



// 01

Loops

Een loop voert een codeblok meerdere keren uit; elke uitvoering is een **iteratie**. Altijd erin: een **startsituatie**, een **conditie** (volgt er nog een iteratie?) en iets in het codeblok dat die conditie doet veranderen.

SOORT	AANTAL ITERATIES	KIES
definite	vooraf bekend	<code>for</code>
indefinite	invoer of een test beslist	<code>while</code> , <code>do while</code>
oneindig	stopt nooit: game loop of bug	<code>while</code>

// 02

while

Test **vooraf**, dus 0 of meer iteraties. De conditie zegt wanneer de loop **doorgaat**, niet wanneer hij stopt.

```
int getal = 0; // waarde vóór de eerste test
while (getal != 666)
{
    Console.WriteLine("Getal? (666 stopt)");
    getal = int.Parse(Console.ReadLine());
}
```

// LET OP

Pas de variabele uit je conditie in de loop aan, in de juiste richting. Anders blijft de test **true**: een **oneindige loop**. Stoppen doe je met de rode stop-knop in Visual Studio.

// 03

Scope in een loop

Een variabele die je binnen de accolades aanmaakt, wordt **elke iteratie opnieuw** aangemaakt en bestaat na de loop niet meer. Optellen of onthouden? Declareer **vóór** de loop.

```
int teller = 1;
int som = 0;
while (teller <= 10)
{
    som = som + teller;
    teller++;
}
Console.WriteLine(som); // teller is nu 11
```

UITVOER
55

// 04

do while

Test **achteraan**, dus minstens 1 iteratie. Ideaal om te blijven vragen tot de invoer klopt.

```
string input; // vóór de do: de test valt
do           // buiten de accolades
{
    Console.WriteLine("Keuze: a, b of c?");
    input = Console.ReadLine();
} while (input != "a" && input != "b"
        && input != "c");
```

Bedenk wanneer de invoer **goed** is (`input == "a" || ...`) en draai dat om met `!(...)`. De Morgan: `!(A || B)` is `!A && !B`.

// LET OP

Achter `} while (...)`; staat een **punktkomma**. Bij `while`, `if` en `for` net niet: `for (...)`; geeft een loop met een leeg codeblok.

// 05

for

Weet je vooraf hoe vaak? De drie stukken staan in de kop.

STUK	BV.	LOOP
setup	<code>int i = 0</code>	één keer, bij de start
finish test	<code>i < 11</code>	vóór elke iteratie (0 of meer keer)
update	<code>i += 2</code>	telkens ná het codeblok

```
for (int i = 10; i > 0; i--)
{
    Console.WriteLine(i + " ");
}
```

UITVOER
10 9 8 7 6 5 4 3 2 1

De teller uit de setup bestaat na de loop niet meer.

// 06

Hoeveel keer loopt hij?

KOP	I WORDT	KEER
<code>for (int i = 0; i < 10; i++)</code>	0 t.e.m. 9	10
<code>for (int i = 1; i < 10; i++)</code>	1 t.e.m. 9	9
<code>for (int i = 1; i <= 10; i++)</code>	1 t.e.m. 10	10
<code>for (int i = 0; i <= 10; i++)</code>	0 t.e.m. 10	11

// LET OP

Eén iteratie te veel of te weinig (**off-by-one**) is dé klassieker. Tel bij twijfel de eerste twee en de laatste twee rondes na.

// 07

Geneste loops

De **outer loop** overloopt de rijen, de **inner loop** draait per rij volledig rond.

```
for (int rij = 1; rij <= 3; rij++)
{
    for (int kolom = 1; kolom <= 5; kolom++)
    {
        Console.WriteLine(rij * kolom + "\t");
    }
    Console.WriteLine(); // einde van de rij
}
```

UITVOER
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15

Aantal keer de binnenste code: **vermenigvuldig**, $3 \times 5 = 15$. Niet meer als de inner loop de teller van de outer loop in zijn kop gebruikt (`kolom <= rij`).

// LET OP

Bij een geneste **while** zet je de teller van de inner loop elke ronde terug op zijn beginwaarde, anders loopt de inner loop maar één keer.

// 08

break en continue

- continue** springt naar de volgende iteratie: in jaar 1 **verboden**.
 - break** springt uit de loop, maar enkel uit de **huidige**. In jaar 1 enkel in een **switch** of bij **zoek-en-stop**.
- Liever een deftige stopconditie. Uit geneste loops: een **bool** in beide condities, `i <= 10 && !gevonden`.