



// 01

Casting

Gaat er data verloren (**narrowing**), dan **cast** je met het type tussen haakjes. Zonder verlies (**widening**) gaat het vanzelf.

```
double meting = 20.9;
int afgekapt = (int)meting; // narrowing: 20
double kopie = afgekapt; // widening: 20.0
```

`meting` blijft een `double`, enkel een kopie wordt omgezet. In een berekening cast je een operand:

```
int som = 20 + 25;
Console.WriteLine(som / 2);
Console.WriteLine((double)som / 2);
```

UITVOER
22
22,5

// LET OP

Een cast rondt niet af: `(int)2.9` is `2`. En `int / int` geeft een `int`: `5 / 2` is `2`. Wil je decimalen, maak dan één kant een `double`.

// 02

Invoer omzetten met Parse

`Console.ReadLine()` geeft **altijd** een `string`, ook als de gebruiker een getal typt. Zet die om met `int.Parse`, `double.Parse` enz.:

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

Zonder `Parse` plakt `+` je invoer aan elkaar: `"3" + "4"` is `"34"`. Op een Belgische pc typt de gebruiker `9,81`; in je code schrijf je altijd `9.81`.

// LET OP

`int.Parse("abc")` crasht je programma; voorlopig ga je uit van foutloze invoer. Wie op een Belgische pc `9.81` typt, krijgt zonder foutmelding `981`.

// 03

Convert en ToString

`Convert` zet bijna alles om, maar is een zwarte doos: je ziet niet of er gecast of geparsed wordt. `ToString()` maakt van elk type een `string`.

```
int leeftijd = Convert.ToInt32("19");
bool isWaar = Convert.ToBoolean(1); // true
string tekst = leeftijd.ToString(); // "19"
```

`ToBoolean` geeft enkel bij `0` een `false`. Van `bool` naar `int` kan enkel met `Convert.ToInt32`, niet met een cast.

// 04

Random

Maak **één keer** een generator en vraag er al je getallen aan. Met een **seed** krijg je bij elke start dezelfde reeks.

```
Random rng = new Random();
int a = rng.Next(0, 11); // 0 t.e.m. 10
double d = rng.NextDouble() * 10; // 0.0 tot 10.0
Random vast = new Random(666); // seed 666
```

// LET OP

De bovengrens telt niet mee: `Next(0, 10)` geeft nooit `10`.

// 05

De Math-bibliotheek

METHODE	DOET	GEEFT
<code>Math.Pow(2, 10)</code>	macht	<code>1024</code>
<code>Math.Sqrt(16)</code>	vierkantswortel	<code>4</code>
<code>Math.Abs(-42)</code>	absolute waarde	<code>42</code>
<code>Math.Max(3, 7)</code>	grootste (Min: kleinste)	<code>7</code>
<code>Math.Clamp(130, 0, 100)</code>	begrenzen	<code>100</code>
<code>Math.PI</code>	het getal pi	<code>3,14159...</code>

// LET OP

Bijna alles uit `Math` geeft een `double`, ook `Math.Pow(2, 10)`: cast met `(int)`. `Sin`, `Cos` en `Tan` rekenen in radialen:

```
hoek * Math.PI / 180.
```

// 06

Als rekenen misloopt

```
double nul = 0;
int groot = int.MaxValue;
Console.WriteLine(10 / nul);
Console.WriteLine(Math.Sqrt(-1));
Console.WriteLine(groot + 1);
```

UITVOER

∞
NaN
-2147483648

Een `int` delen door `0` crasht wel. Na het grootste `int` volgt het kleinste (**overflow**): kies dan `long`. `NaN` test je met `double.IsNaN(x)`.

// LET OP

Oneindig, `NaN` en overflow crashen niet: je rekent stil verder.

// 07

Afkappen en afronden

	2.7	-2.7	4.5	5.5
<code>(int)</code> of <code>Math.Truncate</code>	2	-2	4	5
<code>Math.Floor</code>	2	-3	4	5
<code>Math.Ceiling</code>	3	-2	5	6
<code>Math.Round</code> , <code>Convert.ToInt32</code>	3	-3	4	6
<code>Math.Round</code> + <code>AwayFromZero</code>	3	-3	5	6

Op de helft rondt `Math.Round` af naar het **even** getal (*bankers rounding*). Zoals op school:

```
double a = Math.Round(4.5,
    MidpointRounding.AwayFromZero); // 5
double b = Math.Round(4.12343, 2); // 4.12
```

Enkel mooi tonen? `($"{x:F2}"` laat de variabele ongemoeid.

// LET OP

Een `double` bewaart niet elk kommagetal exact: `(int)(4.35 * 100)` geeft `434`. Voor geld neem je `decimal: 4.35M`.

// 08

Debuggen

- **Breakpoint** (F9): de uitvoer pauzeert vóór die lijn
- **Locals**: de waarde van elke variabele op dat moment
- **Step over** (F10): één lijn verder
- **Continue**: door tot het volgende breakpoint
- **Voorspel** eerst: kijkt het af, dan is er wellicht een bug
- Voer je code geregeld uit, niet pas als alles geschreven is