Inhoud

0 Inleiding

Inleiding – Wat is Python?

Het is een programmeertaal!

Wat is Python eigenlijk? Het korte antwoord is dat Python een programmeertaal is. Deze taal is in de late jaren tachtig van de vorige eeuw in Nederland ontwikkeld door Guido van Rossum. Guido is nog steeds actief betrokken bij de ontwikkeling en evolutie van de taal, wat hem de eretitel 'Benevolent Dictator for Life' (welwillende dictator voor het leven) of kortweg *BDFL* heeft opgeleverd. Python is een open-source project dat door iedereen gratis kan worden gedownload en naar eigen inzicht kan worden gebruikt. De non-profitorganisatie Python Software Foundation (**https://www.python.org/psf/**) beheert het intellectuele eigendom, speelt een grote rol bij de promotie van de taal en financiert sommige onderdelen van de ontwikkeling.

Technisch gezien is Python een sterk getypeerde taal. Dit betekent dat elk object een vaststaand type heeft, dat meestal niet te omzeilen is. Tegelijkertijd is Python dynamisch getypeerd, waardoor er geen typecontrole op je code wordt uitgevoerd voor de uitvoering ervan. Dit wijkt af van statisch getypeerde talen als C++ en Java, waarbij een compiler veel typecontroles voor je uitvoert en programma's verwerpen die objecten misbruiken. De beste beschrijving van het typesysteem van Python is dat het *duck-typing* gebruikt, waarbij de geschiktheid van een object in een specifieke context op het moment van uitvoering wordt bepaald. Dit onderwerp komt uitgebreid in hoofdstuk 8 aan bod.

Python is een algemene programmeertaal. Het is niet bedoeld voor een specifiek domein of een vaststaande omgeving, maar kan worden ingezet voor een grote verscheidenheid aan taken. Er zijn natuurlijk gebieden waar Python minder geschikt voor is dan andere talen, zoals extreem tijdgevoelige of geheugenbeperkte omgevingen, maar over het algemeen is Python net zo flexibel en aanpasbaar als andere moderne programmeertalen en soms zelfs flexibeler en aanpasbaarder.

Python is een geïnterpreteerde taal. Eigenlijk klopt dit technische gezien niet helemaal, want Python *wordt* wel degelijk voor de uitvoering normaal gecompileerd tot een vorm van bytecode. Maar deze compilatie wordt onzichtbaar uitgevoerd, waardoor Python de code onmiddellijk lijkt uit te voeren zonder waarneembare compileerfase. Dit ontbreken van een extra stap tussen bewerking en uitvoering is een van de grote voordelen van het werken met Python.

De syntaxis van Python is ontworpen om duidelijk, leesbaar en expressief te zijn. In tegenstelling tot veel andere populaire talen maakt Python gebruik van witruimte om codeblokken te markeren en heeft het hierdoor geen overbodige haakjes nodig om een universele indeling af te dwingen. Hierdoor ziet alle Python-code er in hoofdlijnen hetzelfde uit en leer je snel hoe je Python-

programma's leest. Bovendien zorgt de expressieve syntaxis ervoor dat je veel betekenis aan een regel code kunt toevoegen. Dankzij de expressieve, leesbare code is het beheer in Python betrekkelijk eenvoudig.

Er zijn veel verschillende implementaties van de taal Python. De oorspronkelijke (en nog steeds meestgebruikte) implementatie is in C geschreven. Deze versie wordt meestal aangeduid met *CPython*. Als iemand zegt dat hij Python 'uitvoert', kun je ervan uitgaan dat hij het over CPython heeft. Dit is de implementatie die in dit boek wordt gebruikt.

Andere implementaties van Python zijn:

- Jython (**http://www.jython.org/**), geschreven voor Java Virtual Machine
- IronPython (**http://ironpython.net/**), geschreven voor het platform .NET
- PyPy (**http://pypy.org/**), geschreven in de taal RPython (nogal circulair), bedoeld voor het ontwikkelen van dynamische talen als Python

Deze implementaties volgen over het algemeen CPython, dat beschouwd wordt als de 'standaard' van de taal. Veel van wat je in dit boek leest, is op alle implementaties van toepassing.

