



// 01

Char en string

Een `char` is één teken tussen apostrofs: `'A'`. Een `string` is een reeks van 0, 1 of meer `char`s: `"Hallo"`.

- **ASCII**: de oude standaard, 128 tekens in 7 bit
- **UNICODE**: de opvolger, met plaats voor meer dan 1 miljoen tekens. De eerste 128 zijn die van ASCII.
- Een `char` bewaart de UNICODE-waarde (16 bit): `'A'` is 65

```
Console.WriteLine('A' + 'B'); // 131
Console.WriteLine('7' + 1);    // 56
```

// LET OP

+ op `char`s telt hun getallen op: een `int`, geen `AB`. In `'7'` zit 55, niet 7: rekenen kan pas na een conversie (H4).

// 02

Escape characters

Een backslash geeft het teken erna een speciale betekenis.

ESCAPE	BETEKENIS	ESCAPE	BETEKENIS
<code>\'</code>	apostrof	<code>\n</code>	nieuwe regel
<code>\"</code>	aanhalingsteken	<code>\t</code>	tab
<code>\\</code>	backslash	<code>\u00A9</code>	UNICODE-teken

```
string zin = "Ik zei \"hoi\"\\nNaam:\\tTim";
Console.WriteLine(zin);
```

```
UITVOER
Ik zei "hoi"
Naam:  Tim
```

// 03

Verbatim en raw strings

Met een `@` voor de string zijn backslashes gewone tekens: escape characters werken niet. Een `"` schrijf je dan dubbel.

```
string pad1 = "C:\\Temp\\Myfile.txt";
string pad2 = @"C:\Temp\Myfile.txt"; // idem
Console.WriteLine(@"Hij zei "hallo"");
```

Een **raw string literal** (C# 11) staat tussen twee `"""`-lijnen. De inspringing tot de sluitende `"""` valt weg.

```
string kunst = """
+---+
| X |
""";
```

// LET OP

`"C:\temp\notities.txt"` toont een tab en een nieuwe regel, want `\t` en `\n` zijn escape characters. Zet er een `@` voor of schrijf `\\`.

// 04

UNICODE-teken tonen

De UTF-8-lijn moet de **allereerste lijn** in `Main` zijn. Zoek de hexadecimale code op in een UNICODE-tabel: `\u` met 4 cijfers, of `\U` met 8 voor een emoji.

```
Console.OutputEncoding =
    System.Text.Encoding.UTF8;
Console.WriteLine("@"); // plakken
Console.WriteLine((char)0x00A9); // casten
Console.WriteLine("\u00A9"); // hexadecimaal
Console.WriteLine("\U0001F600"); // emoji
```

Zonder die lijn wordt € een ? en een emoji twee vraagtekens.

// 05

String interpolation

Een `$` voor de string: alles tussen accolades is code, elke expressie mag. `.Length` geeft het aantal tekens.

```
string naam = "Finkelstein";
int leeftijd = 13;
Console.WriteLine($"Ik ben {naam}.");
Console.WriteLine($"{leeftijd} + 4} jaar");
Console.WriteLine($"{naam.Length} tekens");
```

UITVOER

```
Ik ben Finkelstein.
17 jaar
11 tekens
```

// LET OP

Vergeet je de `$`, dan verschijnt letterlijk `{naam}` op het scherm.

// 06

Formatteren

Na de expressie een dubbelpunt en een format specifier: `{number:F2}`. Er wordt **afgerond**, niet afgekapt.

FORMAAT	WAARDE	UITVOER	BETEKENIS
F2	12.345	12,35	2 cijfers na de komma
D5	123	00123	5 cijfers, gehele getallen
E2	12000000	1,20E+007	wetenschappelijk
C	12.5	€ 12,50	geldbedrag
000.00	12.3	012,30	masker met nullen

// LET OP

Komma of punt, € of \$: dat volgt de landinstellingen van de pc. Een Belgische pc toont € 12,34, een Amerikaanse \$12.34: niets omgerekend.

// 07

De +-operator op strings

+ plakt strings aan elkaar. Voor een zin met variabelen is interpolatie leesbaarder; + gebruik je vooral om lange tekst over meerdere lijnen te splitsen. Het werkt van links naar rechts: zodra een `string` meedoet, is het resultaat een `string`.

`"g" + 1` → `"gg" + 1` → `"ggg"`

`1 + 1` → `"2" + "g"` → `"2g"`

`"g" + (1 + 1)` → `"g" + 2` → `"g2"`

// 08

Environment

Info over de pc waarop je programma nu draait.

ENVIRONMENT.	GEEFT
<code>UserName</code>	wie aangemeld is
<code>MachineName</code>	de naam van de pc
<code>NewLine</code>	<code>\r\n</code> (Windows) of <code>\n</code> (Linux, macOS)
<code>WorkingSet</code>	geheugen van je programma (bytes)
<code>Exit(0)</code>	stopt meteen; exitcode 0 = alles goed

// LET OP

Wat `Environment` teruggeeft, verschilt per machine (`WorkingSet` zelfs per run). Test zulke code op meer dan één pc.