

C# Essentials - Variabelen

Inhoudsopgave

1. Introductie	3
1.1. Wat is een variabele?	3
1.2. Waarom gebruiken we variabelen?	3
1.3. Constante	3
1.4. Naming Conventions	4
1.4.1. Regels	4
1.4.2. Oefening: juist of fout	4
2. Datatypes	6
2.1. Wat is een datatype?	6
2.2. Waarom zijn datatypes belangrijk?	6
2.3. Veelgebruikte datatypes	6
2.4. Declaratie vs. initialisatie	7
2.4.1. Declaratie	7
2.4.2. Initialisatie	7
2.4.3. Declaratie én initialisatie	7
2.4.4. Vergelijking met een doos	7
2.5. Value Types & Reference Types	8
3. Numerieke variabelen	9
3.1. Gehele getallen	9
3.2. Decimale getallen	9
3.3. Signed vs Unsigned	9
3.4. Oefening: juist of fout	10
4. Alfanumerieke variabelen	12
4.1. Tekst	12
4.1.1. Escape sequences	12
4.1.2. Verbatim strings	12
4.1.3. String Interpolation <code>@@@</code>	12
4.2. Teken	13
4.3. Oefening: juist of fout	13
5. Andere datatypes	15
5.1. Boolean (<code>bool</code>)	15
5.2. Datum en tijd (<code>DateTime</code>)	15
5.3. Tijdsduur (<code>TimeSpan</code>)	15
5.4. DateOnly en TimeOnly	16
5.5. Samenvatting	16

5.6. Oefening: juist of fout	17
6. Operatoren	19
6.1. Operatoren	19
6.1.1. Rekenkundige operatoren (Arithmetic Operators)	19
6.1.2. Vergelijgingsoperatoren (Comparison Operators)	19
6.1.3. Logische operatoren (Logical Operators)	20
6.1.4. Toekenningsoperatoren (Assignment Operators)	21
6.2. Volgorde van uitvoering (Operator Precedence)	21
6.2.1. Rekenkundige operatoren	21
6.2.2. Vergelijking en logica	21
6.3. Oefeningen	22
7. Converteren	25
7.1. Conversies	25
7.1.1. Implicite conversie	25
7.1.2. Expliciete conversie (casten)	25
7.2. ToString()	25
7.3. Parsing	26
7.3.1. De Parse-methode	26
7.3.2. De TryParse-methode ↴	26
8. Formattering	28
8.1. Waarom string formatting?	28
8.2. string.Format()	28
8.3. String interpolatie (\$-strings)	28
8.4. Getallen en datums formatteren	29
8.4.1. Getallen	29
8.4.2. Datums	29
8.5. Uitlijnen en opvullen	30

1. Introductie

1.1. Wat is een variabele?

Een variabele is een naam die je gebruikt om te verwijzen naar een geheugenlocatie die je gebruikt om een stukje informatie op te slaan.

Je kan het zien als een doos met een label erop. In elke doos kan informatie bewaard worden en omdat elke doos een label heeft, kan deze informatie eenvoudig worden teruggevonden wanneer dat nodig is.



In C# ziet dat er bijvoorbeeld zo uit:

```
string name = "Wim";
int age = 40;
double height = 1.85;
```

1.2. Waarom gebruiken we variabelen?

Als je een programma maakt, wil je vaak gegevens bijhouden die kunnen veranderen of die je meermaals nodig hebt. Een variabele laat je toe om die gegevens een naam te geven en ze later opnieuw te gebruiken.

Voorbeeld:

```
Console.WriteLine("Wat is uw naam?")
string name = Console.ReadLine();
Console.WriteLine($"Welkom in deze cursus, {name}");
```

1.3. Constante

Sommige variabelen krijgen eenmaal een waarde toegewezen en blijven daarna ongewijzigd. Dit noem je een **constante**. In C# ziet dat er zo uit:

```
const decimal MinimumAmountForFreeDelivery = 20M;
```

1.4. Naming Conventions

Wanneer je variabelen gebruikt in je programma, is het belangrijk dat je dit op een duidelijke en consequente manier doet. Zo kan je code ook door anderen (of door jezelf op een later moment) makkelijker begrepen worden. Deze afspraken over naamgeving noemen we **naming conventions**.

