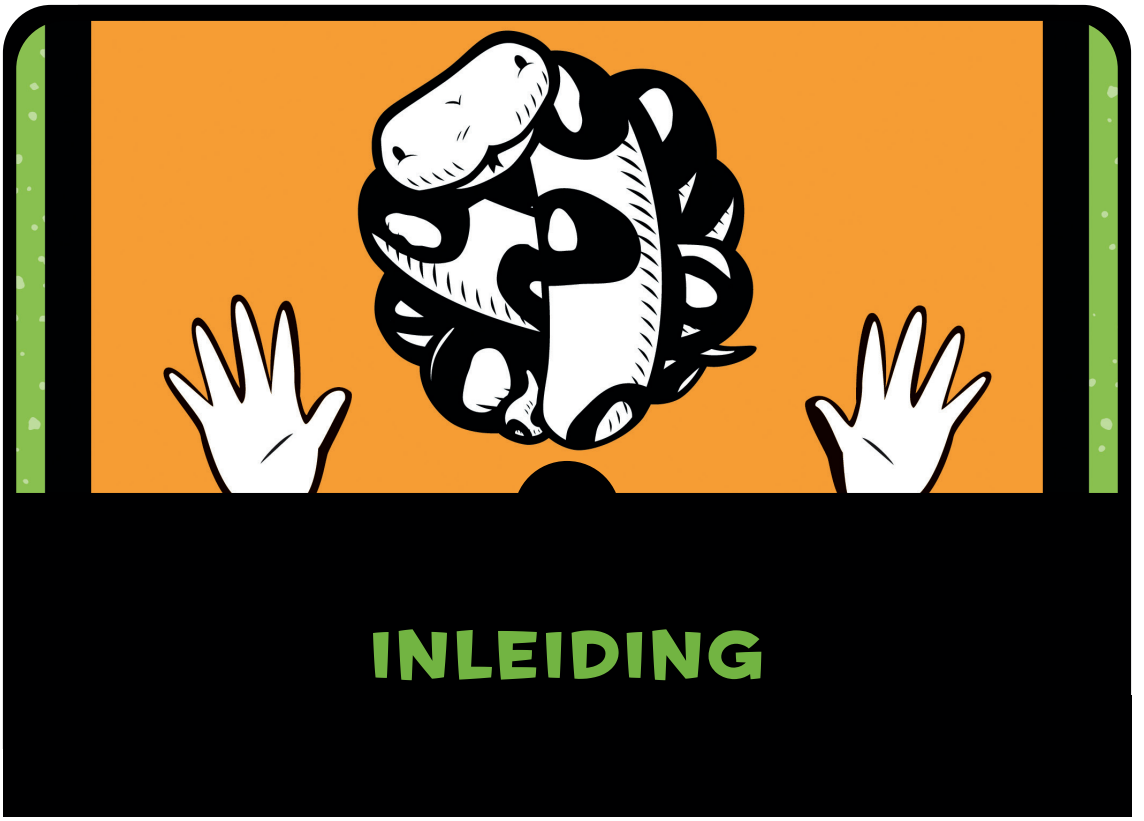


INLEIDING

Waarom zou je leren programmeren? Programmeren stimuleert creativiteit, logisch denken en het probleemoplossend vermogen. De programmeur krijgt de kans om iets uit niets te maken, logica te gebruiken om programmeerprincipes om te werken naar een vorm die voor een computer bruikbaar is en, als alles niet werkt zoals verwacht, het probleem op te lossen. Programmeren is leuk, soms uitdagend (af en toe frustrerend) en de vaardigheden die je erbij leert kunnen nuttig zijn voor je werk op school en in je

toekomstige baan... ook als die niets te maken heeft met computers. Bovendien is programmeren een prima manier om een druilerige middag door te komen.

WAAROM PYTHON?

Python is programmeertaal die gemakkelijk te leren is, met een paar nuttige eigenschappen voor de beginnende programmeur. Vergeleken met andere programmeertalen, is de code vrij gemakkelijk te lezen en Python heeft een interactieve shell waarin je jouw programma's kunt invoeren en uitvoeren. Naast de leesbaarheid en de shell, bevat Python een paar elementen die het leerproces behoorlijk versnellen en je de mogelijkheid bieden om simpele animaties te maken voor je eigen spellen. Eén daarvan is de `turtle` module, geïnspireerd op Turtle graphics (door de programmeertaal Logo gebruikt in de jaren 60 van de twintigste eeuw) en bestemd voor opleidingsdoeleinden. Een ander element is de `tkinter` module, een interface voor de Tk GUI toolkit, die een eenvoudige methode biedt voor het maken van programma's met meer geavanceerde grafische elementen en animaties.

