Hoofdstuk 4

Variabelen

Als je met code werkt, ben je vaak bezig met het ontwerpen van een procedure (of "algoritme") dat een probleem op een algemeen toepasbare manier oplost. Bijvoorbeeld, in opgave 3.1 moest je de prijs van 60 boeken uitrekenen. De code die je schreef lost het probleem alleen op voor precies 60 boeken voor een bepaalde prijs. Als je een dergelijk probleem algemener wilt oplossen, moet je variabelen gebruiken om waardes in op te slaan.

4.1 Variables en waardes

Een variabele is een plaats in het geheugen van de computer die een naam heeft gekregen, en waarin je een waarde kunt opslaan. De naam mag je zelf kiezen, en wordt over het algemeen de "variabele naam" genoemd.

Om een variabele te creëren in Python (dus om een variabele naam te kiezen) moet je een waarde "toekennen" aan gekozen naam middels het is-gelijk (=) symbool. Aan de linkerkant van het is-gelijk symbool zet je de variabele naam, en aan de rechterkant de waarde die je wilt opslaan in de variabele. Dit kan ik het beste uitleggen aan de hand van een voorbeeld:

```
x = 5
print( x )
```

In deze code gebeuren twee dingen. Ten eerste wordt er een variabele gecrëerd met de naam `x` door er een waarde in op te slaan, in dit geval de waarde 5. In het Engels heet dit een "assignment", en het is-gelijk teken wordt ook wel de "assignment operator" genoemd. Ten tweede wordt de inhoud van de variabele `x` op het scherm getoond middels **`print()`**. Merk op dat **`print( x )`** niet de letter `x` toont, maar de waarde die in de variabele `x` is opgeslagen.

Je kunt je de variabele `x` voorstellen als een doos waarop je met een dikke, zwarte viltstift een `x` hebt geschreven, zodat je hem later gemakkelijk terug kunt vinden. Je kunt iets in de doos stoppen, en je kunt in de doos kijken om te zien wat er in zit (er kan wel slechts één ding tegelijkertijd in de doos zitten). Je kunt aan de inhoud van de doos refereren door de naam te gebruiken die je op de doos hebt geschreven. De term "variabele" duidt op de

variabele naam, dus de letter `x` op de doos. De term "waarde" duidt op de waarde die is opgeslagen in de variabele, dus de inhoud van de doos.

Aan de rechterkant van het is-gelijk teken kun je alles plaatsen wat een waarde oplevert. Het hoeft niet een enkel getal te zijn. Het mag ook een berekening zijn, of een string, of een aanroep van een functie die een waarde oplevert (bijvoorbeeld de **`int`**`()` functie).

Opgave In het vorige hoofdstuk was er een oefenopgave die je het aantal seconden in een week liet uitrekenen. Kopieer die berekening in een programma, en ken hem toe aan een variabele. Druk dan de variabele af.

De eerste keer in je programma dat je een waarde toekent aan een specifieke variabele naam, wordt de bijbehorende variabele gecreëerd. Als je later een andere waarde aan dezelfde variabele toekent, wordt de eerste waarde "overschreven." In de metafoor van de doos: je maakt de doos leeg en stopt er iets nieuws in. Een variabele bevat altijd de laatst-verkregen waarde.

```
x = 5
print( x )
x = 7 * 9 + 13   # overschrijft de vorige waarde van x
print( x )
x = "En nu iets heel anders..."
print( x )
x = int( 15 / 4 ) - 27
print( x )
```

Als een variabele is aangemaakt (en dus een waarde heeft) kun je hem overal in je code gebruiken waar je waardes gebruikt. Je kunt bijvoorbeeld de variabele gebruiken in een berekening.

```
x = 2
y = 3
```

```
print( "x =", x )
print( "y =", y )
print( "x * y =", x * y )
print( "x + y =", x + y )
```

