Beslissingen

Zie Scherp Scherper - Hoofdstuk 5

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- **booleaanse expressies** schrijven met **relationele** en **logische** operators
- de **volgorde van bewerkingen** correct toepassen
- beslissingen nemen met `if`, `if/else` en `if/else if`
- de **scope** van een variabele bepalen
- een `switch` gebruiken in plaats van een lange `if`-keten
- een eigen `enum` maken en inzetten voor leesbare, veilige code

Van lineair naar beslissingen

Tot nu toe waren onze programma's een rechtlijnige lijst opdrachten:

```
Neem geld uit spaarpot
Wandel naar de bakker
Vraag om een brood
Betaal en keer huiswaarts
```

De realiteit is zelden zo rechtlijnig. Een beter algoritme **beslist** onderweg:

```
Geld op? Stop dan. Anders: ga verder.
Bakker toe? Stop dan. Anders: koop brood.
```

Booleaanse expressies

> Een **booleaanse expressie** is C#-code die een `bool` als resultaat geeft.

De operators in dit hoofdstuk zijn **test-operators**: ze testen of iets waar is of niet.

Relationele operators

Operator	Betekenis
`> <`	groter / kleiner dan
`>= <=`	groter / kleiner dan of gelijk aan
`==`	gelijk aan
`!=`	niet gelijk aan

- Twee operanden, resultaat is een **`bool`**
- De types moeten **vergelijkbaar** zijn (geen `string` met `int`)

> ⚠ **Waarschuwing**
>
> `=` kent een waarde toe, `==` test op gelijkheid. Een klassieke verwarring.

Logische operators

Voor samengestelde voorwaarden (“hongerig **EN** genoeg geld”):

- `&&` (EN): `true` als **beide** operanden `true` zijn
- `||` (OF): `true` als **minstens één** operand `true` is
- `!` (NIET): keert de waarde om

```
1  bool result = (4 < 6) && ("ja" == "nee");  // true && false  => false
```

Logische operators verwachten **`bool`**-operanden en geven een `bool` terug.

Volgorde van bewerkingen

Belangrijk zodra je expressies combineert:

1. `!` (logische NIET)
2. `* / %`
3. `+ -`
4. `< <= > >=`
5. `== !=`
6. `&&`
7. `||`

> **Tip**
>
> Twijfel je? Gebruik **haakjes**. Dat maakt je code leesbaarder en je bedoeling expliciet.

Test jezelf

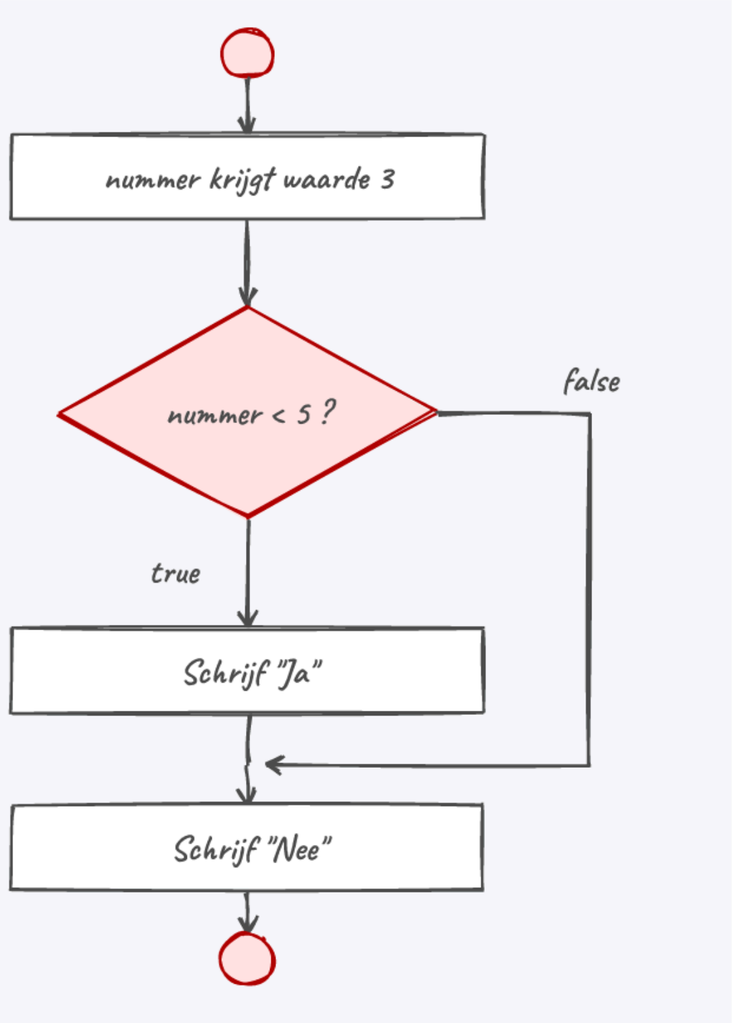
```
4 < 5 && 4 < 3
"a" == "A" || 4 >= 3
(3 == 3 && 2 < 1) || 5 != 4
!(4 <= 3)
```

