde);

B2 Lijst van Code Snippets in C#

#if	Code snippet for #if
#region	Code snippet for #region
attribute	Code snippet for attribute using recommended pattern
checked	Code snippet for checked block
class	Code snippet for class
ctor	Code snippet for constructor
~	Code snippet for destructor
cw	Code snippet for Console.WriteLine
do	Code snippet for do...while loop
else	Code snippet for else statement
enum	Code snippet for enum
equals	Code snippet for implementing Equals() according to guidelines
exception	Code snippet for exception
for	Code snippet for 'for' loop
foreach	Code snippet for foreach statement
forr	Code snippet for reverse 'for' loop
if	Code snippet for if statement
indexer	Code snippet for indexer
interface	Code snippet for interface
invoke	Code snippet for safely invoking an event
iterator	Code snippet for a simple iterator
iterindex	Code snippet for 'named' iterator/indexer pair using a nested class
lock	Code snippet for lock statement
mbox	Code snippet for MessageBox.Show
namespace	Code snippet for namespace
prop	Code snippet for an automatically implemented property
propg	Code snippet for an automatically implemented property with a 'get' access or and a private 'set' accessor
sim	Code snippet for int Main()
struct	Code snippet for struct
svm	Code snippet for 'void Main' method
switch	Code snippet for switch statement
try	Code snippet for try catch
tryf	Code snippet for try finally
unchecked	Code snippet for unchecked block
unsafe	Code snippet for unsafe statement
using	Code snippet for using statement
while	Code snippet for while loop

Gebruik het sleutelwoord en druk 2x keer op TAB om de structuur te verkrijgen.

Indien je de lijst van snippets wenst uit te breiden kan je via Extensions – Manage Extensions bijkomende tool downloaden. Kies onderstaande tool om bijvoorbeeld te werken met snippets voor methods.



B3 Lijst van Class library in C#

Deze verzameling classes is ingedeeld in namespaces.

Hieronder enkele van de belangrijkste namespaces:

- **System.Data**

Classes om databases (SQL Server, Oracle, Access, ...) te gebruiken.

- **System.XML**

Classes om XML (uitwisselbaar dataformaat tussen de meest voorkomende computers en programmeertalen) te gebruiken.

- **System.Diagnostics**

Classes om fouten in je programma op te sporen.

- **System.DirectoryServices**

Classes om Active Directory Services te gebruiken.

- **System.Messaging**

Classes om boodschappen te lezen en te verzenden over netwerken.

- **System.Timers**

Classes om op geregelde tijdstippen (iedere dag, om de minuut, ...) code uit te voeren.

- **System.Globalization**

Classes om taal- en cultuurverschillen in je toepassing te verwerken

(wij schrijven bvb. een datum als dag/maand/jaar, in de USA is dit maand/dag/jaar).

- **System.Net**

Classes om netwerken, het internet en hun protocollen te gebruiken.

- **System.Collections**

Classes om verzamelingen te maken die meer functionaliteit aanbieden dan arrays: lists, queues, arraylists, hashtables, dictionaries.

- **System.Collections.Generic**

Zelfde als bij System.Collections maar dan voor generic lists.

- **System.IO**

Classes om bestanden en directories te gebruiken.

- **System.Text**

Classes om verschillende karaktercoderingen (ANSI, Unicode, ...) te gebruiken.

- **System.Threading**

Classes om multithreading (gelijktijdige uitvoering van code) in je toepassing te gebruiken.

- **System.Reflection**

Classes om tijdens de uitvoering van het programma de eigenschappen en methods van Classes op te vragen.


- **System.Drawing**

Classes om punten, lijnen, rechthoeken, cirkels, afbeeldingen, tekst, ... te tekenen.

- **System.Windows.Forms**

Classes om Windows vensters in je programma te gebruiken met in die vensters alle populaire controls: tekstvakken, knoppen, menu's, ...