Versies van de taal Python

Er worden op dit moment twee belangrijke versies van de taal Python gebruikt: Python 2 en Python 3. Deze twee versies verschillen op belangrijke punten van elkaar, waardoor code voor een van deze versies vaak niet bij de andere werkt, tenzij je voorzorgsmaatregelen neemt. Python 2 is ouder en bekender dan Python 3 (Al zijn steeds meer projecten voornamelijk of zelfs alleen maar voor Python 3 bedoeld), maar in Python 3 zijn beperkingen van de oudere versie verbeterd. Python 3 is de toekomst van Python, dus maak er gebruik van.

Er zijn weliswaar grote verschillen tussen Python 2 en 3, maar de beginselen van beide versies zijn identiek. Leer je een versie kennen, dan is je kennis ook op de andere versie toepasbaar. In dit boek gebruiken we Python 3 en wijzen we je op belangrijke verschillen met Python 2 als dat nodig is.

Het is een standaardbibliotheek!

Naast de programmeertaal biedt Python een krachtige en uitgebreide standaardbibliotheek. Een onderdeel van de Python-filosofie is 'batterijen worden meegeleverd', wat inhoudt dat Python geschikt is voor veel complexe, praktische taken zonder noodzaak pakketten van derden te installeren. Dit is niet alleen bijzonder handig, maar betekent ook dat je Python sneller leert kennen door gebruik te maken van interessante voorbeelden: iets wat we in dit boek dan ook zeker proberen te doen!

Een ander voordeel van de aanpak 'batterijen meegeleverd' is dat veel scripts, zelfs uitgebreide exemplaren, rechtstreeks bij elke Python-installatie kunnen worden uitgevoerd. Dit in tegenstelling tot andere programmeertalen, waarbij je vaak een vervelende drempel hebt door het installeren van extra software.

De documentatie van de standaardbibliotheek is van hoge kwaliteit. De API's zijn uitstekend gedocumenteerd en de modules hebben vaak een goede beschrijving met duidelijke uitleg over de manier waarop je ze inzet. De documentatie van de standaardbibliotheek vind je online (**https://docs.python.org/3/library/index.html**), maar kun je ook lokaal installeren als je wilt.

Aangezien de standaardbibliotheek zo'n belangrijk onderdeel van Python is, kom je overal in dit boek onderdelen ervan tegen. Toch is het niet mogelijk meer dan een klein gedeelte ervan te behandelen, dus we raden je aan de rest zelf te verkennen.

Het is een filosofie

Een beschrijving van Python zou onvolledig zijn zonder te melden dat Python voor veel mensen een filosofie voor het schrijven van code voorstelt. Principes van helderheid en leesbaarheid maken deel uit van het schrijven van correcte, oftewel *Pythonische* code. Het is niet altijd duidelijk wat met *Pythonisch* wordt bedoeld en soms is er meer dan één manier om iets correct te schrijven. Maar het feit dat de Python-gemeenschap belang hecht aan zaken als eenvoud, leesbaarheid en duidelijkheid betekent dat Python-code er over het algemeen fraai uitziet.

Veel Python-principes zijn samengevat in de zogenoemde *Zen van Python* (zie **https://www.python.org/dev/peps/pep-0020/**). Deze 'zen' bestaat niet uit strenge regels, maar meer uit richtlijnen en aanbevelingen om tijdens het programmeren rekening mee te houden. Als je tussen verschillende manieren van werken kunt kiezen, geven deze principes je vaak een duwtje in de juiste richting. In dit boek kom je overal onderdelen van de zen van Python tegen.

Je reis begint...

We vinden Python een prachttaal en helpen je graag je weg erin te vinden. Tegen de tijd dat je dit boek uit hebt, ben je in staat omvangrijke Python-programma's te schrijven en nog complexere te lezen. Belangrijker is het dat dan de basis is gelegd voor het uitbouwen van je kennis met geavanceerde onderwerpen; hopelijk hebben we je enthousiasme aangewakkerd om dit ook daadwerkelijk te doen. Het is een avontuur om alles wat Python te bieden heeft te ontdekken, want het is een omvangrijke taal met een enorm uitgebreid ecosysteem aan ingebouwde en er rond omheen gebouwde software.

Welkom bij Python!

