Herhalingen

Zie Scherp Scherper - Hoofdstuk 6

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- uitleggen wat een **loop** en een **iteratie** zijn
- de drie soorten loops onderscheiden: **definite**, **indefinite** en **oneindig**
- loops schrijven met `while`, `do while` en `for`
- een **oneindige loop** herkennen en de **scope** in loops correct hanteren
- **geneste loops** schrijven en het aantal iteraties berekenen
- bewust omgaan met `break` en `continue`

Van branching naar herhalen

Met `if` konden we **vertakken**, maar niet **teruggaan** in ons algoritme.

> Soms moet een stuk code meerdere keren draaien tot aan een conditie voldaan is: *“Vraag input tot de gebruiker 666 ingeeft.”*

Elke ronde door een loop heet een **iteratie**.

> **Tip**
>
> Herhalende code in een loop is **korter**, **minder foutgevoelig** en beter onderhoudbaar.

Soorten loops

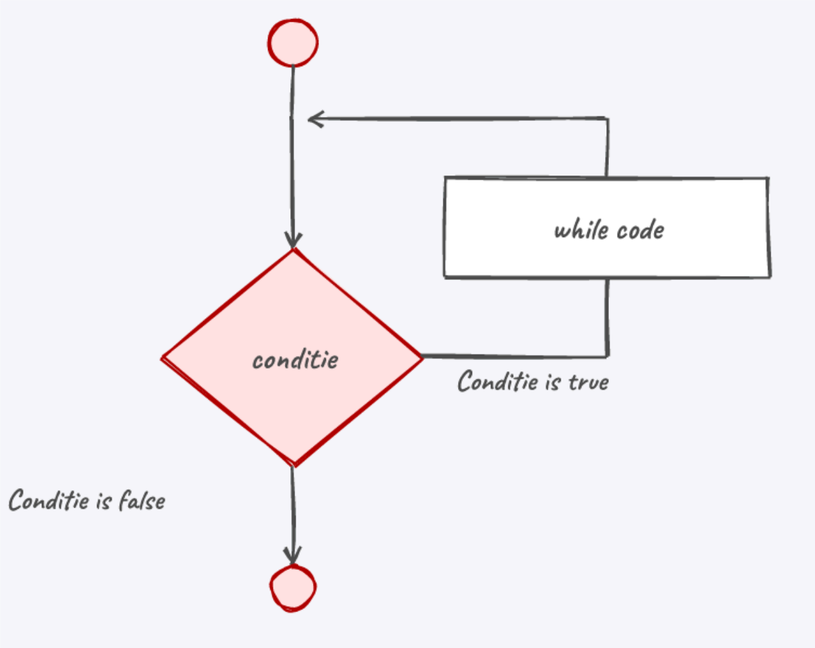
- **Definite** (counted): aantal herhalingen vooraf gekend (getallen 0 tot 100)
- **Indefinite** (sentinel): input of een test bepaalt wanneer het stopt
- **Oneindig**: stopt nooit (soms gewenst, zoals de game loop, vaak een bug)

Looptype	Definite	Indefinite	Oneindig
while / do while	x	x	x
for	x		

While

```
1 int tellertje = 0;
2 while (tellertje < 100)
3 {
4     tellertje++;
5     Console.WriteLine(tellertje);
6 }
```

Zolang de conditie **`true`** is, draait het blok. Is ze meteen `false`, dan draait het blok nooit.



While-flowchart.

> **Tip**
>
> Zie je dezelfde lijnen meermaals onder elkaar (copy-paste)? Grote kans dat een loop beter is.

Oneindige loops

Bewust oneindig:

```
while (true) { /* To infinity and beyond! */ }
```

Per ongeluk oneindig:

```
int teller = 0;
while (teller < 10)
{
    Console.WriteLine(teller);
    teller--; // oeps, dit had teller++ moeten zijn
}
```

> **Tip**
>
> Zorg dat de variabele uit je conditie **effectief verandert** in de loop. Vastgelopen? Stop met de **rode knop** in VS.