Je kunt de inhoud van een variabele kopiëren in een andere variabele, via de assignment operator.

```
x = 2
y = 3
print( "x =", x, "en y =", y )

# Verwissel de waardes van x en y via z
z = x
x = y
y = z
print( "x =", x, "en y =", y )
```

Als je een waarde toekent aan een variabele, mag je zelfs de variabele zelf gebruiken aan de rechterkant van de toekenning, zolang de variabele maar bestaat op het moment dat je dat doet. De rechterkant van een assignment wordt altijd geheel geëvalueerd voordat de toekenning plaatsvindt.

```
x = 2
print( x )
x = x + 3
print( x )
```

Merk op dat de variabele gecreëerd moet zijn voordat je hem kunt gebruiken. De volgende code geeft een fout, omdat `dagen_per_year` nog niet gecreëerd is voordat hij gebruikt wordt in de eerste regel:

```
print( dagen_per_jaar )
dagen_per_jaar = 365
```

4.2 Variabele namen

Tot op dit punt heb ik slechts variabelen `x`, `y`, en `z` gebruikt (en een foute `dagen_per_jaar`). Je bent echter vrij om variabele namen te kiezen die je wilt, als je je daarbij maar aan een aantal eenvoudige regels houdt:

- Een variabele naam mag slechts bestaan uit letters, cijfers, en "underscores" (_)
- A variabele naam moet beginnen met een letter of een underscore.
- A variabele naam mag geen gereserveerd woord zijn

"Gereserveerde woorden" (of "keywords") zijn:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Je mag zowel hoofd- als kleine letters gebruiken in variabele namen, maar je moet wel beseffen dat Python "case sensitive" is, dus gevoelig voor de verschillen tussen hoofd- en kleine letters. Bijvoorbeeld, de variabele naam `wereld` is voor Python niet hetzelfde als de variabele naam `Wereld`.

4.2.1 Conventies

Programmeurs houden zich aan een flink aantal conventies wanneer ze variabele namen kiezen. De belangrijkste zijn de volgende:

- Programmeurs kiezen *nooit* een variabele naam die ook de naam is van een functie (of het nu een standaard Python functie betreft of een functie die ze zelf hebben geschreven). Als je dat doet, loop je de kans dat de functie niet langer door de code gebruikt kan worden, wat kan leiden tot uitermate vreemde fouten.
- Programmeurs proberen variabele namen zo te kiezen dat ze betekenisvol zijn in de context van het programma. Bijvoorbeeld, een variabele die het aantal seconden in een week bijhoudt, zou de naam `secs_per_week` kunnen hebben, maar niet `ik_haat_mijn_baan`. Het zou nog erger zijn om het aantal seconden in een week op te slaan in een variabele `secs_per_maand`.
- Een uitzondering op het kiezen van betekenisvolle variabele namen is het kiezen van namen voor "wegwerp variabelen," dat wil zeggen, variabelen die slechts in een klein deel van de code gebruikt worden, naderhand niet meer gebruikt worden, en die van zichzelf eigenlijk geen betekenis hebben. Programmeurs kiezen vaak één-letter namen voor dit soort variabelen, zoals `i` of `j`.
- Om verwarring tussen hoofd- en kleine letters te vermijden, gebruiken programmeurs meestal alleen kleine letters in variabele namen.
- Als een variabele naam uit meerdere woorden bestaat, zetten programmeurs underscores tussen die woorden.
- Programmeurs kiezen nooit variabele namen die beginnen met een underscore. Het gebruik van dat soort namen is voorbehouden aan de auteurs van Python.

Je moet proberen je voor je eigen code ook aan deze conventies te houden. De conventie met betrekking tot het kiezen van betekenisvolle variabele namen is vooral belangrijk, omdat het code leesbaar en onderhoudbaar maakt. Kijk bijvoorbeeld eens naar de volgende code:

```
a = 3.14159265
b = 7.5
c = 8.25
d = a * b * b * c / 3
print( d )
```

Snap je wat deze code doet? Je ziet waarschijnlijk wel dat er een benadering van π in voorkomt, maar wat stelt `d` voor?

