Hoofdstuk 3

Expressies

Welkom bij het eerste echte programmeerhoofdstuk. In dit hoofdstuk bespreek ik "expressies," die je kunt beschouwen als berekeningen die je ook kunt uitvoeren met een simpele rekenmachine. Het is een klein begin, maar deze expressies ga je nodig hebben in alle volgende hoofdstukken.

3.1 Resultaten tonen

Als je een expressie schrijft in de Python shell, en hem uitvoert, wordt het resultaat van de expressie eronder getoond. Bijvoorbeeld, als je het volgende commando in de shell schrijft en op `Enter` drukt, zie je het resultaat 12.

```
5 + 7
```

Echter, zoals ik liet zien in opgave 2.2, als je een programma schrijft dat alleen het commando `5 + 7` bevat, dan zie je geen resultaat. In plaats daarvan moet je voor programma's altijd expliciet aangeven dat je resultaten wilt tonen, zelfs als het gaat om een commando dat op de laatste regel van het programma staat.

Ook al gaat dit hoofdstuk over expressies, ik moet toch eerst iets uitleggen dat geen expressie is, maar een functie, namelijk een functie die het mogelijk maakt resultaten te laten zien. De functie die dat doet is **`print`**. Ik heb deze functie al gebruikt in hoofdstukken 1 en 2.

De **`print`** functie gebruik je als volgt: je schrijft het woord **`print`**, gevolgd door een rond openingshaakje, gevolgd door hetgeen je wilt laten zien, gevolgd door een rond sluithaakje. Bijvoorbeeld (en ook dit commando heb ik meerdere malen laten zien):

```
print( "Hello, world!" )
```

Als je deze code draait (door het op te slaan in een Python bestand en het te draaien in IDLE), zie je dat de tekst "Hello, world!" in de shell wordt getoond.

Overigens is het gebruikelijk dat wanneer auteurs van teksten over programmeren een functie bij naam noemen, ze er een openings- en sluithaakje achter zetten, om aan te geven dat het een functienaam betreft. Vanaf nu zal ik dat ook doen. Bovendien gebruiken auteurs soms niet het woord "functie," maar het woord "statement" of "commando." Deze termen duiden echter meestal op alles wat Python kan uitvoeren, niet alleen functies. Bijvoorbeeld, een expressie is ook een "commando."

Je kunt meerdere dingen tegelijkertijd laten zien met een **print()** functie, door alles wat je wilt laten zien tussen de haakjes te zetten, met komma's ertussen. De **print()** functie laat dan al die items zien, met spaties ertussen. Bijvoorbeeld:

```
print( "Ik", "heb", "twee", "appels", "en", "een", "banaan" )
```

De spaties in dit commando zijn overigens overbodig.

```
print("Ik","heb","twee","appels","en","een","banaan")
```

is precies hetzelfde als het commando ervoor. Je kunt zulke spaties toevoegen om de leesbaarheid te vergroten. Je mag ook spaties zetten tussen het woord **print** en het openingshaakje, maar de gewoonte is om bij functies altijd het openingshaakje aan de naam van de functie "vast te plakken."

Merk op dat je met **print()** niet alleen teksten, maar ook getallen kunt laten zien. Je mag zelfs teksten en getallen door elkaar gebruiken.

```
print( "Ik", "heb", 2, "appels", "en", 1, "banaan" )
```

Opgave Laat een paar teksten zien met behulp van een Python programma. Let erop dat je alle teksten tussen aanhalingstekens moet plaatsen; je mag naar keuze dubbele of enkele aanhalingstekens gebruiken.

3.2 Data types

Voordat ik toekom aan expressies, is er nog een onderwerp dat aan bod moet komen, en dat is data types. Er zijn drie verschillende data types die je op dit moment moet kennen, en dat zijn strings, integers, en floats.

3.2.1 Strings

Een string is een tekst, die bestaat uit nul of meer tekens, omsloten door aanhalingstekens. Je mag dubbele of enkele aanhalingstekens gebruiken. Wat je kiest maakt niet uit, bijvoorbeeld: `"appel"` is equivalent met `'appel'`. Echter, als je tekst een enkel aanhalingsteken bevat, moet je hem om problemen te voorkomen omsluiten met dubbele aanhalingstekens; dus `"mango's"` is een correcte string, terwijl `'mango's'` niet correct is. Hetzelfde geldt natuurlijk voor een dubbel aanhalingsteken in een string, die dan omsloten moet worden door enkele aanhalingstekens.

