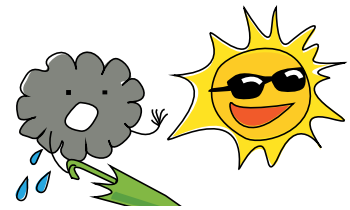
3

KEUZES MAKEN

Nu we hebben besproken hoe je constanten en variabelen maakt, ben je klaar om te leren hoe je jouw computer vertelt keuzes te maken. Dit hoofdstuk gaat over de flow van een computerprogramma: hoe vertel je de computer welke route hij door het programma moet nemen. Als we het over *flow hebben*, bedoelen we de volgorde waarin de instructies van het programma worden uitgevoerd.

Tot nu toe werden de uitdrukkingen alleen uitgevoerd in de volgorde waarin je ze getypt had. Je hebt daar al gave dingen mee gedaan. Maar door de computer te vertellen hoe hij keuzes kan maken over de volgorde waarin hij de instructies uitvoert, kun je nog meer doen. Om de computer een keuze te laten maken, gebruiken we *voorwaardelijke instructies* die de computer vertellen een code uit te voeren, gebaseerd op de waarde van een voorwaarde.

Je gebruikt elke dag al voorwaardelijke instructies om keuzes te maken! Voordat je 's ochtends van huis vertrekt, bekijk je bijvoorbeeld het weerbericht. Als de zon schijnt, zet je misschien een zonnebril op. Als het regent, pak je je paraplu. In beide gevallen controleer je een voorwaarde. Als de voorwaarde "het regent" waar is, neem je je paraplu mee als je naar buiten gaat. Wanneer de voorwaarde waar of onwaar kan zijn, heet dat een *Booleaanse uitdrukking*. Het `Bool` datatype waarover je geleerd hebt in Hoofdstuk 2 wordt gebruikt om de waarde `waar (true)` of `onwaar (false)` weer te geven.



BOOLEAANSE UITDRUKKINGEN

Een veelvoorkomende Booleaanse uitdrukking is er een die twee waarden vergelijkt met behulp van een *vergelijkende operator*. Er zijn zes vergelijkende operatoren. Laten we beginnen met twee eenvoudige: *is gelijk aan* en *is niet gelijk aan*.

IS GELIJK AAN EN IS NIET GELIJK AAN

Je gaat de *is gelijk aan* en *is niet gelijk aan* vergelijkende operatoren veel gebruiken. *Is gelijk aan* wordt geschreven als twee is-gelijktekens naast elkaar, zoals deze: `==`. *Is niet gelijk aan* wordt geschreven als een uitroepteken en één is-gelijk teken, zoals deze: `!=`.

Laten we ze allebei uitproberen in de playground!

```
❶ 3 + 2 == 5                                          true
  4 + 5 == 8                                          false
❷ 3 != 5                                              true
  4 + 5 != 8                                          true
  // Dit is verkeerd en levert een foutmelding op
❸ 3 + 5 = 8                                           error
```

In gewoon Nederlands, zegt de regel bij ❶ "drie plus twee is vijf " en dit is een `ware` bewering. De uitvoer in het deelvenster rechts zal dit bevestigen zodra je klaar bent met typen. Bij ❷ zegt de regel, "drie is niet gelijk aan vijf" en dat is ook een `ware` bewering. Merk op dat ❸ een fout is. Weet je waarom? Hoewel `=` en `==` veel op elkaar lijken, kent een enkel is-gelijk teken (`=`) waarden toe. Die instructie luidt: "Plaats de waarde van 8 in iets dat 3 + 5 heet" en dat werkt niet.

In Swift werkt de `==` operator ook met andere datatypes, en niet alleen met getallen. Laten we proberen enkele andere vergelijkingen te maken.

```
// Strings vergelijken
let mijnNaam = "Anne"
mijnNaam == "Melissa"
mijnNaam == "Anne"
❶ mijnNaam == "anne"
var mijnLengte = 171
mijnLengte == 171
// Dit is verkeerd en levert een foutmelding op
❷ mijnLengte == mijnNaam
```

```
"Anne"
false
true
false
67.5
true

error
```

De regel bij ❶ is een lastige. Had je verwacht dat die `waar` zou zijn? Deze twee strings lijken veel op elkaar, maar ze zijn niet precies hetzelfde. Een *is gelijk* vergelijking is alleen `waar` als de twee waarden precies gelijk zijn. De constante `mijnNaam` heeft de waarde `"Anne"` met een hoofdletter *A* en dat is niet hetzelfde als `"anne"` met een kleine letter *a*.