HOE JE CODE LEERT SCHRIJVEN

Net als alle andere dingen die je voor het eerst doet, kan je het beste bij de basis beginnen, dus begin gewoon bij de eerste hoofdstukken en probeer de neiging te onderdrukken om alvast naar de latere hoofdstukken te bladeren. Niemand kan direct een symfonie spelen als ze voor het eerst een muziekinstrument oppakken. Toekomstige piloten mogen pas vliegen als ze de basisbeginselen onder controle hebben. Turners kunnen meestal geen salto's achterover maken als ze dit voor het eerst proberen. Als jij te snel gaat, sla je niet alleen stappen over en heb je de basis niet goed in je hoofd zitten, maar zal je de stof in de latere hoofdstukken lastiger vinden dan die in werkelijkheid is.

Bij het doorwerken van dit boek is het een goed idee om alle voorbeelden te volgen, zodat je ziet hoe ze werken. Aan het einde van de meeste hoofdstukken vind je ook programmeeroefeningen die je kunt maken om je programmeervaardigheid te verbeteren. Onthoud dit: hoe beter je de basisprincipes begrijpt, des te gemakkelijker het is om de meer ingewikkelde ideeën te snappen die je verderop tegenkomt.

Als je iets frustrerend of te moeilijk vindt, heb ik een paar tips:

1. Verdeel het probleem in kleinere stukken. Probeer te begrijpen wat een klein stukje code precies doet of denk alleen aan een klein stukje van een ingewikkeld concept. Richt je op een klein gedeelte van de code in plaats van alles in één keer te proberen te begrijpen.
2. Als dit niet helpt, is het soms het beste om het even te laten rusten. Slaap er eens een nachtje over en pak het de volgende dag weer op. Dit is een goede manier om een hoop problemen op te lossen en het kan zeker nuttig zijn voor computerprogrammeurs.

VOOR WIE IS DIT BOEK

Dit boek is bedoeld voor iedereen die belangstelling heeft voor het programmeren, of dat nu een kind is of een beginnende volwassene. Als je wilt leren om je eigen software te schrijven in plaats van alleen maar programma's te gebruiken die door anderen zijn ontwikkeld, is *Programmeren met Python* een prima startpunt.

In de volgende hoofdstukken vind je informatie over het installeren van Python, het opstarten van de Python shell, het uitvoeren van simpele berekeningen, het printen van tekst naar het scherm en het maken van lijsten. Ook leer je eenvoudige handelingen uitvoeren met gebruik van de `if` statements en `for` lussen en leer je wat `if` statements en `for` lussen eigenlijk zijn. Je leert hoe je code hergebruikt met functies, je leert de basisprincipes van klassen en objecten en krijgt beschrijvingen van een paar van de vele ingebouwde functies en modules in Python.

Er zijn hoofdstukken over zowel de simpele als de geavanceerde grafische functies van `turtle` en ook over het gebruik van de `tkinter` module om tekeningen te maken op het computerscherm. Aan het einde van diverse hoofdstukken vind je programmeeroefeningen van verschillende moeilijkheidsgraden, waarmee je de zojuist opgedane kennis in praktijk kunt brengen door zelf kleine programma's te schrijven.

Wanneer je eenmaal deze basiskennis hebt opgebouwd, leer je hoe je je eigen spellen kunt schrijven. Je gaat twee grafische spellen ontwikkelen en leert hoe je kunt zien of voorwerpen elkaar raken en je leert meer over gebeurtenissen en diverse animatietechnieken.

De meeste voorbeelden in dit boek gebruiken Python's IDLE (Integrated DeveLopment Environment) shell. IDLE zorgt ervoor dat bepaalde stukken code (syntax) worden gemarkeerd en dat je code kunt kopiëren en plakken, net als in andere applicaties. Ook heb je een bewerkingsscherm waar je code kunt opslaan om later te gebruiken, wat betekent dat IDLE zowel een interactieve experimenteeromgeving is als een soort tekstverwerker. De voorbeelden werken net zo goed met de standaard console en een gewone tekstverwerker, maar IDLE is gebruiksvriendelijker en helpt het je beter te begrijpen. We vertellen je direct in het begin hoe je deze shell installeert.