Maar wat moet je doen als een string zowel dubbele als enkele aanhalingstekens bevat? Je kunt dat oplossen door in de string een "backslash" (\) op te nemen voor ieder dubbel of

enkel aanhalingsteken dat in de string staat. Dit vertelt Python dat dat aanhalingsteken behandeld moet worden als een teken in de string, en niet als een afsluiting van de string. Dus `'mango\'s'` is een correcte string, wat je kunt zien als je hem probeert te printen:

```
print( 'mango\'s' )
```

Maar wat moet je dan doen als je een echte backslash wilt opnemen in de string, en die backslash moet dan ook nog eens staan voor een dubbel of enkel aanhalingsteken? Dat kun je oplossen door voor de backslash een extra backslash op te nemen, wat er een teken in de string van maakt in plaats van een aanduiding dat wat erachter komt letterlijk genomen moet worden. Bijvoorbeeld, controleer wat de volgende code doet als je hem uitvoert (je kunt hem intypen in de Python shell):

```
print( 'mango\\\'s' )
```

Als je dit verwarrend vindt kun je het voorlopig vergeten; ik ga het hierover in een later hoofdstuk nog uitgebreid hebben. Voor dit moment is het voldoende om te weten dat een string een tekst is die omsloten is door dubbele of enkele aanhalingstekens. Een string kan iedere lengte hebben, inclusief nul tekens lang.

Let erop dat je alleen "rechte" aanhalingstekens gebruikt in Python programma's, en niet "ronde." Tekstverwerkers hebben de neiging om rechte aanhalingstekens automatisch te vervangen door ronde, maar Python herkent zulke ronde aanhalingstekens niet. Tekst editors doen dat niet, maar als je om wat voor reden dan ook Python code kopieert naar een tekstverwerker, dan zou het best kunt gebeuren dat je aanhalingstekens gewijzigd worden. Kijk daarvoor uit.

3.2.2 Integers

Integers zijn gehele getallen, die positief of negatief (of nul) kunnen zijn. Er is een zekere grens aan hoe groot integers kunnen worden, die afhankelijk is van je computer en besturingssysteem. Voor de meeste toepassingen maakt dit niet uit, en kom je nooit aan die grenzen toe. Python is niet als rekenmachines met een 10-cijfer display.

Er zijn meerdere manieren mogelijk om een specifieke integer-waarde te schrijven. `1` is hetzelfde als `+1` (er zijn nog meer manieren om `1` te schrijven, maar die volgen in een later hoofdstuk). Dus zowel `print( 1 )` als `print( +1 )` geven hetzelfde resultaat. Voor strings is dat natuurlijk anders: de string `"1"` is niet hetzelfde als de string `"+1"`.

Als je integers in Python gebruikt, mag je ze niet schrijven met scheiders tussen de veelvouden van 1000 om ze leesbaarder te maken. Je moet het getal 1 miljard dus schrijven als `1000000000` en niet als `1,000,000,000` (de Engelse conventie) of `1.000.000.000`.

Bestudeer de volgende code en bedenk wat er gebeurt als je hem uitvoert. Kopieer hem daarna naar de Python shell en voer hem uit.

```
print( 1,000,000,000 )
```

Opgave Als je voorspelling van wat de code doet niet correct is, probeer dan te bedenken waarom de code deze uitkomst geeft.

3.2.3 Floats

Floats, wat kort is voor "floating-point getallen" ("gebroken getallen"), zijn getallen met decimalen. Bijvoorbeeld, `3.14159265` is een float. Merk op dat je een punt in plaats van een komma moet gebruiken als decimaal-scheider. Veel landen (inclusief Nederland) gebruiken een komma als decimaal-scheider, maar Python houdt zich aan de conventie van Engelstalige landen en gebruikt daarom de punt.

Als je een integer hebt die je wilt gebruiken als float, kun je dat doen door er `.0` achter te zetten. Bijvoorbeeld, `13` is een integer, maar `13.0` is een float. Ze geven nog steeds dezelfde waarde weer, en als je ze in code met elkaar vergelijkt (dat bespreek ik in hoofdstuk 6), dan zal Python stellen dat ze hetzelfde zijn.

Er zijn bepaalde begrenzingen aan de grootte van de floats, en aan de precisie. Het is onwaarschijnlijk dat je ooit de maximale groottes bereikt, aangezien Python wetenschappelijke notatie voor grote getallen gebruikt. Maar als je Python gebruikt voor zeer exacte berekeningen, kun je wel in de problemen komen met precisie. Voor de meeste toepassingen gebeurt dat niet, maar als je een natuurkundige bent wiens berekeningen grote getallen omvatten op quantumniveau, moet je je wel bewust zijn van deze beperkingen.