In C# gelden de volgende basisregels voor het benoemen van variabelen:

- Gebruik **camelCasing** voor variabelen (voorbeeld: firstName, lastName)
- Gebruik **PascalCasing** voor constanten (voorbeeld: MinOrderAmount)
- Gebruik **aparte lijnen** voor elke declaratie
- Gebruik **betekenisvolle namen!**
- Gebruik een **Engelse** benaming

[Zie PXL Coding Conventions](#)

1.4.1. Regels

Naast de richtlijnen zijn er ook een aantal regels die je moet volgen om geldige C#-code te schrijven:

- Een variabele **moet** een naam hebben
- Die naam mag geen spaties of speciale tekens bevatten
- De naam mag niet beginnen met een cijfer
- De variabele moet eerst een waarde krijgen voor je ze kan gebruiken

1.4.2. Oefening: juist of fout

Welke van de volgende namen zijn in C# toegestaan? Welke hiervan volgen ook de juiste naming conventions?

```
volume  
AREA  
Length  
3sides  
side1  
heit  
maxsalary  
min salary
```

Oplossing

```
volume      // ✅ juist: toegestaan én correcte stijl  
AREA        // ✅ fout: toegestaan maar 'area' is beter  
Length      // ✅ fout: toegestaan maar 'length' is beter  
3sides      // ✅ fout: niet toegestaan, begint met een cijfer  
side1       // ✅ juist: toegestaan én correcte stijl  
heit        // ✅ juist: toegestaan én correcte stijl, de verkeerde spelling
```

```
doet er niet toe  
maxsalary      // ✘ fout: toegestaan maar 'maxSalary' is beter  
min salary     // ✘ fout: niet toegestaan, bevat een spatie
```

2. Datatypes

2.1. Wat is een datatype?

Wanneer je een variabele aanmaakt in C#, moet je zeggen wat voor soort gegevens je erin wil bewaren. Dit noem je het datatype van de variabele.

In onderstaand voorbeeld heeft elk label een andere kleur. Elke kleur duidt een ander type aan.



```
string name = "Wim";      //een tekst  
int age = 40;            //een geheel getal  
double height = 1.85;    //een decimaal getal
```

2.2. Waarom zijn datatypes belangrijk?

C# is een sterk getypeerde taal (**Strongly Typed**). Dat wil zeggen dat je altijd duidelijk moet zijn over welke soort gegevens je gebruikt, zodat C# weet:

- hoeveel geheugen het moet voorzien
- welke bewerkingen toegestaan zijn
- of jouw code logisch en veilig is

2.3. Veelgebruikte datatypes

Type	Omschrijving	Opslagruimte
int	Een geheel getal, zoals -5, 0, 42	32 bits
double	Een kommagetal, zoals 3.14	64 bits
bool	Een waar/onwaar-waarde: true of false	8 bits (meestal)
string	Een stuk tekst, zoals "Hallo"	varieert (Unicode per karakter)
char	Een enkel teken, zoals 'A' of '5'	16 bits (Unicode)

Type	Omschrijving	Opslagruimte
DateTime	Een datum met tijd, zoals 20/05/2025 20:00	64 bits

2.4. Declaratie vs. initialisatie

Wanneer je met variabelen werkt, zijn er twee belangrijke stappen:

- **Declaratie:** je zegt dat er een variabele bestaat zonder deze een waarde te geven
- **Initialisatie:** je geeft die variabele een *eerste* waarde

2.4.1. Declaratie

Bij een declaratie vertel je aan het programma welk type variabele je gaat gebruiken (=datatype) en hoe ze heet (=naam van de variabele).

```
string olod;      //Datatype=string, Naam=olod
bool isStudent;  //Datatype=bool, Naam=isStudent
```

Op dit moment **heeft de variabele nog geen waarde.**

2.4.2. Initialisatie

Bij initialisatie geef je de variabele ook een beginwaarde:

```
olod = "C# Essentials";
isStudent = true;
```

Nu zit er effectief een waarde in die variabele. Je kan de waarde vanaf nu gebruiken in je code.

2.4.3. Declaratie én initialisatie

Vaak doe je beide stappen in één lijn code:

```
string olod = "C# Essentials";
bool isStudent = true;
```

2.4.4. Vergelijking met een doos

Stel je een variabele voor als een doos:

- Declaratie: je maakt een doos met een label, maar ze is nog leeg.
- Initialisatie: je steekt iets in die doos.



Als je probeert de inhoud van een lege doos te gebruiken (een gedeclareerde maar niet geïnitialiseerde variabele), krijg je een foutmelding in C#.

2.5. Value Types & Reference Types

In C# bestaan er twee grote soorten types: value types en reference types. Het verschil zit in hoe de gegevens worden opgeslagen in het geheugen.