Scope in loops

Een variabele die je **in** de loop declareert, wordt elke iteratie **opnieuw** aangemaakt:

```
int teller = 1;
while (teller <= 10)
{
    int som = 0;     // FOUT: telkens terug op 0
    som = som + teller;
    teller++;
}
Console.WriteLine(som); // bestaat hier niet eens
```

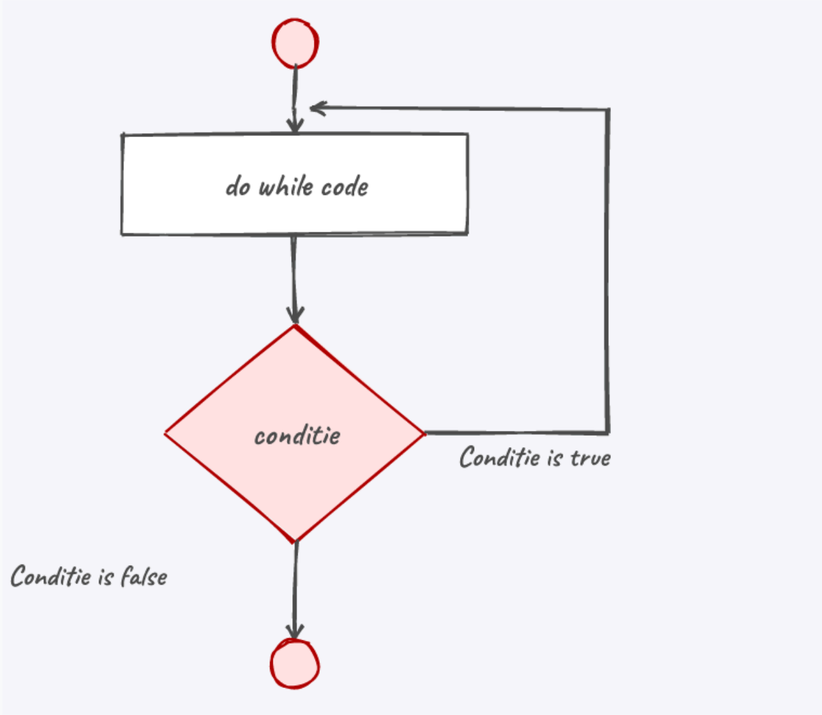
Belangrijk

Wil je een som **bijhouden** over de iteraties heen? Declareer `som` dan **vóór** de loop.

Do while

```
1 int i = 10;
2 do
3 {
4     i--;
5     Console.WriteLine(i);
6 } while (i > 0);
```

Een `do while` draait **minstens één keer**: de conditie wordt pas **na** elke iteratie getest.



Do-while-flowchart.

> ⚠ **Waarschuwing**
>
> Let op de **puntkomma** na `while (conditie);`. Vergeten is een klassieke fout (bij een gewone `while` staat ze er niet).

Foute input opvangen

`do while` is ideaal om te blijven vragen tot de input klopt:

```
1 string input;
2 do
3 {
4     Console.WriteLine("Geef uw keuze: a, b of c");
5     input = Console.ReadLine();
6 } while (input != "a" && input != "b" && input != "c");
```

Belangrijk

Declareer `input` **vóór** de loop: de test achteraan ligt **buiten** de accolades, dus in een andere scope.