WAT STAAT ER IN DIT BOEK

Hier is een kort overzicht van wat je in elk hoofdstuk kunt verwachten.

Hoofdstuk 1 is een inleiding tot het programmeren, met instructies voor het voor de eerste keer installeren van Python.

Hoofdstuk 2 introduceert de basisberekeningen en variabelen en **Hoofdstuk 3** beschrijft een paar basis datatypes in Python, zoals strings, lijsten en tupels.

Hoofdstuk 4 laat je kennismaken met de `turtle` module. We springen van basis programmeerprincipes naar het laten bewegen van de turtle over het scherm.

Hoofdstuk 5 behandelt de verschillende voorwaarden en `if` statements en **Hoofdstuk 6** gaat verder met `for` en `while` lussen.

Hoofdstuk 7 laat ons een begin maken met het gebruiken en maken van functies en in **Hoofdstuk 8** behandelen we klassen en objecten.
We hebben dan genoeg basisprincipes behandeld om een paar van de speltechnieken te begrijpen die we in latere hoofdstukken nodig hebben als we spellen gaan ontwikkelen. Vanaf dit punt wordt alles een beetje ingewikkelder.

Hoofdstuk 9 beschrijft het grootste deel van de ingebouwde functies in Python en **Hoofdstuk 10** gaat verder met een paar modules (eigenlijk zijn dit veel nuttige functionaliteiten) die standaard in Python zijn geïnstalleerd.

Hoofdstuk 11 keert terug naar de `turtle` module en laat de lezer experimenteren met meer ingewikkelde figuren. **Hoofdstuk 12** behandelt het gebruik van de `tkinter` module om meer geavanceerde afbeeldingen te maken.

In de **Hoofdstukken 13** en **14** maken we ons eerste spel, “Bounce!,” waarvoor je de kennis nodig hebt die je in de voorgaande hoofdstukken hebt opgebouwd, en in de **Hoofdstukken 15–18** maken we nog een spel, “Mr. Stick Man holt naar de uitgang.” De hoofdstukken over spelontwikkeling zijn ook de plaatsen waar het weleens fout kan gaan.
Als het niet lukt, download dan de code van de website bij dit boek *www.visualsteps.nl/programmerenpython* en vergelijk jouw code met de voorbeelden die wel goed werken.

In het **Nawoord** ronden we alles af door een blik te werpen op PyGame en op een paar andere programmeertalen.

Tenslotte lees je in de **Bijlage** meer over de sleutelwoorden in Python en kun je in de **Woordenlijst** de definities vinden van de programmeertermen die in dit boek zijn gebruikt.

2

BEREKENINGEN EN VARIABELEN

Nu je Python hebt geïnstalleerd en weet hoe je de Python shell moet starten, ben je zo ver dat je er iets mee kunt doen. We beginnen met een paar eenvoudige berekeningen en gaan dan verder met variabelen. *Variabelen* worden gebruikt om dingen in een computerprogramma op te slaan en ze helpen je bij het schrijven van nuttige programma's.

REKENEN MET PYTHON

Normaal gesproken gebruik je een rekenmachine of pen en papier wanneer je wordt gevraagd uit te rekenen wat de uitkomst is van 8 × 3.57. Wat dacht je ervan om nu eens de Python shell te gebruiken voor deze berekening? Laten we het maar eens proberen.

Start de Python shell door te dubbelklikken op het IDLE icoon op jouw bureaublad. Achter de prompt voer je de som van de eerste regel in en druk je op ENTER:

```
>>> 8 * 3.57
28.56
```

Wanneer je een vermenigvuldigingsberekening invoert in Python, gebruik je het asterisk symbool (*) in plaats van het vermenigvuldigingsteken (×).

Zullen we nu een wat praktischer voorbeeld nemen?

Stel dat je in je achtertuin graaft en je vindt een zak met 20 gouden munten. Een dag later sluip je de kelder in en stop je de munten in de dupliceermachine die jouw opa ooit heeft uitgevonden. Je hoort wat gezoef en geklik en een paar uur later rollen er nog eens 10 glimmende munten uit de machine.