Herinner je je nog uit Hoofdstuk 2 dat we zeiden dat je geen rekenkundige operatoren zoals `+` en `*` kunt gebruiken voor dingen die niet hetzelfde datatype zijn? Hetzelfde geldt voor vergelijkingen. Je kunt geen dingen vergelijken die van verschillende types zijn. De regel bij ❷ zal een foutmelding geven, omdat de ene een `string` is en de andere een `Double`.

GROTER DAN EN KLEINER DAN

Laten we nu eens kijken naar vier andere vergelijkende operatoren. We beginnen met *groter dan* (geschreven als `>`) en *kleiner dan* (geschreven als `<`). Je hebt waarschijnlijk al een goed idee van hoe deze werken. Een Booleaanse uitdrukking als `9 > 7`, wat wil zeggen "9 is groter dan 7" is `waar`. Vaak wil je weten of iets groter dan of gelijk is aan iets of kleiner dan of gelijk aan iets. Hiervoor bestaan er nog twee operatoren: *groter dan of gelijk aan* (dat eruit ziet als `>=`) en *minder dan of gelijk aan* (dat eruit ziet als `<=`). Laten we deze eens uitproberen met wat meer voorbeelden:

```
// Groter dan
9 > 7
// Kleiner dan
9 < 11
// Groter dan of gelijk aan
❶ 3 + 4 >= 7
❷ 3 + 4 > 7
// Kleiner dan of gelijk aan
❸ 5 + 6 <= 11
❹ 5 + 6 < 11
```

```
true

true

true
false

true
false
```

Let op het verschil tussen *groter dan of gelijk aan* bij ❶ en *groter dan* bij ❷. De som van `3 + 4` is niet groter dan `7`, maar het is groter dan of gelijk aan `7`. Op dezelfde manier is `5 + 6` minder dan of gelijk aan `11` ❸, maar niet kleiner dan `11` ❹.

Tabel 3-1 geeft een overzicht van de zes vergelijkende operatoren.

Tabel 3-1: Vergelijkende operatoren.

Symbool	Definitie
==	Is gelijk aan
!=	Is niet gelijk aan
>	Is groter dan
<	Is kleiner dan
>=	Is groter dan of gelijk aan
<=	Is kleiner dan of gelijk aan

Je zult merken dat je deze operatoren vaak gebruikt wanneer je voorwaardelijke instructies schrijft.

SAMENGESTELDE BOOLEAANSE UITDRUKKINGEN

Samengestelde Booleaanse uitdrukkingen zijn eenvoudige Booleaanse uitdrukkingen die samengevoegd zijn. Het lijkt wel wat op het samenvoegen van zinnen in het Nederlands met de woorden *en* en *of*. Bij programmeren is er nog een derde mogelijkheid: *niet*. In Swift noemen we deze woorden *logische operatoren*. We gebruiken de Engelse woorden “and”, “or” en “not”. Een logische operator combineert een Booleaanse uitdrukking met een andere of ontkracht die. De drie logische operatoren in Swift kun je vinden in tabel 3-2.

Tabel 3-2: Logische operatoren.

Symbool	Definitie
&&	Logisch AND
||	Logisch OR
!	Logisch NOT

Met logische operatoren kun je uitdrukkingen schrijven die testen of een waarde binnen een bepaald bereik valt, zoals: “Ligt de leeftijd van deze persoon tussen de 10 en 15?” Dit kun je doen door te testen of de leeftijd tegelijkertijd groter is dan 10 *en* kleiner dan 15, zoals hier:

```
var leeftijd = 12
leeftijd > 10 && leeftijd < 15
```

```
12
true
```

De uitdrukking `leeftijd > 10 && leeftijd < 15` is `waar`, omdat beide voorwaarden `waar` zijn: `leeftijd` is groter dan 10 en kleiner dan 15. Een “AND”-uitdrukking is alleen `waar` als beide voorwaarden `waar` zijn.

Verander de waarde van `leeftijd` maar eens in 18 en kijk wat er gebeurt:

```
var leeftijd = 18
leeftijd > 10 && leeftijd < 15
```

```
18
false
```

Omdat we `leeftijd` nu veranderd hebben in 18, is maar één kant van de uitdrukking `waar`. De variabele `leeftijd` is nog steeds groter dan 10, maar niet meer kleiner dan 15, dus onze uitdrukking wordt `onwaar`.