Deze code berekent het volume van een kegel. Dat zou je waarschijnlijk niet geraden hebben, maar dat is wel wat er gebeurt. Nu vraag ik je de code zo te wijzigen dat het volume van een kegel met een hoogte van 4 meter berekend wordt. Welke wijziging maak je in de code?A Als de hoogte in de formule staat, is het waarschijnlijk `b` of `c`. Maar welk van de twee? Als je een beetje wiskunde kent, en je bekijkt de berekening van `d`, dan zie je dat `b` gekwadrateerd wordt in de berekening. Dat lijkt dus de voet van de kegel te zijn, wat zou betekenen dat `c` de hoogte is. Maar dat kun je niet zeker weten.

Bekijk nu de volgende, equivalente code:

```
pi = 3.14159265
straal = 7.5
hoogte = 8.25
volume_van_kegel = pi * straal * straal * hoogte / 3
print( volume_van_kegel )
```

Dat is een stuk leesbaarder, nietwaar? Als ik je nu vraag de code te wijzigen, verwacht ik niet dat je ook maar een moment zult twijfelen.

Code met betekenisvolle namen wordt vaak "zelf-documenterend" genoemd; je hoeft er geen commentaarregels in op te nemen om de gebruiker te laten begrijpen wat de code doet en hoe het werkt. Dat neemt niet weg dat in bovenstaande code een commentaarregel als:
`# berekening van volume van kegel met straal 7.5 en hoogte 8.25`
niet zou misstaan.

4.2.2 Oefenen met variabele namen

Opgave In de code hieronder wordt de waarde 1 toegekend aan een aantal (potentiële variabele namen. Sommige hiervan zijn correct, andere niet. Identificeer de incorrecte namen en leg uit waarom ze incorrect zijn.

```
classificatie = 1   # 1
Classificatie = 1   # 2
cl@ssificatie = 1   # 3
class1f1cat1e = 1   # 4
1classificatie = 1  # 5
_classificatie = 1  # 6
class = 1           # 7
Class = 1           # 8
```

Antwoord De derde, vijfde, en zevende zijn incorrect. De derde omdat er een at-sign (@) in zit, de vijfde omdat hij begint met een cijfer, en de zevende omdat het een gereserveerd woord is (dat gelukkig opvalt vanwege de syntax highlighting). De andere zijn weliswaar correct, maar de zesde zou volgens de conventie vermeden moeten worden omdat het begint met een underscore, en de tweede en achtste ook, omdat die een hoofdletter bevatten. De achtste is het ergst, want die lijkt op een gereserveerd woord.

4.2.3 Constanten

Veel programmeertalen geven je de mogelijkheid om "constanten" te creëren, wat waardes zijn die aan een variabele zijn toegekend, die geen andere waarde meer kan krijgen. Het is conventie dat alle letters in dit soort variabele namen hoofdletters zijn. Constanten kunnen gebruikt worden om code leesbaarder en onderhoudbaarder te maken. Bijvoorbeeld, om voor een rekening van €24,95 exclusief BTW het eindbedrag te berekenen, kun je de volgende code gebruiken:

```
totaal = 24.95
eind_totaal = int( 100 * totaal * 1.21 ) / 100
print( eind_totaal )
```

Het is echter leesbaarder om te schrijven:

```
BTW_FACTOR = 1.21
CENTEN = 100

totaal = 24.95
eind_totaal = int( CENTEN * totaal * BTW_FACTOR ) / CENTEN
print( eind_totaal )
```

Niet alleen is dit leesbaarder, het maakt het ook gemakkelijk om de code aan te passen als de BTW tarieven wijzigen. Zeker als constanten meerdere malen terugkeren in code, is het beter om ze eenmalig een waarde te geven bovenin de code, waar ze gemakkelijk gevonden en gewijzigd kunnen worden. Als het numerieke waarden betreft, zoals de BTW factor, spreekt men vaak over "magische getallen": getallen waarvan de waarde voor de code een speciale betekenis heeft, die niet duidelijk is als je alleen het getal ziet. Je bent dus beter af als je een betekenisvolle naam ziet in plaats van een getal.

