Werken met data

Zie Scherp Scherper - Hoofdstuk 4

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- data omzetten via **casting**, **parsing** en de `Convert`-bibliotheek
- het verschil tussen **narrowing** en **widening** uitleggen
- gebruikersinvoer correct **inlezen en omzetten** naar een getal
- rekenen met `System.Math` en bewust **afronden** (let op *bankers rounding*)
- **willekeurige getallen** genereren met `Random`
- **debuggen** met breakpoints en je code stap voor stap doorlopen

Appelen en peren

Een waarde van het ene type zomaar aan een ander type toekennen mag niet:

```
1 int leeftijd = 4.3;  // FOUT: een double past niet zomaar in een int
```

Je kan geen appelen in peren veranderen zonder magie. In C# heb je drie manieren:

- **casting**: de klassieke manier (ook in C, C++, Java)
- de `Convert`-bibliotheek van .NET
- **parsing**: enkel om een `string` om te zetten

Casting

Plaats het doeltype tussen haakjes voor de waarde:

```
1  double kommagetal = 13.8;
2  int afgekapt = (int)kommagetal;  // 13
```

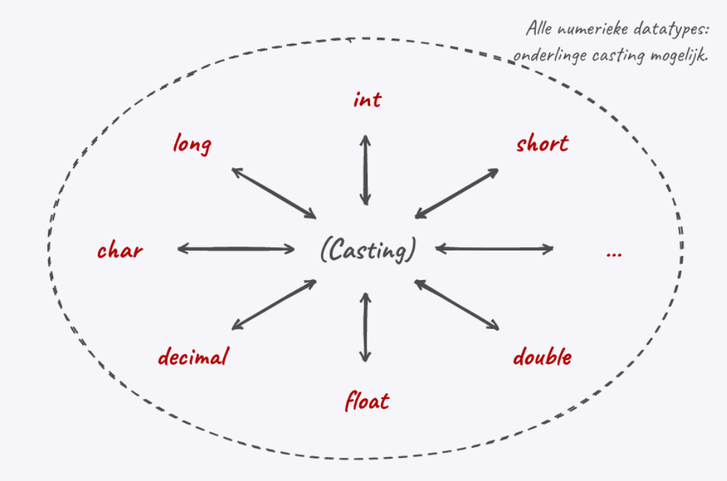
Belangrijk

Een variabele verandert **nooit** van type. Je zet enkel de **waarde** om en gebruikt die elders. `kommagetal` blijft een `double`.

Narrowing

Narrowing = een type omzetten met **verlies aan data** (bv. `double` naar `int`).

- Dit *vereist* een expliciete cast
- Je zegt zo tegen de compiler: “ik weet dat er data verloren gaat, en ik draag de verantwoordelijkheid”



Casting tussen numerieke types.

De klassieke valkuil

`tempGisteren` en `tempVandaag` zijn `int` (20 en 25). Wat geeft dit?

```
1  double gemiddeld = (tempGisteren + tempVandaag) / 2;
```

`22.0`, niet `22.5`. De expressie bevat enkel `int`'s, dus de deling is een gehele deling. Pas daarna gaat het resultaat naar de `double`.

```
1  double gemiddeld = ((double)tempGisteren + tempVandaag) / 2;  // 22.5
```

Let op de volgorde

De plaats van je cast maakt een verschil:

```
1  (double)(tempGisteren + tempVandaag) / 2;   // 22.5
2  (double)((tempGisteren + tempVandaag) / 2); // 22
```

In de tweede lijn dwingen de haakjes de **gehele deling eerst** af. Pas daarna cast je naar `double`, en dan is het kwaad al geschied.

Widening

Een *klein* type in een *groter* type past zonder dataverlies, dus **geen cast nodig**:

```
1  int hoofdMeting = 20;
2  double secundaire = hoofdMeting;  // 20.0, automatisch
```

> **Tip**
>
> Casten mag hier wel (`(double)hoofdMeting`), maar het hoeft niet. *Better safe than sorry* als je twijfelt.