- Een **value type** bewaart de echte waarde zelf.

Typische value types: int, double, bool, char, decimal, DateTime

- Een **reference type** bewaart een verwijzing naar de echte waarde in het geheugen.

Typische reference types: string, class-types, array

Later zal je leren dat dit type invloed heeft wanneer je variabelen kopiëert of "doorgeeft".

3. Numerieke variabelen

3.1. Gehele getallen

Gehele getallen zijn getallen **zonder** komma:

Type	Omschrijving	Opslagruimte	Waardenbereik
int	Standaardtype voor getallen	32 bits	van -2.147.483.648 tot 2.147.483.647
long	Grottere gehele getallen	64 bits	van -9 triljoen tot +9 triljoen
short	Kleinere getallen	16 bits	van -32.768 tot 32.767
byte	Hele kleine getallen	8 bits	van 0 tot 255 (zie unsigned)

Gebruik een `l` of `L` achter de waarde voor long:



```
long distanceFromEarthToMarsInMeters = 225_000_000_000l; // L mag ook
```

* De underscores (`_`) zijn optioneel en dienen enkel voor de leesbaarheid

3.2. Decimale getallen

Als je wél een komma nodig hebt, gebruik je een **floating point**-type:

Type	Gebruik	Precisie	Opslagruimte
float	ruwe berekeningen, grafieken	laag	32 bits
double	standaard kommagetal	gemiddeld	64 bits
decimal	voor geld of nauwkeurige waarden	hoog	128 bits

Gebruik een `f` achter de waarde voor float, een `m` achter de waarde voor decimal:



```
float temperature = 36.6f; // 36.6F mag ook  
decimal price = 19.99m; // 19.99M mag ook
```

3.3. Signed vs Unsigned

- Een signed getal betekent dat je zowel negatieve als positieve waarden kan opslaan.

- Een unsigned getal betekent dat je alleen positieve waarden kan opslaan, maar het bereik is dan dubbel zo groot.

Voorbeeld:

Type	Bereik	Signed
int	-2.147.483.648 tot +2.147.483.647	Ja
uint	0 tot 4.294.967.295	Nee

In de praktijk gebruiken we bijna altijd signed types, zoals int en long, omdat we dan ook negatieve waarden kunnen opslaan. Maar als je zeker weet dat een waarde nooit negatief is (bijv. leeftijd, aantal studenten, voorraad), dan kan unsigned interessant zijn.

3.4. Oefening: juist of fout

Onderstaande regels proberen numerieke variabelen te declareren en initialiseren. Sommige zijn juist, andere bevatten een fout. Kan jij ze allemaal juist inschatten?

```
int age = 25;
long distance = 123456789012345;
byte number = 300;
float temperature = 36.6;
double pi = 3.14159;
decimal price = 99.95;
float ratio = 1.5f;
decimal vat = 0.21m;
short code = 35000;
uint items = -5;
```

Oplossing

```
int age = 25;                                // ✅ juist
// Geldige integer binnen bereik

long distance = 123456789012345;           // ❌ fout
// Zonder `L` suffix is dit een int literal → te groot → compileerfout

byte number = 300;                            // ❌ fout
// 300 is te groot voor byte (max = 255)

float temperature = 36.6;                   // ❌ fout
// Zonder `f` wordt 36.6 als double geïnterpreteerd → compileerfout

double pi = 3.14159;                          // ✅ juist
// Double is standaard voor kommagetalen
```

```
decimal price = 99.95;           // ✘ fout
// Zonder `m` is dit een double → moet `99.95m` zijn

float ratio = 1.5f;             // ✘ juist
// Correcte notatie voor float

decimal vat = 0.21m;            // ✘ juist
// Decimal voor geldbedragen → met `m` suffix

short code = 35000;             // ✘ fout
// 35000 is te groot voor `short` (max = 32.767)

uint items = -5;                // ✘ fout
// `uint` kan geen negatieve waarden bevatten
```

4. Alfanumerieke variabelen

4.1. Tekst

Een tekst (of tekenreeks) noemen we in C# een `string`. De waarde van een stringvariabele staat in C# altijd tussen **dubbele quotes**:

```
string old = "C# Essentials";
string department = "PXL Digital";
string translation = "age";
```

4.1.1. Escape sequences

Soms wil je in een string speciale tekens gebruiken, zoals een **nieuwe lijn** of een **aanhalingstekens**. Dat doe je met een **escape character**, meestal voorafgegaan door een backslash (\).

```
string message = "Hello\nWorld!";           // \n = nieuwe lijn
string quote = "He said: \"Hi!\"";          // \" = aanhalingstekens
string path = "C:\\\\Users\\\\Emma";           // \\\\" = backslash
```