Door de manier waarop Python floats opslaat, kunnen bepaalde getallen niet precies vastgelegd worden. Bijvoorbeeld, het statement `print( (431 / 100) * 100 )` geeft als antwoord 430.99999999999994, en niet 431 zoals je zou verwachten. Als je weet dat de uitkomst van een berekening waarin floats zitten een integer moet zijn, dan doe je er goed aan om de uitkomst af te ronden naar het dichtstbijzijnde gehele getal. Dat kun je doen met behulp van de `round()` functie, die ik bespreek in hoofdstuk 5.

3.3 Expressies

Dan kan nu eindelijk het onderwerp van dit hoofdstuk onder de loep genomen worden, namelijk "expressies." Een expressie is een combinatie van één of meerdere waardes (zoals strings, integers, of floats) met behulp van operatoren, die dan een nieuwe waarde oplevert. Je kunt je expressies dus voorstellen als berekeningen.

3.3.1 Eenvoudige berekeningen

Eenvoudige berekeningen worden gemaakt door twee waardes te combineren met een operator ertussenin. Een aantal voor de hand liggende operatoren zijn:

```
+   optelling
-   aftrekking
*   vermenigvuldiging
/   deling
//  integer deling
**  machtsverheffing
%   modulo
```

Hier volgen een paar voorbeelden:

```
print( 15+4 )
print( 15-4 )
```

```
print( 15*4 )
print( 15/4 )
print( 15//4 )
print( 15**4 )
print( 15%4 )
```

Ik neem aan dat je weet wat deze operatoren voorstellen, behalve misschien de integer deling en de modulo.

De integer deling (ook wel genoemd "floor division") is simpelweg een deling die naar beneden afrondt naar een geheel getal. Als er floats in de berekening zitten, is het resultaat nog steeds een float, maar naar beneden afgerond. Als de berekening alleen integers omvat, is het resultaat een integer.

De modulo operator (%) produceert de rest die overblijft na deling. Bijvoorbeeld: als ik 14 deel door 5, is de uitkomst 2.8. Dat betekent dat ik twee keer 5 kan aftrekken van 14, en dan nog steeds een positief getal overhoud, maar als ik het een derde keer aftrek wordt het resultaat negatief. Dus als ik 5 twee keer aftrek van 14, rest er een getal kleiner dan 5. Deze rest is wat de modulo operator oplevert.

In eenvoudige termen: als ik 14 koekjes heb die ik moet verdelen over 5 kinderen, kan ik ieder kind 2 koekjes geven. Ik heb dan nog 4 koekjes over, omdat ik dan meer kinderen dan koekjes heb. Dus als je 14 deelt door 5 met integer deling, geeft dat 2 (koekjes per kind), terwijl 14 modulo 5 als rest 4 (koekjes in mijn hand) geeft.

Als zijdelingse opmerking: de code hierboven bestaat uit meerdere regels. Iedere regel is één "statement," bestaande uit een commando dat Python uitvoert (in de code hierboven is dat voor iedere regel een `print()` commando). De meeste programmeertalen stellen het als een verplichting dat ieder statement eindigt met een speciaal teken, bijvoorbeeld een puntkomma (;). Python verlangt dat niet, maar dan moet ieder statement ook op zijn eigen regel staan. In principe mag je meerdere Python statements op één regel zetten, maar dan moeten er puntkomma's tussen de statements staan. In de praktijk doen Python programmeurs dat niet, omdat het code lelijk, slecht leesbaar, en slecht onderhoudbaar maakt. Dus ik stel voor dat je de Python-gewoonte volgt en ieder statement zijn eigen regel geeft.

3.3.2 Complexe berekeningen

Je mag waardes en operatoren combineren om grotere berekening te maken, net zoals je kunt met geavanceerde rekenmachines. Je mag daarbij haakjes gebruiken om de evaluatievolgorde te bepalen, en je mag die haakjes zelfs nesten. Zonder haakjes zal Python de operatoren evalueren in de volgorde die wiskundigen gebruiken, waarvoor op basisscholen vaak de zin "Meneer Van Dalen Wacht Op Antwoord" wordt gebruikt (machtsverheffen, vermenigvuldigen, delen, worteltrekken, optellen, aftrekken).[3]

Bekijk de berekening hieronder en probeer te bepalen wat de uitkomst is voordat je hem uitvoert in de Python shell.