Parsing en .ToString()

Parsing zet een `string` om naar een ander type:

```
1  int numVal = int.Parse("-105");
```

De omgekeerde weg, naar tekst, doe je met **`.ToString()`** (dat elk type heeft):

```
1  int getal = 5;
2  string tekst = getal.ToString();
```

> **Tip**
>
> Niet zeker dat de gebruiker een geldig getal invoert? Dan is `int.TryParse()` je vriend (zie appendix).

Invoer van de gebruiker

> `Console.ReadLine()` geeft **altijd** een `string` terug.

Wil je een getal, dan lees je eerst als tekst en **parse** je daarna:

```
1  Console.WriteLine("Geef je gewicht:");
2  double gewicht = double.Parse(Console.ReadLine());
```

> **Opmerking**
>
> De komende hoofdstukken mag je ervan uitgaan dat de gebruiker **foutloze** invoer geeft.

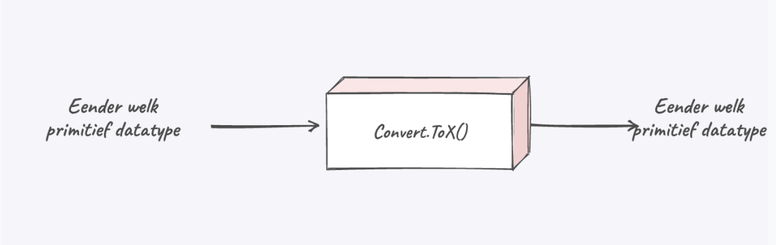
Komma of punt?

- **In je code** schrijf je een kommagetal-literal altijd met een **punt**: `9.81` (onafhankelijk van je taalinstellingen)
- **Als invoer** hangt het van de landinstellingen af: een Belgisch systeem verwacht `9,81`, een Engels `9.81`

De Convert-bibliotheek

```
1 int g = Convert.ToInt32(3.2);
2 double d = Convert.ToDouble(5);
3 bool b = Convert.ToBoolean(1);
4 int l = Convert.ToInt32("19");
```

- Naar `int` is `.ToInt32()`, naar `short` is `.ToInt16()`



Convert is een zwarte doos.