Escape sequence	Betekenis
\n	Nieuwe lijn
\t	Tab
\"	Dubbel aanhalingstekens
\\\\"	Backslash

4.1.2. Verbatim strings

Als je geen escape characters wil gebruiken, kan je een **verbatim string** gebruiken met een @ ervoor. Op die manier is de waarde van de string letterlijk hetgeen wat tussen de dubbele quotes staat.

```
string path = @"C:\\\\Users\\\\Emma\\\\Documents";
```

Als je aanhalingstekens wil gebruiken in een verbatim string, moet je ze **verdubbelen**:



```
string quote = @"""He said: ""\"Hi!""";
```

4.1.3. String Interpolation

Met **string interpolation** kan je variabelen eenvoudig in een string verwerken. Je gebruikt een \$ vóór de string en accolades {} rond de variabelen:

```
string olod = "C# Essentials";
Console.WriteLine($"Het vak {olod} is echt superleuk!");
```

Dit is overzichtelijker dan:

```
Console.WriteLine("Het vak " + olod + " is echt superleuk!");
```

Je kan ook (eenvoudige) berekeningen gebruiken in interpolatie!

```
int a = 5;
int b = 3;
string result = $"De som is {a + b}";
```

4.2. Teken

Het datatype `char` gebruik je om één enkel teken op te slaan, zoals een letter, cijfer of symbool.

- Een `char` staat tussen **enkele quotes**: 'A'
- Je kan er **exact 1 teken** in bewaren, geen woord of zin

```
char letter = 'A';
char digit = '7';
char symbol = '#';
```

4.3. Oefening: juist of fout

Onderstaande regels proberen tekst en tekens op te slaan. Sommige zijn correct, andere bevatten fouten. Kan jij het verschil herkennen?

```
string name = "Alice";
char initial = 'A';
string empty = '';
char symbol = "#";
char digit = '7';
string text = "Hello, world!";
char emoji = '\u263a';
char invalid = 'AB';
string quote = "He said: \"Hi!\"";
string line = "First line\nSecond line";
```

Oplossing

```
string name = "Alice"; // ✅ juist
// Correcte string initialisatie

char initial = 'A'; // ✅ juist
```

```
// Enkele letter, tussen enkele aanhalingstekens

string empty = '';
// Lege tekst moet tussen dubbele aanhalingstekens staan: `""` 

char symbol = "#";
// Dubbele quotes horen bij string, `char` vereist enkele quotes

char digit = '7';
// Een enkel teken tussen enkele aanhalingstekens

string text = "Hello, world!";
// Geldige string

char emoji = '\u263a';
// Emoji is één Unicode-teken → mag in `char`

char invalid = 'AB';
// `char` mag slechts 1 teken bevatten

string quote = "He said: \"Hi!\"";
// Escape character `\"` correct gebruikt

string line = "First line\nSecond line";
// Nieuwe lijn via `\\n` is toegestaan
```

5. Andere datatypes

5.1. Boolean (`bool`)

Met een `bool` geef je een **ja/nee-antwoord** of **waar/onwaar-waarde** aan.

```
bool isLoggedIn = true;  
bool isAvailable = false;
```

Typische toepassingen:

- controleren of iets geactiveerd is
- testen of een conditie waar is

Je gebruikt `bool` ook in `if`-statements:

```
if (isLoggedIn)  
{  
    Console.WriteLine("Welcome!");  
}
```



De `if`-structuur leer je in de module controlestructuren.

5.2. Datum en tijd (`DateTime`)

Het `DateTime` type gebruik je om een specifieke datum of tijdstip op te slaan.

Voorbeelden:

```
DateTime today = DateTime.Today;  
DateTime now = DateTime.Now;  
DateTime birthday = new DateTime(2000, 5, 20);
```

Je kan een string omzetten naar een `DateTime`:

```
DateTime parsedDate = DateTime.Parse("2024-01-01");
```

Extra functies:

```
Console.WriteLine(today.DayOfWeek); // bijv. Monday  
Console.WriteLine(now.ToString()); // bijv. "donderdag 20 mei 2025"
```

5.3. Tijdsduur (`TimeSpan`)

Een `TimeSpan` drukt een verschil in tijd uit (duur tussen twee tijdstippen).