[3] In het Engels wordt de volgorde PEMDAS genoemd, en eigenlijk geeft die beter aan wat daadwerkelijk de volgorde is: haakjes (parentheses), exponenten, multiplicatie en divisie (deling), additie (optelling) en subtractie (aftrekking). Feitelijk is de Nederlandse volgorde incorrect, omdat worteltrekken na delen wordt geplaatst, terwijl wiskundig gezien worteltrekken een vorm van machtsverheffen is, en dus voor vermenigvuldigen komt.

```
print( 5*2-3+4/2 )
```

Ik moet een paar opmerkingen maken over deze berekening.

Op de eerste plaats valt het op dat de uitkomst een float is (zelfs al zijn er geen decimalen). De reden is dat er een deling in de berekening zit, en dat betekent voor Python dat de uitkomst automatisch een float is.

Op de tweede plaats is het zo dat, zoals ik al eerder opmerkte, spaties door Python genegeerd worden. De code hierboven is dus equivalent met:

```
print( 5 * 2 - 3 + 4 / 2 )
```

Het is zelfs gelijk aan:

```
print( 5*2 - 3+4    / 2 )
```

Ik heb lange discussies moeten voeren met mensen die beweren dat de code hierboven als uitkomst 6.5 of 1.5 geeft, want het is toch *overduidelijk* dat je eerst $5 * 2$ en $3 + 4$ moet berekenen voordat je toekomt aan die aftrekking en deling. Dat is kolder. Het maakt niet uit hoeveel spaties je rondom de operatoren zet: spaties worden genegeerd. Als je echt eerst $3 + 4$ wilt laten berekenen, moet je er haakjes omheen zetten. Spaties kunnen leesbaarheid verhogen als je ze goed toepast, maar voor Python zijn ze betekenisloos.

```
print( (5*2) - (3+4)/2 )
print( ((5*2)-(3+4)) / 2 )
```

Opgave Nu is de tijd gekomen om je eerste programma te schrijven. Schrijf een programma dat het aantal seconden in een week berekent. Je moet daarvoor natuurlijk niet je rekenmachine of smartphone grijpen, daarop de berekening doen, en dan gewoon de uitkomst afdrukken. Je moet de berekening doen in Python code. Het programma is slechts één regel lang, dus je kunt het gewoon in de Python shell doen, maar ik raad je aan om echt een programmabestand te maken.

3.3.3 String expressies

Een aantal van de hierboven genoemde operatoren kunnen voor strings worden gebruikt, maar niet allemaal.

Specifiek, je kunt de plus (+) gebruiken om twee strings aan elkaar te "plakken," en je kunt de ster (*) gebruiken met een string en een integer om een string te maken die een herhaling van de originele string bevat. Zie hier:

```
print( "tot"+"ziens" )
print( 3*"hallo" )
print( "tot ziens"*3 )
```

Je kunt geen getal optellen bij een string, of twee strings met elkaar vermenigvuldigen. Zulke operatoren zijn niet gedefinieerd, en geven foutmeldingen. Geen van de andere operatoren werkt voor strings.

3.3.4 Type casting

Soms moet je een data type of waarde veranderen in een ander data type. Je kunt dat doen met "type casting" functies.

In latere hoofdstukken (5 en 8) ga ik in detail op functies in, maar voor nu is het voldoende als je weet dat een functie een naam heeft, en parameters kan hebben tussen de haakjes die achter de naam staan. De functie doet iets met die parameters en geeft een resultaat terug. Bijvoorbeeld, de functie **print()** drukt de parameter waardes af op het scherm, en geeft niks terug.

Type casting functies nemen de parameter waarde die tussen de haakjes is gegeven, en geven een waarde terug die (bijna) hetzelfde is als de parameter waarde, maar van een verschillend type. De drie belangrijkste type casting functies zijn:

- **int()** geeft de parameter waarde terug als integer (indien noodzakelijk afgerond naar beneden)
- **float()** geeft de parameter waarde terug als float (waarbij .0 indien noodzakelijk wordt toegevoegd)
- **str()** geeft de parameter waarde terug als string

Bekijk de verschillen tussen de volgende twee regels code:

```
print( 15/4 )
print( int( 15/4 ) )
```