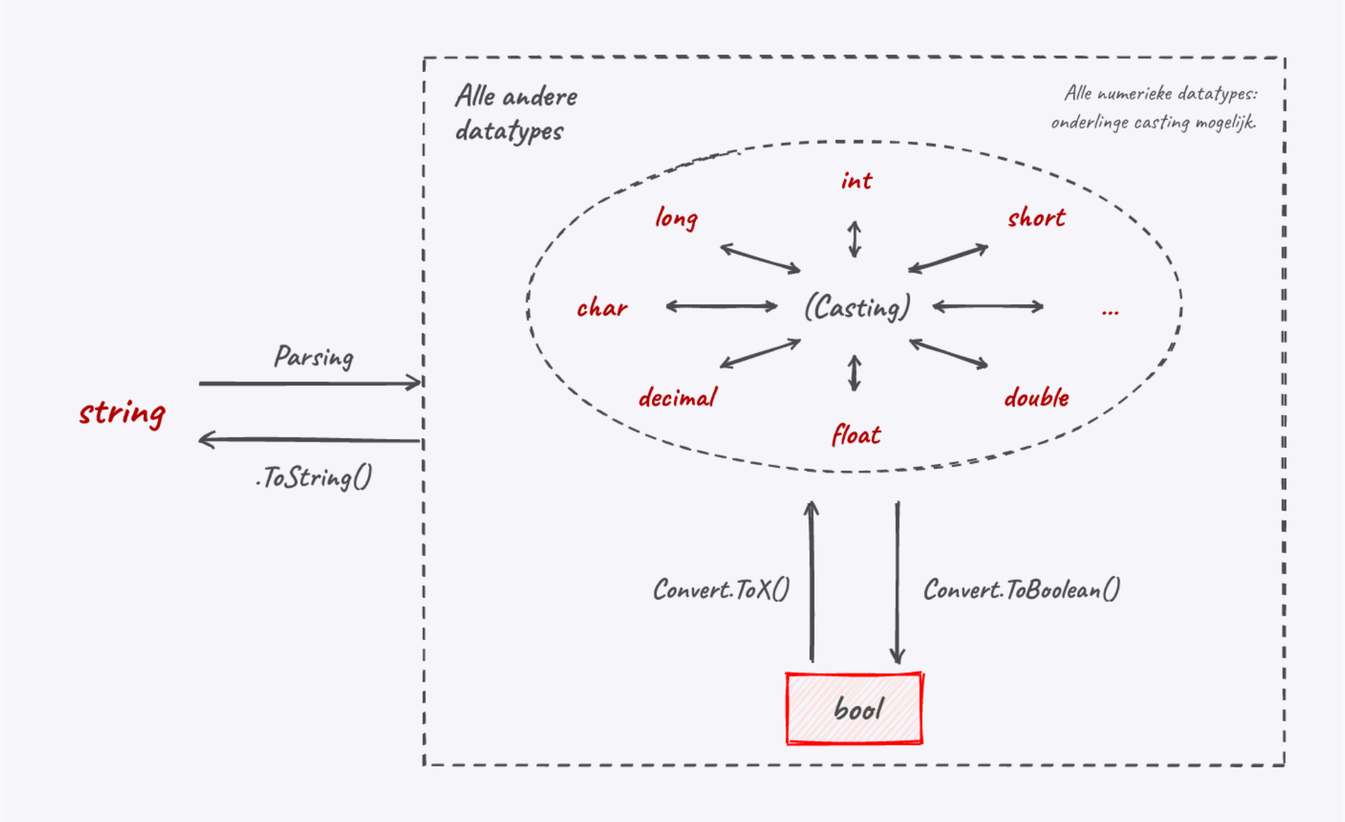
> ⚠ **Waarschuwing**
>
> Een **zwarte doos**: je ziet niet of er gecast of geparset wordt. Gebruik `Convert.To` pas als je het verschil goed snapt.

Convert en bool

> **Tip**
>
> **`Convert.ToBoolean`** geeft voor **elk** getal `True`, behalve voor `0` (of `0.0`): dan `False`.

Een `bool` omzetten naar een `int` (of omgekeerd) lukt **enkel** via `Convert`, niet via casting.

Casting, conversie of parsing?



Alle omzettingen samengevat.

Casting en parsing zijn compact en werken ook in andere talen. Beheers ze dus, ook al is `Convert` verleidelijk eenvoudig.

Rekenen met System.Math

De `Math`-bibliotheek bevat handige wiskunde:

```
double macht = Math.Pow(getal, 3);   // grondtal, exponent
double wortel = Math.Sqrt(144);      // 12
double omtrek = Math.PI * 2 * straal;
```

Verder o.a. `Sin`, `Cos`, `Tan`, `Ceiling`, `Floor`, `Abs`.

> **Tip**
>
> Typ `Math.` en IntelliSense toont alles. Cursor op een methode + **`F1`** opent de documentatie.

Het bereik in code

Elk datatype kent z’n eigen grenzen:

```
1  Console.WriteLine($"int gaat van {int.MinValue} tot {int.MaxValue}");
2  Console.WriteLine($"{double.MaxValue}");
```

Bij kommagetallen bestaat zelfs `double.PositiveInfinity` en `double.NegativeInfinity`.

Over afronden

We hebben al drie manieren gezien om af te ronden: **casting**, `Math.Round` en `Convert.ToX`. Ze gedragen zich niet hetzelfde.

- **Casting** kapt af richting nul (*truncatie*)

> ⚠ **Waarschuwing**
>
> Let op met negatieve getallen: `(int)-2.7` geeft **-2** (richting nul), maar `Math.Floor(-2.7)` geeft **-3** (richting min oneindig).

Bankers rounding

```
1  Console.WriteLine($"{Math.Round(4.5)} en {Math.Round(5.5)}");
```

Resultaat: 4 en 6?!