Voorbeelden:

```
TimeSpan duration = new TimeSpan(2, 30, 0); // 2 uur, 30 minuten  
Console.WriteLine(duration); // 02:30:00
```

Je kan ook het verschil tussen twee datums berekenen:

```
DateTime start = new DateTime(2024, 1, 1);  
DateTime end = DateTime.Now;  
  
TimeSpan difference = end - start;  
Console.WriteLine(difference.Days); // verschil in aantal dagen
```

5.4. DateOnly en TimeOnly

Sinds .NET 6 bestaan er twee nieuwe types die handig zijn wanneer je enkel een datum of enkel een tijd nodig hebt:

- `DateOnly` stelt enkel een kalenderdatum voor (jaar, maand, dag), zonder tijdstip.
- `TimeOnly` stelt enkel een tijdstip voor (uur, minuut, seconde), zonder datum.

Dit maakt je code duidelijker dan wanneer je hiervoor altijd `DateTime` zou gebruiken.

Voorbeelden:

```
DateOnly birthDate = new DateOnly(2000, 5, 12); // 12 mei 2000  
Console.WriteLine(birthDate); // 12/05/2000  
  
TimeOnly startTime = new TimeOnly(9, 30); // 9u30  
Console.WriteLine(startTime); // 09:30  
  
// Combineren  
DateOnly today = DateOnly.FromDateTime(DateTime.Now);  
TimeOnly now = TimeOnly.FromDateTime(DateTime.Now);  
Console.WriteLine($"Vandaag: {today}, tijdstip: {now}");
```



Gebruik `DateOnly` of `TimeOnly` telkens wanneer je geen volledige datum-tijd combinatie nodig hebt. Dit maakt je code eenvoudiger en verkleint de kans op fouten.

5.5. Samenvatting

Type	Wat sla je op?	Voorbeeld
<code>bool</code>	waar/onwaar	<code>bool isOpen = true;</code>

Type	Wat sla je op?	Voorbeeld
DateTime	een specifieke datum/tijd	DateTime now = DateTime.Now;
TimeSpan	een tijdsverschil of duur	TimeSpan d = end - start;

5.6. Oefening: juist of fout

```
bool isValid = true;  
DateTime birthday = "2023-04-15";  
TimeSpan duration = new TimeSpan(1, 30, 0);  
DateTime eventDate = DateTime.Parse("15/04/2023");  
bool isActive = "true";  
TimeSpan interval = new TimeSpan(2, 75, 0);  
DateTime now = DateTime.Now;  
bool flag = false;  
TimeSpan span = DateTime.Now - DateTime.Today;  
DateTime start = new DateTime(2023, 15, 1);
```

?

Oplossing

```
TimeSpan span = DateTime.Now - DateTime.Today;           // ✅ juist
// Verschil tussen twee DateTimes is een TimeSpan

DateTime start = new DateTime(2023, 15, 1);            // ❌ fout
// Ongeldige maand: 15 bestaat niet → runtime-fout
```

6. Operatoren

6.1. Operatoren

We verdelen de operatoren in vier grote groepen:

6.1.1. Rekenkundige operatoren (Arithmetic Operators)

Deze gebruik je voor wiskundige bewerkingen zoals optellen en delen.

Operator	Betekenis	Voorbeeld	Resultaat
+	Optellen	5 + 3	8
-	Aftrekken	10 - 4	6
*	Vermenigvuldigen	2 * 3	6
/	Delen	10 / 2	5
%	Modulo (rest)	10 % 3	1
++	Vermeerderen met 1	5++	6
--	Verminderen met 1	5--	4

```
int x = 5;
int y = 3;
int sum = x + y;           // sum = 8
x++;                      // x = 6
int product = x * 2;       // product = 6 * 2 = 12
int mod = product % 10;    // mod = 12 % 10 = 2
```

! Bij delen van gehele getallen (`int`) wordt het resultaat ook een geheel getal.

```
double result = 7 / 2; // resultaat = 3.0, niet 3.5
```

6.1.2. Vergelijgingsoperatoren (Comparison Operators)

Deze gebruik je om twee waarden te vergelijken. Het resultaat is altijd `true` of `false`.

Operator	Betekenis	Voorbeeld	Resultaat
<code>==</code>	Gelijk aan	5 == 5	<code>true</code>

Operator	Betekenis	Voorbeeld	Resultaat
<code>!=</code>	Niet gelijk aan	<code>5 != 3</code>	<code>true</code>
<code><</code>	Kleiner dan	<code>3 < 5</code>	<code>true</code>
<code>></code>	Groter dan	<code>6 > 2</code>	<code>true</code>
<code><=</code>	Kleiner of gelijk	<code>4 <= 4</code>	<code>true</code>
<code>>=</code>	Groter of gelijk	<code>7 >= 8</code>	<code>false</code>

```
bool isEqual = 10 == 10;      // true
bool isSmaller = 3 < 2;      // false
```

6.1.3. Logische operatoren (Logical Operators)

Deze gebruik je om meerdere voorwaarden te combineren.