- `false` (`true && false`)
- `true` (`false || true`, let op: hoofdlettergevoelig)
- `true` (`(true && false) || true`)
- `true` (`!false`)

If

```
1 int nummer = 3;
2 if (nummer < 5)
3 {
4     Console.WriteLine("Ja");
5 }
6 Console.WriteLine("Nee");
```

Het `if`-blok draait enkel als de expressie **`true`** is. De lijn erna draait **altijd**.



Flowchart.

Werk altijd met accolades

Zonder accolades hoort **enkel de eerste lijn** bij de `if`:

```
1  if (tijd < 20)
2      Console.WriteLine("Doe zo voort.");
3      Console.WriteLine("Je bent er bijna!"); // verschijnt ALTIJD
```

Tip

Inspringen heeft in C# **geen** invloed op de programmaflow (anders dan in Python). De accolades bepalen het blok.

Veelgemaakte fouten met if

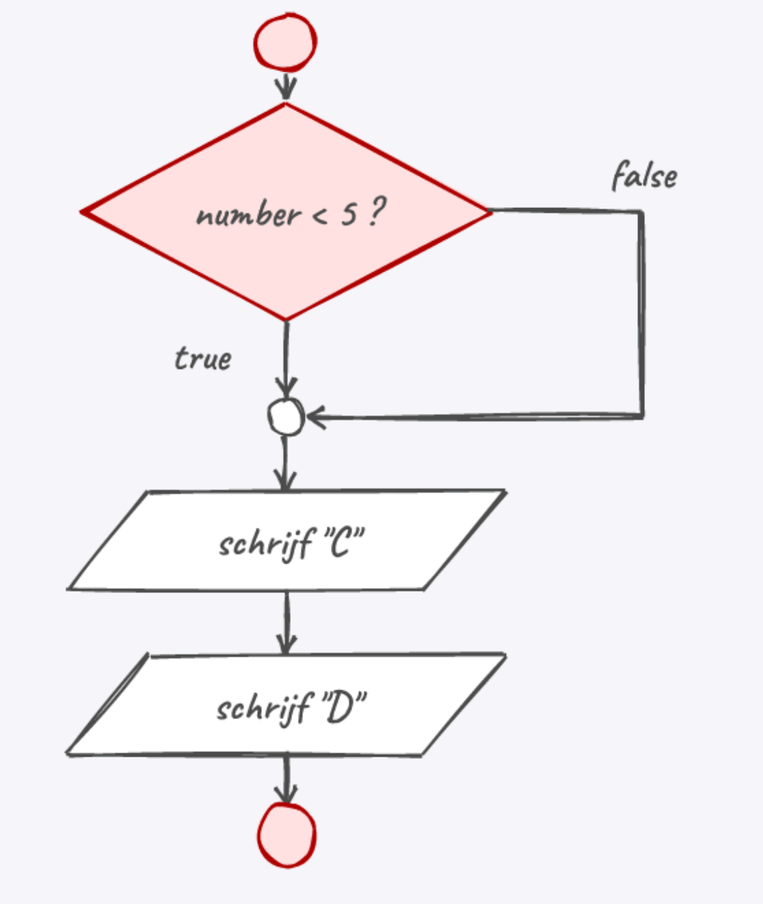
Voorman hier, even wakker schudden. Klassiekers met `if`:

- **appelen en peren** vergelijken (`"4" > 3`)
- **accolades vergeten**
- een **puntkomma** na de booleaanse expressie

De puntkomma-val

```
1 if (naam == "neo");
2 {
3     Console.WriteLine("Red pill?");
4     Console.WriteLine("Or blue pill?");
5 }
```

De `;` sluit de `if` meteen af. Het blok eronder draait **altijd**, ongeacht de naam.



