Python: If/Else/Elif

D. Leeuw

17 juni 2025
0.0.0

1 Condities

We kunnen een computer beslissingen laten nemen of een keuze laten maken. Dit doen we door een conditie op te geven en als deze waar is dan doet de computer iets, bijvoorbeeld twee getallen optellen. Een conditie is een opdracht die waar of niet-waar kan zijn en op basis daarvan voert de computer een taak uit, of niet.

Een conditie kan bijvoorbeeld zijn dat de waarde van een variabele 5 is. Ook groter dan of kleiner dan kunnen gebruikt worden.

Python kent verschillende operators om condities mee te bouwen. In de volgende secties zullen we een aantal operators behandelen.

1.1 Comparison operators

Met vergelijkingsoperators (comparison operators) kunnen twee waarden met elkaar vergelijken. Het vergelijken noemen we het testen op een conditie, waarbij er waar (True) of niet-waar (False) uit de vergelijking komt. In tabel 1 vind je de verschillende manieren waarop je een waarde kan vergelijken:

Operator	Betekenis	Hoe te gebruiken
==	is gelijk aan	x == y
!=	is niet gelijk aan	x != y
>	is groter dan	x >y
<	is kleiner dan	x <y
>=	is groter of gelijk aan	x >= y
<=	is kleiner of gelijk aan	x <= y

Tabel 1: Comparison operators

1.2 Logical operators

Met logische operators (logical operators) kunnen we twee condities tegelijk testen. Bij deze test is het geheel waar (True) of niet-waar (False). In de tabel 2 vind je een overzicht van hoe om te gaan met logical operators.

Met logische operator `not` kunnen we een False conditie True maken of een True conditie False.

Operator	Betekenis	Hoe te gebruiken
and	Als beide condities waar zijn, dan is het hele statement waar (True)	x <5 and y == 10
or	Als één van de condities waar is, dan is het hele statement waar (True)	x <5 or y == 10

Tabel 2: Logical operators

```
not(a == 5)
```

is dus hetzelfde als

```
a != 5
```

In combinatie met de `and` en de `or` wordt het dan:

```
not(x < 5 and y == 10)
```

Deze code betekent dat als `x` kleiner is dan 5 en `y` is 10 dan is de uitkomst **False**.

2 if - een keuze

Om de computer een keuze te laten maken gebruiken we het key-woord `if`. Bijvoorbeeld:

```
if a < 5:
    a += 1
```

Hier staat: "als a kleiner is dan 5 tel dan 1 op bij a". In alle andere gevallen doet de computer niets. We hebben de computer dus een simpele berekening laten uitvoeren op basis van de conditie of `a` kleiner dan 5 is. `a < 5` is de conditie. `if` geeft aan dat er een conditie komt. : zegt dat als de condite waar is dat dan de inspringende code uitgevoerd moet worden.

De conditie kent twee waarden: True en False. Let op binnen Python worden beide met de eerste letter als hoofdletter geschreven. We kunnen dan ook code maken met een True:

```
if True:
    a += 1
```

Deze `if`-conditie is altijd waar en dus zal altijd de `a` plus 1 uitgevoerd worden. Zinloze code dus, maar het laat wel zien dat een conditie waar of onwaar moet zijn.

De afspraak (syntax) binnen Python is dat code met spaties moet inspringen na een conditie. Alle inspringende code behoort bij de uit te voeren code na de `if`. Als het inspringen stopt gaat Python ervan uit dat het `if`-statement voorbij is. De keuze voor het aantal spaties is geheel naar de keuze van de programmeur, hoewel er veel gebruik wordt gemaakt van 4 spaties. Ook wij zullen deze ongeschreven regel volgen en altijd 4 spaties gebruiken bij het inspringen.

```
if a < 5:
    a += 1
print(a)
```

Het `print` commando wordt niet meer ingesprongen en behoort dus niet bij de `if`. De `print` wordt dus altijd uitgevoerd, ook als `a` groter is dan 5.

Als we code laten inspringen zonder dat er een statement is krijgen we een syntax-error van Python. Deze code:

```
a = 1
b = 2
   c =  a+b
print(c)
```

geeft bij het uitvoeren deze error:

```
IndentationError: unexpected indent
```

Er is dus een “identation” fout, indentation is het Engelse woord voor inspringen.

3 else - wat anders

Bij het `if`-statement hebben we gezien dat als er niet meer wordt ingesprongen dat het statement dan afgelopen is en dat de normale loop van het programma verder gaat. Het zou kunnen gebeuren dat we bij een conditie iets willen doen en in alle andere gevallen iets anders willen doen. We krijgen dan een als X doe iets, anders doe iets anders. In programmeerland noemen we dat `if` X doe iets, `else` doe iets anders:

```
if a < 5:
    a += 1
else:
    a += 2
    print(a)
exit()
```

Dus als `a` kleiner is dan 5 dan tellen we 1 op bij `a` in alle andere gevallen tellen we 2 op bij `a` en printen we de waarde van `a` op het scherm.

Ook na de `else` wordt er ingesprongen. In dit geval behoren dus de `a += 2` en de `print(a)` bij de code die uitgevoerd wordt na de `else`. De `exit()` behoort niet bij de `if`, niet bij de `else`, maar is onderdeel van de normale loop van het programma en wordt dus altijd uitgevoerd.

4 elif - een andere keuze

Bij het maken van een keuze hebben we nu gezien hoe we een keuze maken met `if`, wat we kunnen doen in alle ander gevallen met `else`, blijft er alleen nog over hoe we een tweede keuze kunnen maken. Binnen Python is er voor die laatse het `elif`-statement.

```
if a < 5:
    a += 1
elif a > 5:
    a += 2
else:
    a += 3
print(a)
```

Het gevolg van deze code is dat als `a` kleiner is dan 5 er 1 wordt opgeteld bij `a`, als `a` groter is dan 5 dan wordt er 2 opgeteld bij `a` en in alle andere gevallen (als `a` dus 5 is) dan wordt er 3 bij `a` opgeteld.

De `elif` mogen we een oneindig aantal keer herhalen, zodat we steeds een andere keuze kunnen maken. Om de code efficient te houden is het minimaliseren van het aantal keuzes aan te raden.

Ook het documenteren van code is belangrijk. Je hoeft daarbij niet uit te leggen wat de `if` constructie doet, je mag ervan uitgaan dat iemand dat kan lezen, maar het is wel belangrijk om het waarom van de keuze uit te leggen. Beantwoord in je uitleg de vraag wat betekent het voor de rest van het programma als de `if`, `elif` of de `else` doorlopen wordt.

Index