Operator	Naam	Voorbeeld	Resultaat voor <code>int x = 15;</code>
<code>&&</code>	AND (<i>en</i>)	<code>x > 10 && x < 20</code>	<code>true</code>
<code> </code>	OR (<i>of</i>)	<code>x == 10 x == 20</code>	<code>false</code>
<code>!</code>	NOT (<i>niet</i>)	<code>!(x == 15)</code>	<code>false</code>

```
bool result = (5 > 3) && (2 < 4); // true
bool result = !(6 == 6);           // false
```

`&&` `!=` `||`

'en' is niet hetzelfde als 'of'

Zie jij wat er fout is met deze code?

```
string input = Console.ReadLine();
int number = int.Parse(input);

if (number < 10 && number > 20)
{
    Console.WriteLine("Nummer ligt niet tussen 10 en 20.");
}
```



Probeer de code in je hoofd uit te voeren en gebruik telkens een andere waarde voor

number (bv 8, 13 en 27)

Oplossing

Een getal kan nooit én kleiner dan 10 én groter dan 20 zijn. Deze voorwaarde zal dus **nooit** true als resultaat hebben.

6.1.4. Toekenningsoperatoren (Assignment Operators)

Deze gebruik je om een waarde toe te kennen aan een variabele. Er bestaan ook verkorte vormen.

Operator	Betekenis	Voorbeeld	Uitleg
=	Toekennen	x = 5	x wordt 5
+=	Optellen en toekennen	x += 2	x = x + 2
-=	Aftrekken en toekennen	x -= 3	x = x - 3
*=	Vermenigvuldigen	x *= 4	x = x * 4
/=	Delen en toekennen	x /= 2	x = x / 2

```
int x = 10;  
x += 5; // x is nu 15
```

6.2. Volgorde van uitvoering (Operator Precedence)

Wanneer een expressie meerdere operatoren bevat, dan bepaalt de **volgorde van uitvoering** welke bewerking eerst gebeurt. Dit is belangrijk om fouten te vermijden.

6.2.1. Rekenkundige operatoren

De volgorde bij wiskundige operatoren is gelijkaardig aan de regels uit de wiskunde:

1. Haakjes ()
2. Vermenigvuldigen en delen * / %
3. Optellen en aftrekken + -

```
int result = 5 + 3 * 2; // result = 11 (niet 16!)  
int correct = (5 + 3) * 2; // correct = 16
```

6.2.2. Vergelijking en logica

Ook bij vergelijkingen en logische operatoren is er een vaste volgorde:

1. `!` (niet)
2. `==`, `!=`, `<`, `>`, `<=`, `>=`
3. `&&` (en)
4. `||` (of)

```
int x = 5;
bool result = x == 5 || x > 10 && x == 10;
// resultaat = true (want && wordt eerst uitgevoerd)

bool result2 = !(5 > 3) || true;
// resultaat = false || true => true
```

Je kan altijd haakjes gebruiken om zelf de volgorde te bepalen. Dit maakt je code ook duidelijker leesbaar.



```
bool isOk = (score >= 50) && (passed == true);
```

6.3. Oefeningen

Voorspel de juiste waarde van `result`:

- Oefening 1

```
int x = 10;
int y = 7;
x = 8;
int result = x - y;
```

Oplossing

```
result : 1
```

- Oefening 2

```
int x = 3;
int y = 3;
y = y + 2;
int result = x * y;
```

Oplossing

```
result : 15
```

- Oefening 3

```
int x = 8;  
int y = 4;  
x /= 2;  
int result = x % y;
```

☒ *Oplossing*

```
result : 0
```

- Oefening 4

```
int x = 8;  
int y = 4;  
x -= 2;  
bool result = x > y;
```

☒ *Oplossing*

```
result : true
```

- Oefening 5

```
int x = 2;  
int y = 7;  
  
bool result = x >= y;
```

☒ *Oplossing*

```
result : false
```

- Oefening 6

```
int x = 3;  
int y = 6;  
x = 18 / x;  
bool result = x == y;
```

☒ *Oplossing*

```
result : true
```



x = 18 / x resulteert in x = 6.6 = 6 dus x is gelijk aan y.