`Math.Round` (en `Convert.ToInt32`) ronden standaard af naar het **dichtstbijzijnde even getal**. Dat heet *bankers rounding*: handig voor bankiers, verwarrend voor ons.

Afronden zoals je verwacht

Geef een **`MidpointRounding`** mee:

```
1  Math.Round(4.5, MidpointRounding.AwayFromZero); // 5
2  Math.Round(5.5, MidpointRounding.AwayFromZero); // 6
```

- `ToEven` (standaard): bankers rounding
- `AwayFromZero`: het meest voorspelbare resultaat (onze voorkeur)

Willekeurige getallen

Maak **eenmalig** een generator en vraag dan getallen op:

```
1  Random generator = new Random();
2  int getal1 = generator.Next();
3  int getal2 = generator.Next();
```

> **Tip**
>
> De `new Random()`-syntax pluizen we vanaf hoofdstuk 9 uit. Lig er nu nog niet van wakker.

Een bereik kiezen

```
1  int a = generator.Next(0, 11);   // 0 tot en met 10
2  int b = generator.Next(55, 100); // 55 tot en met 99
```

Belangrijk

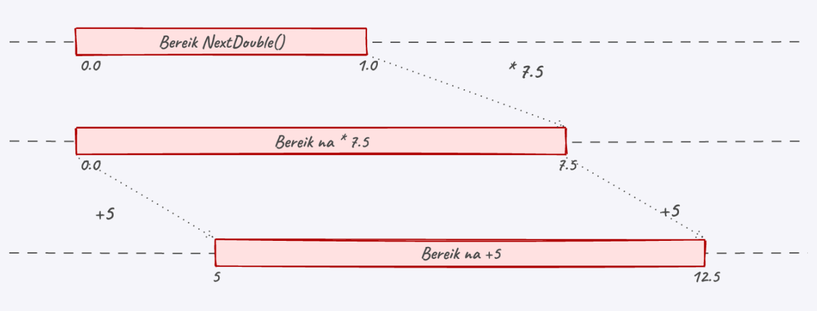
De ondergrens telt mee, de bovengrens niet. Wil je een getal *tot en met* X, schrijf dan **X + 1** als tweede parameter.

Kommagetallen met NextDouble

`NextDouble()` geeft een getal tussen `0.0` en `1.0`. Schaal en verschuif naar je bereik:

```
1  // tussen 5.0 en 12.5
2  double r = 5.0 + (generator.NextDouble() * 7.5);
```

Bereik = `12.5 - 5.0 = 7.5`, daarna `+5` om te verschuiven.



Bereik van Random verschuiven en schalen.

Steeds dezelfde getallen?

Twee generators die je vlak na elkaar aanmaakt, gebruiken bijna dezelfde **seed** (de tijd) en geven dus **dezelfde** reeks.

```
1  Random a = new Random();
2  Random b = new Random(); // slecht idee!
```

Belangrijk

Maak daarom maar **één** generator aan. Wil je voor het testen telkens dezelfde reeks? Geef dan een vaste seed mee: `new Random(666)`.

Debuggen: logica vs syntax

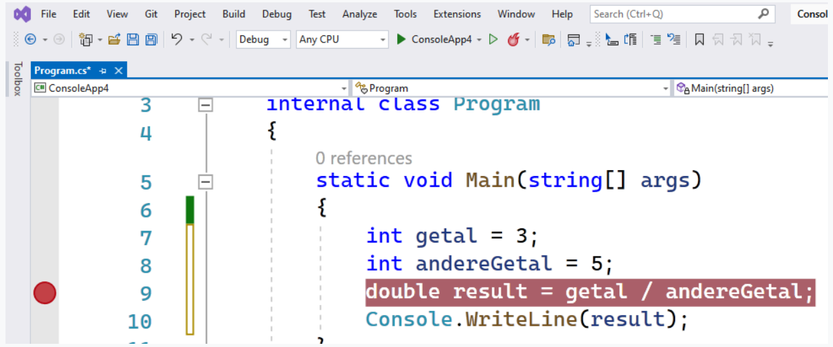
Code die compileert, is enkel **grammaticaal** correct. De betekenis kan nog altijd fout zijn:

```
1 Neem natrium
2 Neem water
3 Voeg beide samen
```

Perfecte “code”, maar een **logische fout** (en een stevige explosie). Dat is een **bug**.