Hoewel constanten erg nuttig kunnen zijn, worden ze niet door Python ondersteund (wat erg jammer is). Dat wil zeggen dat in de code hierboven `BTW_FACTOR` een reguliere variabele is die overal in de code gewijzigd kan worden. Het is echter de gewoonte om dit soort variabelen die volledig uit hoofdletters bestaan te beschouwen als constanten die niet in de code gewijzigd mogen worden nadat ze hun initiële waarde hebben gekregen. Ik raad je aan dit soort semi-constanten te gebruiken als er magische getallen in je code voorkomen.

4.3 Debuggen met variabelen

Een veelvoorkomende oorzaak van functionele fouten in programma's is dat variabelen blijken niet de waardes te bevatten waarvan je dacht dat ze ze bevatten. Een goede manier om je code te "debuggen" (dat wil zeggen, uit te vinden waar in je code fouten staan en die te verbeteren) is het printen van variabele namen op geschikte plaatsen. Bijvoorbeeld, de volgende code geeft een foutmelding als je hem uitvoert.

listing0401.py

```
nr1 = 5
nr2 = 4
nr3 = 5
```

```
print( nr3 / (nr1 % nr2) )
nr1 = nr1 + 1
print( nr3 / (nr1 % nr2) )
nr1 = nr1 + 1
print( nr3 / (nr1 % nr2) )
nr1 = nr1 + 1
print( nr3 / (nr1 % nr2) )
```

Misschien zie je wat het probleem is, maar stel dat je het niet ziet, hoe vind je dan uit wat er mis is? Je ziet dat de fout ontdekt wordt op regel 10, wat wil zeggen dat alles nog steeds werkte op regel 9. Als je een extra regel code zet tussen regel 9 en 10, die de waarde afdrukt van `nr1`, `nr2`, `nr3` and misschien ook `nr1%nr2`, dan ontdek je waarschijnlijk snel wat er misloopt. **`print()`** statements veranderen niks aan de variabelen, dus je kunt ze veilig toevoegen. Een fatsoenlijke manier om het probleem in deze code op te lossen (dus een andere manier dan gewoon de laatste regel te verwijderen) zal ik in een later hoofdstuk introduceren.

Opgave Voeg de extra regel toe aan de foute code.

4.4 Soft typing

Alle variabelen hebben een data type. In veel programmeertalen wordt het data type van een variabele gespecificeerd op het moment dat de variabele gecreëerd wordt. Bijvoorbeeld, in C++ zet je het type van een variabele voor de variabele naam op het moment dat je de variabele definieert, als volgt:

```
int secs_per_week = 7 * 24 * 60 * 60;
```

Dit wordt "hard typing" genoemd.[4] Hard typing heeft als voordeel dat als je waarde in een variabele probeert te stoppen van het verkeerde type, het programma een foutmelding kan geven. Dit vermijdt een aantal vervelende problemen die kunnen optreden tijdens het schrijven of uitvoeren van een programma.

In Python geef je niet een vast type aan een variabele, maar de variabele heeft wel een type, namelijk het type van de waarde die erin is opgeslagen. Dit betekent dat als een variabele een nieuwe waarde krijgt, het type ook kan veranderen. Dit wordt "soft typing" genoemd. (Nota bene: Ik ben persoonlijk van mening dat Python nog geschikter zou zijn om beginnelingen programmeren te leren als het hard typing in plaats van soft typing zou hebben, maar Guido van Rossum, de ontwerper van Python, is het daar niet mee eens.)