- Oefening 7

```
int x = 9;  
int y = 3;
```

```
x++;  
bool result = x % y == 0;
```

Oplossing

```
result : false
```



x++ resulteert in $x = 10$. $10 \% 3 = 1$ en is dus niet gelijk aan 0

- Oefening 8

```
int x = 11;  
int y = 14;  
  
bool result = x >= 10 && y <= 15;
```

Oplossing

```
result : true
```



$x = 11$ (en dus groter of gelijk aan 10) én $y = 14$ (en dus kleiner of gelijk aan 15).

- Oefening 9

```
int x = 10;  
int y = 15;  
  
bool result = x == 10 || y == 20;
```

Oplossing

```
result : true
```

- Oefening 10

```
int x = 10;  
int y = 15;  
  
bool result = x == 10 && x == 14;
```

Oplossing

```
result : false
```



Een getal kan nooit gelijk zijn aan 10 én aan 14

7. Converteren

7.1. Conversies

Wanneer je werkt met variabelen in C#, komt het vaak voor dat je een waarde van het ene type naar een ander type moet omzetten. Dit noemen we **conversie**. Er zijn verschillende manieren om dit te doen:

7.1.1. Impliciete conversie

Een **impliciete conversie** gebeurt automatisch wanneer C# zeker weet dat er geen informatie verloren kan gaan. Bijvoorbeeld wanneer je een `int` omzet naar een `double`.

```
int wholeNumber = 42;
double result = wholeNumber; // impliciete conversie van int naar double =>
result = 42.00
```

In dit geval is er geen informatieverlies: elk geheel getal kan zonder probleem opgeslagen worden als een `double`.

Let op: niet alle types ondersteunen impliciete conversie. Zo kan je bijvoorbeeld geen `double` automatisch omzetten naar een `int`, omdat je dan precisie zou verliezen.

7.1.2. Expliciete conversie (casten)

Wanneer je een type **expliciet** wil omzetten, gebruik je een **cast**. Dit doe je door het gewenste type tussen haakjes te zetten.

```
double pi = 3.14;
int rounded = (int)pi; // expliciete conversie => rounded = 3
```

Hier verlies je het decimale gedeelte. Je zegt dus expliciet tegen C#: “**ik weet wat ik doe.**”

7.2. `ToString()`

Elk datatype in C# bevat de methode `ToString()`, die je kan gebruiken om een waarde **als tekst** voor te stellen.

```
int number = 42;
Console.WriteLine(number.ToString()); // "42"

DateTime now = DateTime.Now;
Console.WriteLine(now.ToString()); // "20/05/2025 14:36:12"
```



1. Sommige types (zoals `DateTime` en `double`) laten toe om extra parameters te geven aan `ToString()` om de opmaak aan te passen. Meer hierover leer je in het volgende hoofdstuk "Formatting".

2. Je hoeft niet altijd expliciet `ToString()` te typen. De `ToString()`-methode wordt automatisch aangeroepen:

- bij `Console.WriteLine(obj);`
- in string interpolatie: `$"Naam: {obj}"`
- bij `string.Format("Naam: {0}", obj);`

```
int number = 42;
Console.WriteLine(number); // "42"
```

7.3. Parsing



7.3.1. De Parse-methode

Een veelgebruikte manier om een `string` om te zetten naar een ander type is via de `Parse`-methode. Elk basisgegevenstype (zoals `int`, `double`, `bool`, ...) heeft zijn eigen `Parse`-methode.

```
string text = "123";
int number = int.Parse(text);
// of
double number = double.Parse(text);
```

Let op, deze functie geeft een error:

- Als de string geen geldig getal is (bijvoorbeeld `"abc"`)
- Als de string `null` (leeg) is

`Parse` is vooral handig wanneer je zeker weet dat de invoer correct is, bijvoorbeeld bij data uit een vertrouwde bron.

7.3.2. De TryParse-methode

De `TryParse`-methode is een **veilige** manier om een `string` om te zetten naar een ander type zonder dat je programma crasht bij foutieve input. In plaats van een fout (exception) te geven, zal `TryParse`

gewoon `false` teruggeven als de omzetting niet lukt.

Het grote voordeel? Je kan eerst controleren of de conversie gelukt is, voor je met het resultaat verder werkt.

```
string input = "456";
if (int.TryParse(input, out int result))
{
    Console.WriteLine("Geldig getal: " + result);
}
else
{
    Console.WriteLine("Ongeldige invoer.");
}
```

In dit voorbeeld:

- `input` is de string die je wil omzetten.
- `out int result` betekent dat de uitkomst van de conversie in de variabele `result` wordt gestopt, maar pas als de conversie lukt.
- De methode geeft `true` of `false` terug, afhankelijk van of het gelukt is.