De flow loopt langs de if.

If/else

```
if (waterpeil > MAX)
{
    Console.WriteLine("Waterpeil staat te hoog!");
}
else
{
    Console.WriteLine("Waterpeil is in orde.");
}
```

> ⚠ **Waarschuwing**
>
> Een `else` krijgt **geen** eigen booleaanse expressie. `else (a <= b)` is fout: de `else` draait gewoon wanneer de `if` niet waar was.

If - else if

```
if (x > 100)
    Console.WriteLine("Groter dan 100");
else if (x > 10)
    Console.WriteLine("Groter dan 10");
else
    Console.WriteLine("Kleiner dan of gelijk aan 10");
```

Belangrijk

De **volgorde is cruciaal**. Zet je `x > 10` bovenaan, dan wordt `x > 100` nooit bereikt: bij 110 slaat de eerste test al toe.

Nesting

Beslissingen in beslissingen:

```
1  if (huidigeTemperatuur < MAX_TEMP)
2  {
3      Console.WriteLine("Temperatuur normaal");
4  }
5  else
6  {
7      Console.WriteLine("Temperatuur te hoog!");
8      if (dokterVanWacht == "")
9          Console.WriteLine("Oei! Geen dokter van wacht!");
10     else
11         Console.WriteLine($"{dokterVanWacht} gecontacteerd");
12 }
```

Combineren met logische operators

```
if (leeftijd > 18 && heeftIdentiteitskaart)
    Console.WriteLine("Welkom");
else
    Console.WriteLine("Niet toegelaten!");
```

Tip

Schrijf `heeftIdentiteitskaart`, niet `heeftIdentiteitskaart == true`. Een `bool` is zelf al een booleaanse expressie. Testen op `false`? Schrijf `!heeftIdentiteitskaart`.

if blijft lastig

Laat al die uitleg je niet wijsmaken dat `if`-structuren eenvoudig zijn. Het is als met pijl en boog leren jagen: vlekkeloos op een stilstaande schijf, tot er een mammoet op je afkomt. *Da's andere kak hé?*

Teken zeker in het begin een **flowchart**. *Bezint eer ge begint.*

Scope van variabelen

De **omliggende accolades** bepalen waar een variabele leeft:

```
1 if (iLoveCSharp)
2 {
3     int getal = int.Parse(Console.ReadLine());
4 } // einde scope van getal
5
6 Console.WriteLine(getal); // FOUT: getal bestaat hier niet meer
```

Vergelijk het met kamers in een gebouw: wat je in een kamer maakt, blijft in die kamer.

Scope oplossen

Declareer de variabele **vóór** het blok waar je ze later nodig hebt:

```
1  int getal = 0;
2  if (iLoveCSharp)
3  {
4      getal = int.Parse(Console.ReadLine());
5  }
6  Console.WriteLine(getal); // ok
```

Belangrijk

Binnen eenzelfde scope mag je niet twee variabelen met **dezelfde naam** maken.

Switch

Een lange **`if/else if`**-keten korter schrijven:

```
switch (option)
{
    case 1:
        Console.WriteLine("Afbreken gekozen");
        break;
    case 2:
        Console.WriteLine("Opslaan gekozen");
        break;
    default:
        Console.WriteLine("Onbekende keuze");
        break;
}
```

Vier nieuwe keywords: **`switch`**, **`case`**, **`break`**, **`default`**.

Switch: de spelregels

- De **case-waarden** zijn **constanten** (literals), geen variabelen
- Ze moeten van **hetzelfde type** zijn als de testwaarde
- Vergelijking gaat **van boven naar onder**; geen match? Dan `default`

> **Tip**
>
> Accolades per case hoeven niet, maar elke case met code eindigt op `break`.

Fallthrough en de break-regel

```
case 2:
case 3:
    Console.WriteLine("Laden of opslaan gekozen");
    break;
```

Twee cases die dezelfde code delen. Dit werkt enkel omdat `case 2:` zelf **geen** code bevat.

> ⚠ **Waarschuwing**
>
> Kom je van C, Java of JavaScript: in C# **mag** een case met code niet “doorvallen”. Vergeet je `break`, dan **compileert** je code niet. C# behoedt je voor een klassieke bug.