De data types die je tot nu toe gezien hebt zijn integer, float, en string. Je kunt het data type van een waarde of variabele zien met behulp van de functie **`type()`**.

```
a = 3
print( type( a ) )
a = 3.0
print( type( a ) )
a = "3.0"
print( type( a ) )
```

[4] Er bestaat geen Nederlandse benaming voor dit fenomeen.

Omdat variabelen een type hebben, past het effect van operatoren die tussen variabelen staan zich aan aan de types van de variabelen. Bijvoorbeeld, in de code hieronder wordt de optelling (+) twee keer gebruikt, en het effect verandert naar gelang de variabele types.

```
a = 1
b = 4
c = "1"
d = "4"
print( a + b )
print( c + d )
```

Omdat `a` en `b` beide getallen zijn, is de + in `a + b` de numerieke optelling. Omdat `c` en `d` beide strings zijn, is de + in `c + d` de "concatenatie" ("vastplak") operator.

Opgave Wat gebeurt er in de code hierboven als je `a + c` probeert te printen? Als je het niet weet, probeer het dan.

Opgave Wat toont de code hieronder? Denk erover na, en voer dan de code uit. Zorg dat je snapt wat er gebeurt.

```
naam = "John Cleese"
print( "naam" )
```

Opgave Wjzig de code hierboven zodat de naam van een bekend lid van Monty Python wordt getoond.

4.5 Verkorte operatoren

Middels de operatoren die ik heb uitgelegd, kun je de inhoud van variabelen zo vaak wijzigen in je code als je nodig hebt. Je kunt nieuwe waardes in bestaande variabelen stoppen. Vaak wil je dat inderdaad doen. Bijvoorbeeld, het komt in code vaak voor dat er 1 moet worden opgeteld bij een numerieke variabele (waarom dat zo vaak voorkomt zul je zien in hoofdstuk 7). Omdat dit zo vaak gebeurt, bevat Python een aantal " verkorte notaties" om de inhoud van variabelen aan te passen.

De volgende code:

```
aantal_bananen = 100
aantal_bananen = aantal_bananen + 1
print( aantal_bananen )
```

is hetzelfde als:

```
aantal_bananen = 100
aantal_bananen += 1
print( aantal_bananen )
```

Het verschil is de tweede regel. Als je iets wilt optellen bij een variabele, kun je += gebruiken als assignment operator, met de variabele aan de linkerkant en wat je erbij op wilt tellen aan de rechterkant. Dit bespaart de moeite van het twee keer typen van de variabele naam, en maakt je code over het algemeen leesbaarder (omdat programmeurs ervan uitgaan dat je de += operator gebruikt als je ergens iets bij wilt optellen).

Op vergelijkbare manier kun je -= gebruiken om iets af te trekken van een variabele, *= voor vermenigvuldiging, /= voor deling, **= voor machtsverheffing, etcetera. De meeste verkorte versies worden weinig gebruikt, behalve +=, die juist heel veel gebruikt wordt, en -=, die zo nu en dan gebruikt wordt.

Opgave Wat toont de code hieronder? Controleer of je gelijk hebt.

listing0402.py

```
aantal_bananen = 100
aantal_bananen += 12
aantal_bananen -= 13
aantal_bananen *= 19
aantal_bananen /= aantal_bananen
print( aantal_bananen )
```

4.6 Commentaar

De code die je vanaf dit punt schrijft zal vaak meer dan vijf regels lang zal zijn. Dat is voldoende aanleiding om het gebruik van commentaarregels te bediscussiëren. Commentaarregels zijn teksten in code die Python negeert tijdens de uitvoering, maar die bedoeld zijn om specifieke delen van de code uit te leggen. Commentaar is niet alleen van nut voor andere mensen die je code moeten bestuderen en/of wijzigen, maar ook voor jezelf, aangezien je soms je eigen code moet wijzigen weken of maanden nadat je de code oorspronkelijk geschreven hebt, en je niet meer precies weet wat je gedaan hebt.