Hoofdstuk 1

Aan de slag

In dit hoofdstuk lees je hoe je Python downloadt en installeert met Windows, Ubuntu Linux of macOS. Je gaat eenvoudige Python-code schrijven en maakt kennis met de beginselen van de programmeercultuur van Python, zoals de Zen ven Python, terwijl je de komische oorsprong van de naam van de taal niet uit het oog verliest.

Je leert in dit hoofdstuk:

Hoe en waar je Python downloadt.

Hoe je Python installeert op Windows, macOS en Linux.

Hoe je de shell start en verlaat.

Hoe je code vormgeeft.

Werken met gegevenstypen, operatoren en lussen.

Python 3 ophalen en installeren

Er zijn twee belangrijke versies van de programmeertaal Python: *Python 2*, de algemeen gebruikte versie en *Python 3*, het heden en de toekomst van de taal. Veel Python-code werkt naar behoren zonder conversie tussen de laatste versie van Python 2 (**Python 2.7**) en recente versies van Python 3, zoals **Python 3.**6 (**https://www.python.org/downloads/**). Er zijn echter wel enkele belangrijke verschillen tussen de grote versies, waardoor ze strikt genomen niet compatibel zijn. In dit boek gebruiken we Python 3 en melden we de grote verschillen met Python 2. Aangezien die boek over de basiselementen van Python gaat, is de kans groot dat de inhoud ervan ook van toepassing is op toekomstige versies van Python 3, dus wees niet terughoudend om die uit te proberen als ze beschikbaar zijn.

Voordat je in Python kunt programmeren, moet je een Python-omgeving downloaden. Python is een bijzonder flexibele taal die voor alle grote besturingssystemen beschikbaar is. Je kunt de voorbeelden in dit boek maken op een computer met Windows, macOS X of Linux; de enige paragrafen met platformspecifieke uitleg volgen hierna, voor de installatie van Python 3. Je hoeft hiervan alleen de paragraaf te lezen die voor jou van toepassing is.

Windows

1. Voor een installatie op Windows ga je naar de officiële Python-website (**https://www.python.org**) en klik je op de downloadlink. Kies het installatieprogramma dat bij het platform past, 32- of 64-bits.
2. Download het installatieprogramma en voer het uit.
3. Geef aan of je Python voor jezelf of voor alle gebruikers van de computer wilt installeren.
4. Kies een locatie voor de Python-distributie. De standaardlocatie voor Python 3.5 is `C:\Python35` op het schijfstation `C:`. We raden je aan Python niet in de map `Program Files` te zetten, want deze sinds Windows Vista gebruikte virtuele bestandsopslag om toepassingen van elkaar te scheiden kan problemen opleveren met het snel installeren van Python-pakketten van derden.
5. We raden je aan op de pagina *Customize Python* van het installatieprogramma de standaardinstellingen te gebruiken, die vergen minder dan 40 MB opslagruimte.
6. Het installatieprogramma installeert niet alleen de Python-runtime en de standaardbibliotheek, maar registreert ook verschillende bestandstypen bij de Python-interpreter, zoals `*.py`.
7. Nadat Python is geïnstalleerd, moet je Python toevoegen aan de omgevingsvariabele `PATH` van het systeem. Kies hiertoe in het Configuratie-

scherm eerst de categorie **Systeem en beveiliging** en dan **Systeem**. Een andere manier om dit venster te openen is met ingedrukte Windows-toets op toets Break op je toetsenbord te drukken. Klik aan de linkerkant van het venster op **Geavanceerde systeeminstellingen** om het tabblad **Geavanceerd** van het dialoogvenster **Systeemeigenschappen** in beeld te brengen. Klik op de knop **Omgevingsvariabelen** om het bijbehorende dialoogvenster te openen.