Enum: de bestaansreden

Een dag van de week bijhouden zonder enum is foutgevoelig:

- met een `int` (1 tot 7): wat bij waarde 9 of -4? En was 2 nu dinsdag?
- met een `string` (`"maandag"`): één typefout (`"Maandag"`) en je test faalt

Enum: het beste van twee werelden

Een **`enum`** is een eigen datatype met een vaste set toegelaten waarden:

```
1  enum Weekdagen { Maandag, Dinsdag, Woensdag, Donderdag, Vrijdag, Zaterdag, Zondag };
```

- Leesbaarder en minder foutgevoelig
- IntelliSense vult de mogelijke waarden voor je in

> **Opmerking**
>
> Definieer het type binnen `class Program`, maar **niet** binnen `Main`.

Enum gebruiken

```
1  Weekdagen dagKeuze = Weekdagen.Donderdag;
2
3  if (dagKeuze == Weekdagen.Woensdag) { /* ... */ }
4
5  switch (dagKeuze)
6  {
7      case Weekdagen.Maandag:
8          Console.WriteLine("It's monday!");
9          break;
10     // ...
11 }
```

Variabelen zoals gewoonlijk, maar beperkt tot de gedefinieerde waarden.
Veel leesbaarder.

Enum is intern een int

```
1  int keuze = 3;
2  Weekdagen dag = (Weekdagen)keuze; // Donderdag (telt vanaf 0)
```

- Standaard genummerd vanaf **0** (`Maandag` = 0, `Dinsdag` = 1, …)
- Je kan dat zelf sturen:

```
1  enum Weekdagen { Maandag = 1, Dinsdag, Woensdag, Zaterdag = 50, Zondag = 60 }
```

Geen expliciet getal? Dan krijgt een waarde die van de vorige plus één.

Enum in de praktijk

Gebruik `enum` telkens je vooraf weet welke waarden mogelijk zijn. Klassiek bij *finite state machines*:

```
1  enum Gamestate { Intro, Startmenu, Ingame, Gameover }
```

Daarna een nette `switch` op `playerGameState`. Of denk aan schaken: `Schaakstuk.Paard`.

Conclusie

- **Booleaanse expressies** combineren relationele en logische operators tot een `bool`
- **`if`**, **`if/else`** en **`if/else if`**: let op accolades, puntkomma's en de **volgorde**
- De **scope** van een variabele wordt door de omliggende accolades bepaald
- Een **`switch`** verkort een lange `if`-keten; elke case met code eindigt op **`break`**
- Een **`enum`** maakt code leesbaarder en veiliger: een eigen type met vaste waarden

> **Tip**
>
> Volgende halte: **herhalingen** met `while`, `do-while` en `for`.

Meer weten

Kennisclips

- Logische operators en expressies
- If
- Scope van variabelen
- Switch
- Enum
- Demo: Enums gebruiken

Oefeningen H5 · 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: logische operators in Python

Python gebruikt woorden in plaats van symbolen:

```python
if leeftijd > 18 and heeft_kaart:
    print("Welkom")

resultaat = not (0 == 2)
```

Waar C# `&&`, `||` en `!` schrijft, leest Python bijna als een gewone zin (`and`, `or`, `not`). De werking is identiek.

Zie verder: if/else in Python

Python kent geen accolades. De **inspringing** bepaalt het blok, en het tussenstuk heet `elif`:

```
if leeftijd >= 18:
    print("meerderjarig")
elif leeftijd > 12:
    print("tiener")
else:
    print("kind")
```

In C# telt enkel de accolade en is inspringen louter cosmetisch. In Python is het net omgekeerd: de inspringing is verplicht en bepaalt de structuur.

Zie verder: switch in JavaScript

In **JavaScript** valt de uitvoer wél door naar de volgende case als je `break` vergeet:

```
switch (option) {
    case 1:
        console.log("Afbreken gekozen");
        // GEEN break: valt door naar case 2!
    case 2:
        console.log("Opslaan gekozen");
        break;
}
```

Bij `option` gelijk aan `1` verschijnen hier **beide** lijnen. Een beruchte bron van bugs. C# weigert net daarom code zonder `break` bij een case met inhoud.

Zie verder: enum in TypeScript

JavaScript kent geen `enum`. Maar **TypeScript** (JavaScript met types) wel, en die lijkt sterk op die van C#:

```
1 enum Weekdag { Maandag, Dinsdag, Woensdag, Donderdag, Vrijdag }
2
3 let keuze: Weekdag = Weekdag.Woensdag;
```

Net als in C# worden de waarden vanaf `0` genummerd en blijft een variabele beperkt tot deze waarden.

Samenvattende poster

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