Er zijn twee manieren om commentaar op te nemen in Python code. De eerste manier is door gebruik te maken van de "hash mark" (#), die aangeeft dat alles dat op dezelfde regel code rechts van de markering staat, commentaar is (mits de hash mark niet in een string staat). De tweede manier is door een stuk commentaar tekst, dat meerdere regels mag beslaan, vooraf te gaan door drie aanhalingstekens (dubbel of enkel), en dezelfde markering aan het einde van de tekst te zetten. In dit geval moet je de drievoudige aanhalingstekens aan het begin ook aan het begin van een regel zetten, en je kunt deze manier van commentaar geven niet gebruiken in een blok code dat inspringt. De reden is dat je feitelijk een string die uit meerdere regels bestaat in je code plaatst (meer hierover in hoofdstuk 10).

De volgende code laat voorbeelden zien van beide manieren van becommentariëren:

listing0403.py

```
# Commentaar: schrijf je code hier...
# Valt het je op dat alles rechts van de # commentaar is?
print( "Iets..." ) # dat door Python genegeerd wordt?
print( "en iets anders.." ) # Geef zo commentaar op je code!
"""Een andere manier van commentaar geven is middels
```

```
drievoudige aanhalingstekens. Dit soort commentaar mag
meerdere regels""" # beslaan.
'''dit mag ook via enkele aanhalingstekens''' # maar pas op
# met het gebruik van aanhalingstekens IN je commentaar als
# je de meerdere-regels method gebruikt.
print( "Klaar." )
```

Wat je geleerd hebt

In dit hoofdstuk is het volgende besproken:

- Wat variabelen zijn
- Het toekennen van een waarde aan een variabele
- Correcte namen voor variabelen
- Conventies met betrekking tot variabele namen
- Soft typing
- Het debuggen van code waarin variabelen onverwachte waardes hebben
- Verkorte operatoren
- Commentaar

Opgaves

Opgave 4.1 Definieer drie variabelen `var1`, `var2` en `var3`. Bereken het gemiddelde en stop het in een variabele `gemiddelde`. Toon het gemiddelde. Voeg drie commentaren toe.

Opgave 4.2 Schrijf code die de oppervlakte van een cirkel berekent, gebruik makend van variabelen `straal` en `pi` = 3.14159. Voor het geval je het vergeten bent, de formule is straal keer straal keer pi. Toon de uitkomst als volgt: "De oppervlakte van een cirkel met straal ... is ..."

Opgave 4.3 Schrijf code die een hoeveelheid centen (opgeslagen in een variabele met de naam `bedrag`) classificeert als een combinatie van grotere geldstukken. Je code gebruikt dollars (100 ct), kwartjes (25 ct), dubbeltjes (10 ct), stuivers (5 ct), en centen (1 ct). Je programma begint met het bepalen hoeveel dollarstukken er in het bedrag passen, dan hoeveel kwartjes er in het restbedrag zitten nadat de dollars eruit genomen zijn, dan de hoeveelheid dubbeltjes, dan de stuivers, en tenslotte de centen. Het resultaat is dat je het bedrag uitdrukt in het minimale aantal muntjes dat nodig is.

Opgave 4.4 Kun je een manier bedenken om de inhoud van twee numerieke variabelen om te wisselen zonder daarbij gebruik te maken van een derde hulp-variabele? Basiscode hiervoor is gegeven in het code blok hieronder. Probeer de inhoud van `a` en `b` te verwisselen zonder een derde variabele erbij te halen. Om je te helpen, is de eerste stap al gegeven. Je hoeft slechts twee regels code toe te voegen.

exercise0404.py

```
a = 17
b = 23
print( "a =", a, "en b =", b )
a += b
# Voeg hier twee regels toe om a en b om te wisselen
print( "a =", a, "en b =", b )
```