Hoeveel munten zou jij in je schatkist hebben als je dit een jaar lang elke dag zou doen? Op papier ziet de rekensom er zo uit:

10 × 365 = 3650
20 + 3650 = 3670

Het is natuurlijk niet zo moeilijk om deze sommen met een rekenmachine of op papier uit te rekenen, maar we kunnen alles ook met de Python shell uitrekenen. Eerst vermenigvuldigen we 10 munten met de 365 dagen in een jaar, wat 3650 oplevert. Daarna voegen we de eerste 20 originele munten weer toe en komen dan uit op 3670.

```
>>> 10 * 365
3650
>>> 20 + 3650
3670
```

Wat zou er nu gebeuren als een raaf die blinkende munten ziet liggen in jouw slaapkamer en elke week drie munten steelt? Hoeveel munten zou je aan het eind van het jaar nog over hebben? Hieronder zie je hoe de berekening eruit ziet in de shell:

```
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Eerst vermenigvuldigen we 3 munten met de 52 weken in het jaar. De uitkomst is 156. Dit getal trekken we af van het totale aantal munten dat we hebben (3670) en zo komen we erachter dat we aan het eind van het jaar nog 3514 munten over zouden hebben.

Dit programma is heel erg simpel. In dit boek ga je leren om deze basis te gebruiken voor het schrijven van programma's die nog veel nuttiger zijn.

PYTHON OPERATOREN

In de Python shell kun je vermenigvuldigen, optellen, aftrekken, delen en een aantal andere rekenkundige bewerkingen uitvoeren die we hier nu niet gaan bespreken. De standaardsymbolen die Python voor deze berekeningen gebruikt, heten *operatoren* en je ziet ze in Tabel 2-1.

Tabel 2-1: Standaard Python operatoren

Symbool	Bewerking
+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Delen

De *voorwaartse schuine streep* (/) wordt gebruikt voor het delen, omdat je die ook gebruikt wanneer je een breuk opschrijft. Als je bijvoorbeeld 100 piraten en 20 grote vaten hebt en je wilt uitrekenen hoeveel piraten je in elk vat kunt verstoppen, zou je 100 piraten door 20 vaten kunnen delen (100 ÷ 20) door in de Python shell `100 / 20` in te voeren. Onthoud wel dat de voorwaartse schuine streep naar rechts helt.

DE VOLGORDE VAN BEWERKINGEN

In een programmeertaal gebruiken we haakjes om de volgorde van de bewerkingen te bepalen. Een *bewerking* is alles wat gebruikmaakt van een operator. Vermenigvuldigen en delen gaan boven optellen en aftrekken en dat betekent dat deze bewerkingen als eerste worden uitgevoerd. Met andere woorden, als je een berekening invoert in Python, worden vermenigvuldigingen en delingen het eerst uitgevoerd en dan pas worden er getallen opgeteld of afgetrokken.

In de volgende berekening worden de getallen 30 en 20 bijvoorbeeld eerst vermenigvuldigd en wordt daarna het getal 5 opgeteld bij de uitkomst (het product) van die vermenigvuldiging.

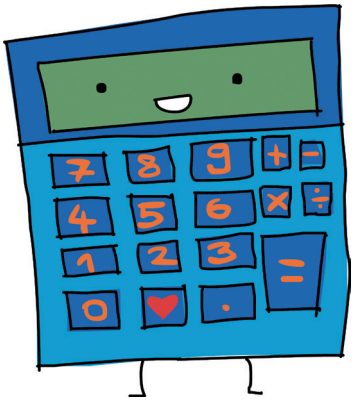
```
>>> 5 + 30 * 20
605
```

Deze vergelijking is als het ware een manier om te zeggen dat je “30 met 20 moet vermenigvuldigen en dan 5 moet optellen bij het resultaat.” Het eindresultaat is 605. We kunnen de volgorde van de bewerkingen veranderen door haakjes te zetten om de eerste twee getallen, zoals hier:

```
>>> (5 + 30) * 20
700
```

Het resultaat van deze vergelijking is 700 en niet 605, omdat de haakjes Python vertellen dat de bewerking tussen die haakjes het eerst moet worden gedaan en dan pas het deel wat buiten de haakjes staat. Dit voorbeeld zegt “tel 5 op bij 30 en vermenigvuldig het resultaat met 20”.

Haakjes kunnen *genest* worden, wat betekent dat je haakjes binnen andere haakjes kunt zetten, zoals in het voorbeeld hieronder:



```
>>> ((5 + 30) * 20) / 10
70.0
```

In dit geval voert Python de berekening tussen de binnenste haakjes het eerste uit, daarna die tussen de buitenste haakjes en gaat dan verder met de deling die buiten de haakjes staat. Deze vergelijking zegt eigenlijk “tel 5 op bij 30, vermenigvuldig het resultaat met 20 en deel dat resultaat dan weer door 10”. Het gaat dus zo:

- 5 plus 30 is 35.
- 35 maal 20 is 700.
- 700 gedeeld door 10 is 70. Dat is het eindresultaat.

Als we geen haakjes hadden gebruikt, zou het resultaat wel anders zijn geweest:

```
>>> 5 + 30 * 20 / 10
65.0
```

In dit geval wordt eerst 30 met 20 vermenigvuldigd en is de uitkomst 600, dan wordt 600 gedeeld door 10 met als uitkomst 60. Tenslotte wordt 5 opgeteld bij 60, met als eindresultaat 65.

LET OP!

Onthoud dat vermenigvuldigen en delen altijd boven optellen en aftrekken gaan, tenzij er haakjes worden gebruikt om een andere volgorde van de bewerkingen te bepalen.

VARIABELEN LIJKEN OP LABELS

Bij het programmeren wordt het woord *variabele* gebruikt om een plek aan te duiden waar je informatie in kunt opslaan, zoals getallen, letters, lijsten, teksten, enzovoorts. Je kunt een variabele ook beschouwen als een label, ofwel een etiket voor iets.

Als wij bijvoorbeeld een variabele maken die `fred` heet, gebruiken we een "is gelijk" teken (`=`) en dan vertellen we Python voor welke informatie we dit label gebruiken. Hier maken we een variabele `fred` en vertellen we Python dat deze variabele een label is voor het getal 100 (een andere variabele mag trouwens ook best een waarde van 100 hebben):

```
>>> fred = 100
```

Om te zien welke waarde een variabele heeft, hoef je alleen maar `print` in te voeren in de shell, gevolgd door de naam van de variabele tussen haakjes, zoals hier:

```
>>> print(fred)
100
```

We kunnen Python ook vertellen om de variabele `fred` te wijzigen, zodat hij een label voor een andere waarde wordt. Op deze manier kun je bijvoorbeeld `fred` veranderen in het getal 200:

```
>>> fred = 200
>>> print(fred)
200
```

Op de eerste regel zeggen we dat `fred` een label is voor het getal 200. Op de tweede regel vragen we wat precies de waarde is van `fred` om te zien of de wijziging klopt. Python print het resultaat op de laatste regel.

We kunnen ook meer dan één label (meer dan één variabele) gebruiken voor hetzelfde item:

```
>>> fred = 200
>>> jan = fred
>>> print(jan)
200
```

In dit voorbeeld vertellen we Python dat we willen dat de naam (of variabele) `jan` een label is voor hetzelfde getal als `fred` door het is gelijk teken te gebruiken tussen `jan` en `fred`.

Uiteraard is `fred` niet zo'n handige naam voor een variabele, want zo'n soort naam vertelt ons niet waar die variabele voor wordt gebruikt. Laten we deze variabele maar eens `aantal_munten` noemen in plaats van `fred`, zie hieronder:

```
>>> aantal_munten = 200
>>> print(aantal_munten)
200
```

Hiermee wordt duidelijk dat we het hebben over 200 munten.

Variabele namen kunnen bestaan uit letters, cijfers en het underscore teken (_), maar ze mogen niet beginnen met een cijfer. Je kunt alles als naam gebruiken voor een variabele, of het nu één enkele letter is (zoals `a`) of een lange zin. Een variabele mag geen spatie bevatten, dus moet je een underscore gebruiken om woorden van elkaar te scheiden. Soms is het beter een korte variabele naam te gebruiken, vooral als je snel iets moet doen. De naam die je kiest is afhankelijk van de betekenis die je aan de variabele toekent.

Je weet nu hoe je variabelen maakt, dus gaan we eens kijken hoe je ze kunt gebruiken.

VARIABELEN GEBRUIKEN

Weet je nog hoe we berekenden hoeveel munten je aan het eind van het jaar zou hebben als je met jouw opa's uitvinding in de kelder nieuwe munten kon maken? We hadden deze vergelijking gemaakt:

```
>>> 20 + 10 * 365
3670
>>> 3 * 52
156
>>> 3670 - 156
3514
```

Deze som kunnen we ook in één enkele coderegel opschrijven:

```
>>> 20 + 10 * 365 - 3 * 52
3514
```

Wat als we van die getallen nou eens variabelen maakten? Probeer dit maar eens in te voeren:

```
>>> gevonden_munten = 20
>>> magische_munten = 10
>>> gestolen_munten = 3
```