B4 Color table in C#

	AliceBlue	#FFF0F8FF		DarkTurquoise	#FF00CED1		LightSeaGreen	#FF20B2AA		PapayaWhip	#FFFEEFD5
	AntiqueWhite	#FFFAEBD7		DarkViolet	#FF9400D3		LightSkyBlue	#FF87CEFA		PeachPuff	#FFFDDAB9
	Aqua	#FF00FFFF		DeepPink	#FFFF1493		LightSlateGray	#FF778899		Peru	#FFCD853F
	Aquamarine	#FF7FFFD4		DeepSkyBlue	#FF00BFFF		LightSteelBlue	#FFB0C4DE		Pink	#FFFC0CB
	Azure	#FFF0FFFF		DimGray	#FF696969		LightYellow	#FFFFFFF0		Plum	#FFDDA0DD
	Beige	#FFF5F5DC		DodgerBlue	#FF1E90FF		Lime	#FF00FF00		PowderBlue	#FFB0E0E6
	Bisque	#FFF0E4C4		Firebrick	#FFB22222		LimeGreen	#FF32CD32		Purple	#FF800080
	Black	#FF000000		FloralWhite	#FFFFFFAF0		Linen	#FFFAF0E6		Red	#FFFF0000
	BlanchedAlmond	#FFF0EBCD		ForestGreen	#FF228B22		Magenta	#FFFF00FF		RosyBrown	#FFBC8F8F
	Blue	#FF0000FF		Fuchsia	#FFFF00FF		Maroon	#FF800000		RoyalBlue	#FF4169E1
	BlueViolet	#FF8A2BE2		Gainsboro	#FFDCDCDC		MediumAquamarine	#FF66CDAA		SaddleBrown	#FF8B4513
	Brown	#FFA52A2A		GhostWhite	#FFF8F8FF		MediumBlue	#FF0000CD		Salmon	#FFFA8072
	BurlyWood	#FFDEB887		Gold	#FFFD7000		MediumOrchid	#FFBA55D3		SandyBrown	#FFFA4A60
	CadetBlue	#FF5F9EA0		Goldenrod	#FFDAA520		MediumPurple	#FF9370DB		SeaGreen	#FF2E8B57
	Chartreuse	#FF7FFF00		Gray	#FF808080		MediumSeaGreen	#FF3CB371		SeaShell	#FFFFFFFEE
	Chocolate	#FFD2691E		Green	#FF008000		MediumSlateBlue	#FF7B68EE		Sienna	#FFA0522D
	Coral	#FFF07F50		GreenYellow	#FFADFF2F		MediumSpringGreen	#FF00FA9A		Silver	#FFC0C0C0
	CornflowerBlue	#FF6495ED		Honeydew	#FFF0FFF0		MediumTurquoise	#FF48D1CC		SkyBlue	#FF87CEEB
	Cornsilk	#FFFFFF8DC		HotPink	#FFFF69B4		MediumVioletRed	#FFC71585		SlateBlue	#FF6A5ACD
	Crimson	#FFDC143C		IndianRed	#FFCD5C5C		MidnightBlue	#FF191970		SlateGray	#FF708090
	Cyan	#FF00FFFF		Indigo	#FF4B0082		MintCream	#FFF5FFFA		Snow	#FFFFFFFAFA
	DarkBlue	#FF00008B		Ivory	#FFFFFFF0		MistyRose	#FFFE4E1		SpringGreen	#FF00FF7F
	DarkCyan	#FF008B8B		Khaki	#FFF0E68C		Moccasin	#FFFE4B5		SteelBlue	#FF4682B4
	DarkGoldenrod	#FFB8860B		Lavender	#FFE6E6FA		NavajoWhite	#FFFDEAD		Tan	#FFD2B48C
	DarkGray	#FFA9A9A9		LavenderBlush	#FFFFF0F5		Navy	#FF000080		Teal	#FF008080
	DarkGreen	#FF006400		LawnGreen	#FF7CFC00		OldLace	#FFFD5E6		Thistle	#FFD8BFD8
	DarkKhaki	#FFBDB76B		LemonChiffon	#FFFFFACD		Olive	#FF808000		Tomato	#FFF6347
	DarkMagenta	#FF8B008B		LightBlue	#FFADD8E6		OliveDrab	#FF6B8E23		Transparent	#00FFFFFF
	DarkOliveGreen	#FF556B2F		LightCoral	#FFF08080		Orange	#FFFA500		Turquoise	#FF40E0D0
	DarkOrange	#FF8B0000		LightCyan	#FFE0FFFF		OrangeRed	#FFFA4500		Violet	#FFEE82EE
	DarkOrchid	#FF9932CC		LightGoldenrodYellow	#FFFAFAD2		Orchid	#FFDA70D6		Wheat	#FFF5DEB3
	DarkRed	#FF8B0000		LightGray	#FFD3D3D3		PaleGoldenrod	#FFEE8AA		White	#FFFFFFF
	DarkSalmon	#FFE9967A		LightGreen	#FF90EE90		PaleGreen	#FF98FB98		WhiteSmoke	#FFF5F5F5
	DarkSeaGreen	#FF8FBC8F		LightPink	#FFF0F0F0		PaleTurquoise	#FFAFEEEE		Yellow	#FFFFF000
	DarkSlateBlue	#FF483D8B		LightSalmon	#FFFA0A7A		PaleVioletRed	#FFDB7093		YellowGreen	#FF9ACD32
	DarkSlateGray	#FF2F4F4F									