Of de volgende twee regels:

```
print( 15+4 )
print( float( 15+4 ) )
```

Ik had al aangegeven dat je de plus-operator niet kunt gebruiken om een getal aan een string vast te plakken. Als je zoiets toch wilt doen, kun je een oplossing maken met behulp van string type casting:

```
print( "Ik heb " + str( 15 ) + " appels." )
```

3.4 Stijl

Misschien is het je opgevallen dat ik in mijn voorbeelden veel spaties gebruik. Bijvoorbeeld, bij de haakjes die achter een functienaam staan, zet ik altijd een spatie na het openingshaakje en voor het sluithaakje. In berekeningen zet ik vaak spaties rondom operatoren als dat de berekening beter leesbaar maakt. Ik zet ook vaak lege regels in mijn code, en gebruik consequent vier spaties als tabulatie waar nodig.

De meeste van deze zaken zijn gewoon "stijl." De spaties bij de haakjes en rond de operatoren zijn niet nodig. Python begrijpt de code ook prima als ze er niet staan. De volgende vier regels code zijn equivalent:

```
# Equivalente regels code
print( 2 + 3 )
print(2+3)
print( 2+3)
print        (        2        +        3        )
```

Het "vastplakken" van het openingshaakje aan de functienaam doet vrijwel iedere programmeur, maar de rest verschillen de stijlen van spaties plaatsen tussen programmeurs (bijvoorbeeld, mijn stijl waarin ik een spatie plaats voor een sluithaakje is hoogst zeldzaam). Je kunt hierin je eigen stijlvoorkeur gebruiken, je hoeft niet de mijne te volgen. Maar ik raad je wel aan om een consistente stijl te gebruiken, want dat maakt je code leesbaarder, zelfs voor programmeurs die er een andere stijl op nahouden.

Merk op dat de code hierboven een "hash mark" (#; hiervoor bestaat geen Nederlandstalig woord) heeft op de eerste regel, waarna een tekst volgt die wat details over de code uitlegt. Deze regel is een commentaarregel: als je een hash mark gebruikt in je code (behalve als die in een string staat, natuurlijk) dan is alles wat rechts van de hash mark staat op de regel commentaar, dat door Python genegeerd wordt. Je kunt commentaar gebruiken om details over je code te geven, als je denkt dat uitleg nodig is. Meer over het geven van commentaar volgt in een later hoofdstuk.

Wat je geleerd hebt

In dit hoofdstuk is het volgende besproken:

- Het gebruik van de **`print()`** functie om zaken op het scherm te tonen
- Data types string, integer, en float
- Berekeningen
- Basale string expressies
- Type casting tussen strings, integers, en floats, middels **`str()`**, **`int()`**, en **`float()`**

Opgaves

Opgave 3.1 Een boek kost in de winkel €24,95, maar boekwinkels krijgen 40 procent korting bij inkoop. Het verschepen van boeken kost €3 voor het eerste boek, en 75 cent voor ieder volgende boek. Bereken hoeveel de winkel betaalt voor 60 boeken.

Opgave 3.2 Kun je de fouten in de volgende regels code identificeren? Verbeter ze.

exercise0302.py

```
print( "Een boodschap" ).
print( "Een boodschap' )
print( 'Een boodschapf"' )
```

Opgave 3.3 Als er iets fout zit in code, geeft Python meestal een foutmelding. Dit zijn vaak "syntax fouten," die aangeven dat er iets fout zit in de vorm van je code (bijvoorbeeld, voor de code hierboven werden `SyntaxError`s gerapporteerd). Er zijn ook "runtime errors," die aangeven dat je code op zich syntactisch correct lijkt, maar dat er iets fout is gegaan bij de uitvoering ervan. Een goed voorbeeld is de `ZeroDivisionError`, die aangeeft dat je probeerde te delen door nul (wat niet mag, zoals je weet). Schrijf een kort programma dat zo'n fout genereert als je het uitvoert.

Opgave 3.4 Hier is een ander illustratief voorbeeld van een runtime error. Voer het programma uit en bestudeer de fout die gemeld wordt. Kun je het probleem oplossen?

exercise0304.py

```
print( ((2*3)/4 + (5-6/7)*8 )
print( ((12*13)/14 + (15-16)/17)*18 )
```

Opgave 3.5 Je kijkt op de klok en je ziet dat het 14.00u is. Je zet een alarm dat 535 uur later af moet gaan. Hoe laat is het als het alarm afgaat? Schrijf een programma dat het antwoord afdrukt. Hint: Gebruik de modulo operator.