8. Heb je beheerdersprivileges (Administrator), dan kun je de paden `C:\Python35` en `C:\Python35\Scripts` toevoegen aan de met puntkomma's van elkaar gescheiden lijst met items die bij de systeemvariabele `PATH` hoort. Is dit niet het geval, dan maak je een variabele `PATH` (of pas je een bestaande versie aan) voor jouw gebruiker met dezelfde waarden.
9. Open nu een *nieuw* consolevenster, Powershell of cmd voldoet, en controleer of je Python vanaf de opdrachtregel kunt uitvoeren:

```
> python
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Welkom bij Python! De prompt met drie pijlpunten geeft aan dat Python op je invoer wacht.

Je mag de volgende paragrafen over de installatie van Python bij Mac en Linux overslaan.

macOS

1. Voor een installatie op macOS moet je de officiële Python-website op **http://python.org** bezoeken. Klik eerst op de link **Downloads** en dan op de versie van Python die je wilt installeren.
2. Er wordt een DMG-schijfkopie gedownload die je in de stapel Downloads of in de Finder kunt openen.
3. In het Finder-venster vind je het installatiebestand `Python.mpkg`. Klik met de secundaire of rechtermuisknop om het contextmenu van dit bestand te openen en kies de optie **Open**.
4. Bij sommige versies van OS X krijg je de melding dat het bestand van een onbekende ontwikkelaar afkomstig is. Klik in dit dialoogvenster op **Open** om door te gaan met de installatie.
5. Volg de aanwijzingen van het installatieprogramma van Python.
6. Het is niet nodig de installatie te configureren; gebruik gewoon de standaardinstellingen. Zodra de knop **Installeren** zichtbaar wordt, klik je erop. Misschien moet je je wachtwoord invoeren voordat de installatie begint. Nadat de installatie is voltooid, sluit je het installatievenster.

7 Na de installatie van Python 3 open je een terminalvenster en controleer je of je Python 3 op de opdrachtregel kunt starten:

```
$ python
Python 3.5.0 (default, Nov  3 2015, 13:17:02)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Welkom bij Python! De prompt met drie pijlpunten geeft aan dat Python op je invoer wacht.

Linux

1 Gebruik de pakketbeheerder om Python voor een installatie op Linux. We laten zien hoe dit bij een recente versie van Ubuntu in zijn werk gaat, al gaat dit proces op bijna dezelfde manier bij de meeste andere moderne Linux-distributies.
2 Bij Ubuntu start je het 'Ubuntu Softwarecentrum'. Vaak kun je hiertoe op een knop in de launcher klikken. Maar je kunt het ook via het dashboard uitvoeren door te zoeken naar 'Ubuntu Softwarecentrum' en dan op de selectie te klikken.
3 In het softwarecentrum typ je `python 3.5` in het zoekvak rechtsboven en druk je op Return.
4 Een van de resultaten die je te zien krijgt, is "Python (v3.5)" met klein eronder "Python Interpreter (v3.5)". Selecteer dit item en klik op de installatieknop die erbij verschijnt.
5 Misschien moet je je wachtwoord opgeven om de software te kunnen installeren.
6 Er hoort een voortgangsindicator te verschijnen die weer verdwijnt nadat de installatie is voltooid.
7 Open een terminal (met Ctrl-Alt-T) en controleer of je Python 3.5 vanaf de opdrachtregel kunt uitvoeren:

```
$ python3.5
Python 3.5.0+ (default, Oct 11 2015, 09:05:38)
[GCC 5.2.1 20151010] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Welkom bij Python! De prompt met drie pijlpunten geeft aan dat Python op je invoer wacht.

De shell van Python starten

Nadat je Python hebt geïnstalleerd, kun je er direct mee aan de slag gaan. De shell van Python is een handig hulpmiddel om de taal te leren kennen, wat te experimenteren en snelle test uit te voeren tijdens de softwareontwikkeling.

De shell is de opdrachtregelomgeving van Python, een zogeheten *Read-Eval-Print-Loop*. Python leest (Read) de invoer, Evalueert die, Print het resultaat en keer terug (Loop) naar het begin. Naar deze opdrachtregelomgeving wordt ook wel verwezen met het acroniem 'REPL'.

Na te zijn gestart, toont de shell wat informatie over de versie van Python die je gebruikt, gevolgd door een prompt met drie pijlen. Deze prompt laat je weten dat Python op invoer wacht.

Tijdens een interactieve Python-sessie kun je fragmenten van Python-programma's invoeren, waarna je direct resultaat te zien krijgt. Laten we met een paar eenvoudige sommen beginnen:

```
>>> 2 + 2
4
>>> 6 * 7
42
```

Zoals je ziet, leest Python de invoer, evalueert die, print Python het resultaat en keert hij terug om opnieuw te beginnen.

Je kunt variabelen toekennen in de shell:

```
>>> x = 5
```

de inhoud ervan afdrukken door eenvoudigweg de naam ervan te typen:

```
>>> x
5
```

en ernaar verwijzen in expressies:

```
>>> 3 * x
15
```

In de shell gebruik je een van de weinige obscure verkorte notaties van Python, de onderstrepingsvariabele, om naar de laatst afgedrukte waarde te verwijzen:

```
>>> _
15
```

Deze onderstrepingsvariabele mag je ook in een expressie gebruiken:

```
>>> _ * 2
30
```

Alleen in de shell

Onthoud dat dit nuttige trucje alleen in de shell werkt; de onderstreping heeft geen speciale betekenis in Python-scripts of -programma's.

Je ziet dat niet alle opdrachten een retourwaarde geven. Bij het toekennen van de waarde 5 aan x was er geen retourwaarde; je maakt er alleen de variabele x mee. Andere opdrachten hebben duidelijkere bijwerkingen.

Probeer het volgende eens:

```
>>> print('Hello, Python')
Hello, Python
```

Je ziet dat Python deze opdracht onmiddellijk evalueert en uitvoert, de string 'Hello, Python' afdrukt en dan terugkeert met een prompt. Het is belangrijk dat je beseft dat deze reactie niet het gevolg is van de expressie die de REPL, oftewel de shell evalueert en weergeeft, maar een bijwerking is van de functie `print()`.

Print()

De functie print is een van de grootste verschillen tussen Python 2 en Python 3. In Python 3 zijn de haakjes verplicht, terwijl ze het in Python 2 niet waren. Dit komt doordat `print()` in Python 3 een functieaanroep is. Verderop lees je meer over functies.

De shell verlaten

Het is ook belangrijk dat je weet hoe je de shell afsluit om naar de systeemprompt terug te keren. Dit doe je door de stuurcode *einde bestand* (end-of-file) aan Python te sturen, al gaat dit helaas op verschillende manieren bij verschillende platforms.

Windows

Bij Windows gebruik je de toetsencombinatie Ctrl-Z om de shell af te sluiten.

Mac/Unix

Bij een Mac of Linux gebruik je de toetsencombinatie Ctrl-D om de shell af te sluiten.

Maak je regelmatig van verschillende platforms gebruik en gebruik je per ongeluk Ctrl-Z op een Unix-achtige computer, dan pauzeer je de Python-interpreter en keer je terug naar de shell van het besturingssysteem. Je activeert Python dan opnieuw door er simpelweg met de opdracht `fg` een voorgrondproces van te maken:

```
{lang="text"}
$ fg
```

Druk daarna een paar keer op `Enter` om de prompt met drie pijlen van Python terug te halen.

Codestructuur en significante inspringing

Start de Python 3-interpreter:

```
{language=doscon}
> python
```

bij Windows of:

```
{language=shell}
$ python3
```

op een Mac of bij Linux.

De structuren voor het programmaverloop van Python, zoals for-lussen, while-lussen en if-opdrachten, worden allemaal ingeleid met opdrachten die met een dubbele punt eindigen om aan te geven dat wat volgt de inhoud van de opdracht is. Een for-lus heeft bijvoorbeeld inhoud nodig, dus als je het volgende invoert:

```
>>> for i in range(5):
...
```

toont Python een prompt met drie punten om naar die inhoud te vragen.

Een opvallend (en soms controversieel) aspect van Python is dat beginspaties syntactisch van belang zijn. Dit houdt in dat Python verschillende niveaus inspringing gebruikt om codeblokken te markeren in plaats van de accolades van andere programmeertalen. Moderne Python-code wordt volgens de conventie met vier spaties per niveau ingesprongen.

Dus nadat Python de prompt met drie stippen toont, voeren we vier spaties in en een opdracht die de inhoud van de lus vormt:

```
...     x = i * 10
```

Bij de inhoud van de lus komt nog een tweede opdracht, dus nadat we op Return hebben gedrukt, voeren we weer vier spaties in, gevolgd met een aanroep naar de ingebouwde functie `print()`:

```
...     print(x)
```

Om het codeblok te voltooien, voer je een lege regel in de shell in:

```
...
```

Nadat het codeblok is voltooid, voert Python de ingevoerde code uit en worden de veelvouden van 10 getoond die kleiner zijn dan 50:

```
0
10
20
30
40
```

Kijk je naar een scherm dat vol met Python-code staat, dan zie je dat de inspringing duidelijk overeenkomt (en ook *moet* overeenkomen) met de structuur van het programma; dat zie je goed in de afbeeldingen op de pagina hiernaast. De programmastructuur is ook duidelijk zichtbaar als de code wordt vervaagd.

Elke opdracht die met een dubbele punt wordt afgesloten, leidt tot een nieuwe regel en een extra niveau inspringing. Dit gaat net zo lang door tot een uitspringing naar een eerder niveau terugkeert. Elk niveau inspringing heeft standaard vier spaties; verderop lees je meer over de regels.

```
'"""Class model for aircraft flights."""


class Flight:
    """A flight with a particular aircraft."""

    def __init__(self, number, aircraft):
        if not number[:2].isalpha():
            raise ValueError("No airline code in '{}'".format(number))

        if not number[:2].isupper():
            raise ValueError("Invalid airline code '{}'".format(number))

        if not (number[2:].isdigit() and int(number[2:]) <= 9999):
            raise ValueError("Invalid route number '{}'".format(number))

        self._number = number
        self._aircraft = aircraft

        rows, seats = self._aircraft.seating_plan()
        self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]


    def _passenger_seats(self):
        """An iterable series of passenger seating allocations."""
        row_numbers, seat_letters = self._aircraft.seating_plan()
        for row in row_numbers:
            for letter in seat_letters:
                passenger = self._seating[row][letter]
                if passenger is not None:
                    yield (passenger, "{}{}".format(row, letter))
```

Afbeelding 1.1 *Python-broncode.*

Afbeelding 1.2 *Vervaagde code.*

Pythons aanpak van significante witruimte heeft drie grote voordelen:

1. Ontwikkelaars worden gedwongen één niveau inspringing in een codeblok te gebruiken. Dit wordt over het algemeen in alle talen als een verstandige aanpak beschouwd omdat de code er veel leesbaarder door wordt.
2. Code met significante witruimte hoeft niet volgepropt te worden met overbodige accolades en er ontstaan nooit discussies over de locatie van de accolades. Alle codeblokken zijn in Python duidelijk herkenbaar en worden door iedereen op dezelfde manier geschreven.
3. Significante witruimte vereist een consistente interpretatie van de codestructuur door de auteur, het runtimesysteem van Python en toekomstige programmeurs die de code moeten lezen. Hierdoor is het in een programma niet mogelijk een codeblok volgens Pythons gezichtspunt te hebben dat niet op een blok uit menselijk perspectief lijkt.

De regels voor de inspringing bij Python lijken misschien gecompliceerd, maar zijn in de praktijk erg eenduidig.

- De gebruikte witruimte mag uit spaties en tabs bestaan. Algemeen geldt dat *spaties de voorkeur hebben boven tabs* en dat *vier spaties de standaard is in de Python-gemeenschap*.
- Een belangrijke regel is dat je *nooit* spaties en tabs mag combineren. De Python-interpreter accepteert dit niet en je collega's zullen je met de nek aankijken.
- Het is toegestaan verschillende hoeveelheden inspringing op verschillende momenten te gebruiken. De belangrijkste regel is dat *opeenvolgende regels code op hetzelfde inspringingsniveau als onderdeel van hetzelfde codeblok worden beschouwd*.
- Er zijn enkele uitzonderingen op deze regels, maar die hebben bijna altijd te maken met het verbeteren van de leesbaarheid, bijvoorbeeld door onvermijdelijke lange opdrachten over meerdere regels te verdelen.

Deze rigoureuze aanpak bij het opmaken van code is 'Programmeren zoals Guido het bedoelde'. Een filosofie waarbij veel waarde wordt gehecht aan codekwaliteiten, zoals leesbaarheid, raakt de kern van de Python-cultuur, iets waar we het nu over gaan hebben.