Breakpoints

- Klik links van een lijn voor een **breakpoint** (rode bol)
- Je programma **pauzeert** daar
- In **Locals**/**Autos** zie je de waarde van elke variabele
- Je kan ook over een variabele **hoveren**



Een breakpoint.

Door je code steppen

Eens gepauzeerd, gebruik je de debugknoppen:

- **Continue**: verder tot het volgende breakpoint of het einde
- **Step over** (het gebogen pijltje): één lijn verder, dan weer pauzeren
- **Stop** (rode knop): stoppen om je code aan te passen

Debuggen is een essentiële skill

Een programmeur die niet kan debuggen is als een vis die niet kan zwemmen.

Voorspel telkens eerst wat er moet gebeuren: welke waarden, welke output? Klopt de werkelijkheid niet met je voorspelling, dan heb je waarschijnlijk een bug beet.

> ⚠ **Waarschuwing**
>
> Zomaar door je code steppen en hopen dat de bug magisch verschijnt, werkt niet. Wees kritisch.

Programmeren met A.I.

Tijd om dat krachtige beest in een omheind veldje te zetten, maar mét regels:

> **Belangrijk**
>
> **De verboden prompt**: *"Schrijf de C#-oplossing voor deze opgave."* Dat is als de Tour de France-winnaar de Mont Ventoux voor je laten opfietsen: je geraakt boven, maar leert niets.

Beter zijn prompts die je **doen denken**:

- **Zoek-de-fout**: laat AI code met ingebouwde fouten maken en zoek ze zelf
- **Vergelijk**: laat AI jouw oplossing met de modeloplossing vergelijken

> **Tip**
>
> AI wordt pas krachtig in **dialoog**. Anders is het een searchengine *on steroids*.

Conclusie

- Omzetten kan via **casting**, **parsing** en `Convert`: ken het verschil
- **Narrowing** (met dataverlies) vereist een cast, **widening** gaat vanzelf
- `ReadLine` geeft altijd een `string`: parse naar het juiste type
- `Math` helpt bij rekenen; let bij afronden op **bankers rounding**
- Eén `Random`-generator volstaat; **debuggen** doe je door te voorspellen

> **Tip**
>
> Volgende halte: **beslissingen** nemen met `if`, vergelijkingen en `switch`.

Meer weten

Kennisclips

- Casting, conversie en parsing
- Input verwerken en omzetten
- Math-library en berekeningen
- Afronden
- Random
- Debuggen

Oefeningen H4 · Quizlet flashcards

Zie verder: andere talen

Dezelfde concepten uit dit hoofdstuk, maar dan in andere talen. Handig om later code in Python of JavaScript te leren lezen.

Zie verder: type-coercion

C# is streng. **JavaScript** zet types net automatisch om (*coercion*):

```
1 console.log("5" + 3);  // "53"  (de 3 wordt een string)
2 console.log("5" * 3);  // 15    (de "5" wordt een getal)
```

Bij `+` wint de string, bij `*` het getal. In C# geeft dit een compileerfout.

Zie verder: input parsen

Python doet dit korter: de naam van het type is meteen de omzetfunctie:

```
1  gewicht = float(input("Geef je gewicht: "))
```

C# heeft een aparte `Parse` per type (`int.Parse`, `double.Parse`).

Zie verder: een module importeren

In **Python** zit wiskunde in een aparte *module* die je eerst moet importeren:

```python
import math

wortel = math.sqrt(144)   # 12.0
```

`math.` ervoor is de modulenaam. In C# hoort `Math` standaard bij de taal: niets importeren.

Samenvattende poster

Een handige samenvattende poster (Opgelet: gemaakt m.b.v. ChatGPT Image Gen 2).