Gebruik `TryParse` altijd bij invoer van de gebruiker (via `Console.ReadLine()` bijvoorbeeld), want gebruikers maken fouten. Je programma zal dan niet crashen.

[meme users] | *meme-users.webp*

Voorbeeld:

```
Console.Write("Geef een leeftijd in: ");
string input = Console.ReadLine();

if (int.TryParse(input, out int age))
{
    Console.WriteLine("Je bent " + age + " jaar oud.");
}
else
{
    Console.WriteLine("Dat is geen geldig getal.");
}
```

Zo voorkom je dat een gebruiker je programma crasht door “twintig” in te typen i.p.v. “20”.

8. Formatting

8.1. Waarom string formatting?

Stel: je wil een bericht tonen zoals:

```
Hello John, you have 5 new messages.
```

In plaats van zelf alle stukjes tekst aan elkaar te plakken, kan je gebruik maken van **string formatting**. Dat is duidelijker en voorkomt fouten.

8.2. string.Format()

Met `string.Format()` kan je **placeholders** (`{0}`, `{1}`...) gebruiken die vervangen worden door waarden.

```
string name = "John";
int messages = 5;

string result = string.Format("Hello {0}, you have {1} new messages.", name,
messages);
Console.WriteLine(result); // Hello John, you have 5 new messages.
```



de nummers in de accolades verwijzen naar de positie van het argument na de string.

8.3. String interpolatie (\$-strings)

Een modernere en leesbaardere manier is **interpolatie**:

```
string name = "John";
int messages = 5;

string result = $"Hello {name}, you have {messages} new messages.";
Console.WriteLine(result);
```

De `$` voor de string betekent dat je variabelen of expressies rechtstreeks in de tekst mag verwerken tussen accolades (`{}`):



- `{name}` wordt vervangen door de waarde van de variabele `name`, dus "John".
- `{messages}` wordt vervangen door de waarde van de variabele `messages`, dus 5.

Veel programmeurs verkiezen deze manier omdat het duidelijker is en je meteen ziet welke variabelen gebruikt worden.

8.4. Getallen en datums formatteren

Je kan ook een **specifieke opmaak** toepassen, bijvoorbeeld om het aantal decimalen te beperken of een datum netjes weer te geven.

8.4.1. Getallen

Decimalen

```
double price = 12.3456;  
Console.WriteLine($"Price: {price.ToString("F2")}""); // Price: 12,35  
// of  
Console.WriteLine($"Price: {price:F2}"); // Price: 12,35
```

F2 betekent: "2 cijfers na de komma".

Valuta

```
double price = 12.3456;  
Console.WriteLine($"Price: {price.ToString("c")}""); // Price: € 12,35  
// of  
Console.WriteLine($"Price: {price:c}"); // Price: € 12,35
```

c betekent "currency" (valuta)

- Afhankelijk van de regio-instellingen van het besturingssysteem kan het decimaalteken of de munteenheid verschillen
- Standaard wordt het euroteken niet correct weergegeven in een console-applicatie, voeg daarom `Console.OutputEncoding = System.Text.Encoding.UTF8;` als eerste regel toe in de `Main`-methode

8.4.2. Datums

```
DateTime today = DateTime.Now;  
Console.WriteLine($"Today is: {today:dd/MM/yyyy}"); // Today is: 20/05/2025
```

Hier bepaal je zelf de structuur van de datum:

- dd = dag
- MM = maand
- yyyy = jaar

Ook de structuur van de tijd kan je zelf bepalen:

- HH = uren (24-uurs notatie)
- mm = minuten

- ss = seconden



Bepaalde formaten zijn hoofdlettergevoelig!



- [Bekijk de volledige lijst van numerieke notaties op learn.microsoft.com](#)
- [Bekijk de volledige lijst van standaard datumnotaties op learn.microsoft.com](#)
- [Bekijk de volledige lijst van aangepaste datumnotaties op learn.microsoft.com](#)

8.5. Uitlijnen en opvullen

Je kan een waarde links of rechts uitlijnen en eventueel opvullen met spaties:

```
int number = 42;
Console.WriteLine($"|{number,5}|"); // | 42| (rechts uitgelijnd)
Console.WriteLine($"|{number,-5}|"); // |42| (links uitgelijnd)
```