Probeer nu "OR" uit door de volgende code in je playground in te voeren:

```
let naam = "Jacqueline"
naam == "Jack"
❶ naam == "Jack" || naam == "Jacqueline"
```

```
"Jacqueline"
false
true
```

Eerst verzinnen we een persoon met de naam Jacqueline door de constante `naam` in te stellen op `"Jacqueline"`. Vervolgens testen we enkele voorwaarden om te zien of ze `waar` of `onwaar` zijn. Omdat `naam == "Jacqueline"` `waar` is, is de "OR"-uitdrukking bij ❶ `waar`, hoewel `naam == "Jack"` `onwaar` is. In het Nederlands zegt deze uitdrukking: "De naam van deze persoon is Jack *of* de naam van deze persoon is Jacqueline". In een "OR"-uitdrukking hoeft slechts één van de voorwaarden `waar` te zijn om de hele uitdrukking `waar` te laten zijn.

Nu proberen we enkele "NOT"-uitdrukkingen. Voer de volgende code in je playground in:

```
let isEenMeisje = true
❶ !isEenMeisje && naam == "Jack"
isEenMeisje && naam == "Jacqueline"
❷ (!isEenMeisje && naam == "Jack") || (isEenMeisje && naam == "Jacqueline")
```

```
true
false
true
true
```

De `!` operator wordt gebruikt in de samengestelde Booleaanse uitdrukking ❶, die je kunt lezen als "Onze persoon is *niet* een meisje *en* onze persoon heet Jack". Deze uitdrukking heeft twee logische operatoren, namelijk `!` en `&&`. Je kunt bij het schrijven van samengestelde Booleaanse uitdrukkingen zoveel logische operatoren combineren als je wilt.

Soms is het een goed idee om haakjes te gebruiken om de computer te laten weten wat er eerst beoordeeld moet worden. Haakjes maken de code ook beter leesbaar. Dit is vergelijkbaar met de manier waarop je haakjes gebruikt wanneer je verschillende rekenkundige bewerkingen in één vergelijking gebruikt, zoals je in "Bewerkingen forceren met haakjes" op pagina 49 hebt kunnen lezen. Bij ❷ gebruiken we haakjes om de computer te vertellen dat hij eerst moet controleren op `!isEenMeisje && naam == "Jack"` en vervolgens op `isEenMeisje && naam == "Jacqueline"`. Na evaluatie van de twee delen kan de computer het "OF"-gedeelte beoordelen voor de gehele verklaring (die `waar` zal zijn, omdat het tweede deel `waar` is). Nogmaals, in een "OR"-verklaring is de hele uitdrukking `waar` als één van de voorwaarden `waar` is.

In tabel 3-3 zie je de drie logische operatoren en de samengestelde uitdrukkingen die je ermee kunt maken, met de bijbehorende Booleaanse waarden.

Tabel 3-3: Samengestelde Booleaanse uitdrukkingen met logische operatoren.

Logische operator	Samengestelde uitdrukking	Waarde
NOT (!)	!true	false
NOT (!)	!false	true
AND (&&)	true && true	true
AND (&&)	true && false	false
AND (&&)	false && true	false
AND (&&)	false && false	false
OR (||)	true || true	true
OR (||)	true || false	true
OR (||)	false || true	true
OR (||)	false || false	false

Het eerste item in de tabel laat zien dat iets dat NIET `waar` is, `onwaar` is. En iets dat NIET `onwaar` is, is dus `waar`.

Met de "AND"-operator is alleen iets dat `waar && waar` is `waar`. Dit betekent dat de uitdrukkingen aan beide kanten van de `&&` operator `waar` moeten zijn voordat de `&&` uitdrukking `waar` is. Een samengestelde uitdrukking die `waar && onwaar` is, geeft `onwaar`. En een samengestelde `&&` uitdrukking waarin beide voorwaarden `onwaar` zijn, zal ook `onwaar` opleveren.

Als het gaat om de "OR"-operator hoeft maar één van de uitdrukkingen aan beide kanten van de `||` operator `waar` te zijn om de `||`uitdrukking `waar` te laten zijn. Dus een `waar || waar` is `waar`, en een `waar || onwaar` is ook `waar`. Alleen een samengestelde "OR"-uitdrukking waarin beide zijden `onwaar` zijn, wordt uiteindelijk `onwaar`. Je ziet dat in de tabellen en de voorbeelden de Engelse woorden "true" (waar) en "false" (onwaar) gebruikt worden, omdat de Swift-programmeertaal in het Engels is.

VOORWAARDELIJKE INSTRUCTIES

Er bestaan twee soorten voorwaardelijke instructies: `if` instructies en `switch` instructies. Deze instructies geven de computer een voorwaarde op basis waarvan de computer een keuze maakt.

IF INSTRUCTIES

Een `if` instructie begint met het trefwoord `if`, gevolgd door een *voorwaarde* die altijd een Booleaanse uitdrukking is. De computer onderzoekt de voorwaarde en voert de code in de `if` instructie uit als de conditie `waar` is, en