Even tussendoor: De Morgan

Dezelfde logica, anders geschreven:

```
input != "a" && input != "b" && input != "c"
!(input == "a" || input == "b" || input == "c")
```

Tip

De **Wetten van De Morgan**:

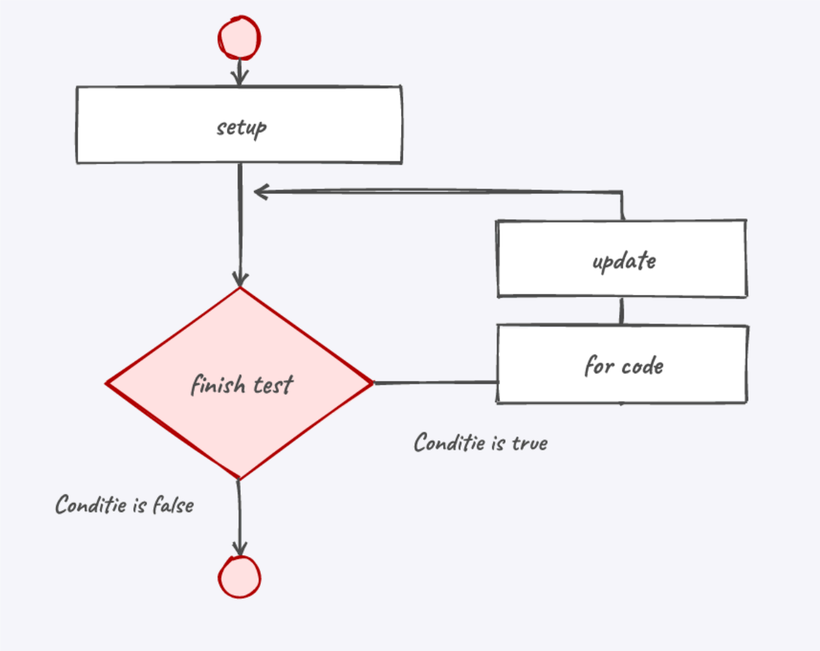
- `!(A && B)` is hetzelfde als `!A || !B`
- `!(A || B)` is hetzelfde als `!A && !B`

For

Een compacte loop wanneer je een teller bijhoudt:

```
for (int i = 0; i < 11; i += 2)
{
    Console.WriteLine(i);
}
```

Toont 0, 2, 4, 6, 8, 10.



For-flowchart.

For: de drie delen

```
for (setup; finish test; update)
```

- **setup** (*initializer*): zet de wachter-variabele op de beginwaarde (`int i = 0`)
- **finish test** (*condition*): de booleaanse test (`i < 11`)
- **update** (*iterator*): wat na elke iteratie gebeurt (`i += 2`)

> **Tip**
>
> Typ `for` + tweemaal **[tab]** in VS voor een kant-en-klare for-loop.

continue en break

```
1 for (int i = 1; i <= 10; i++)
2 {
3     if (i == 5) continue;   // sla 5 over
4     Console.WriteLine(i);
5 }
```

- **`continue`**: stop de huidige iteratie, ga naar de volgende
- **`break`**: stap meteen volledig uit de loop

De goto-politie, deel 2

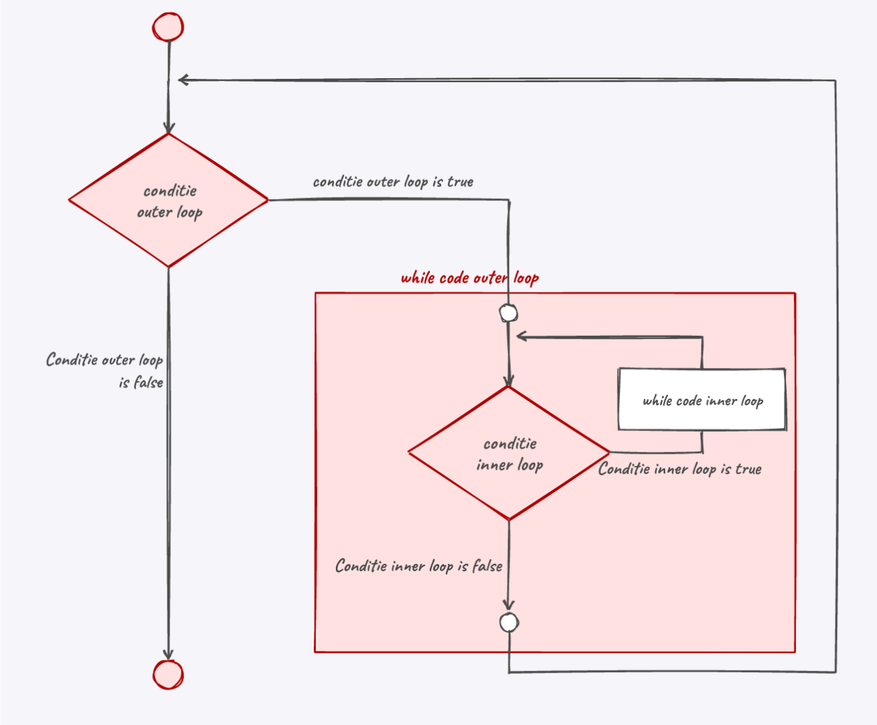
Olla!? Wat denken we dat we aan het doen zijn?

`break` en `continue` zijn de subtiele vrienden van `goto`. Probeer eerst je loop netjes te stoppen via een goede **stopconditie**.

> **Tip**
>
> Wanneer is `break` wél oké? Bij *zoek-en-stop*: zodra je gevonden hebt wat je zocht, is verder zoeken zinloos. Gebruik het enkel als het je code echt duidelijker maakt.

Geneste loops

```
1  while (tellerA < 3)          // outer
2  {
3      tellerA++;
4      tellerB = 0;
5      while (tellerB < 5)      // inner
6      {
7          tellerB++;
8          Console.WriteLine($"A:{tellerA}, B:{te
9      }
10 }
```



Geneste loops.

Belangrijk

`tellerB = 0` moet elke ronde **gereset** worden, anders blijft hij op 5 staan en draait de inner loop maar één keer.

Geneste loops tellen

```
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 5; j++)
    {
        Console.WriteLine("Hallo");
    }
}
```

Outer 10 keer, inner telkens 5 keer: **10 x 5 = 50** keer “Hallo”.

Belangrijk

`break` haalt je enkel uit de **huidige** loop. Uit alle geneste loops springen vraagt een andere aanpak (bv. een booleaanse vlag).

Conclusie

- Een **loop** herhaalt code; elke ronde is een **iteratie**
- `while` (0 of meer keer) en `do while` (minstens 1 keer; let op de `;`)
- `for` is compact wanneer je een teller bijhoudt: *setup*, *finish test*, *update*
- Let op **oneindige loops** en op de **scope** van variabelen in een loop
- Tel geneste loops door de iteraties te **vermenigvuldigen**; spaarzaam met `break`

> **Tip**
>
> Volgende halte: **methoden**, om je code op te delen in herbruikbare blokken.

Meer weten

Kennisclips

- Loops intro
- While en Do-while loops
- De for loop
- Loop nesting
- Micro-tips om loop-opgaven op te lossen

Oefeningen H6 · 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: do-while bestaat niet overal

De `do while` ken je terug in **C**, **C++**, **Java** en **JavaScript**. **Python** heeft hem niet. Iets dat minstens één keer draait, bootst men er na met `while True` + `break`:

```
while True:
    keuze = input("Geef uw keuze: a, b of c ")
    if keuze in ("a", "b", "c"):
        break
```

In Python ben je dus vaak verplicht om `break` te gebruiken waar C# een nette stopconditie achteraan de `do while` toelaat.

Zie verder: de for-loop elders

De drie-delige for komt uit **C** en is bijna letterlijk overgenomen door **C++**, **Java** en **JavaScript**:

```java
for (int i = 0; i < 11; i += 2) {
    System.out.println(i);
}
```

Python breekt ermee: geen teller-met-conditie, je loopt altijd *over* iets. Tellen doe je met `range`:

```python
for i in range(0, 11, 2):
    print(i)
```

Samenvattende poster

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