Inhoud

Hoofdstuk 1

Uw eerste webscraper

Zodra u begint met webscrapen, gaat u al die kleine dingen waarderen die browsers voor u doen. Het web ziet er zonder een laag HTML-layout, CSS-styling, JavaScript-uitvoering en beeldrendering in eerste instantie een beetje intimiderend uit, maar in de komende hoofdstukken gaan we bekijken hoe u gegevens kunt formatteren en interpreteren zonder de hulp van een browser.

Dit hoofdstuk begint met de basisprincipes van het verzenden van een GET-request naar een webserver (een verzoek om de inhoud van een webpagina op te halen) voor een specifieke pagina, het lezen van de HTML-uitvoer van die pagina en het uitvoeren van een simpele gegevensextractie om de gewenste inhoud uit te filteren.

Verbinding maken

Als u niet veel tijd hebt besteed aan netwerken of netwerkbeveiliging, kan de machinerie van internet een beetje mysterieus lijken. U wilt niet hoeven nadenken over wat het netwerk precies doet wanneer u een browser opent en naar **google.com** gaat. Tegenwoordig hoeft u dat ook niet. Sterker nog, ik wil zelfs beweren dat het fantastisch is dat computerinterfaces zo ver gevorderd zijn dat de meeste mensen die internet gebruiken geen flauw benul hebben van hoe het werkt.

Webscrapen vereist echter dat een deel van de sluier van deze interface wordt weggerukt – niet alleen op browserniveau (hoe het al deze HTML, CSS en JavaScript interpreteert), maar soms ook op het niveau van de netwerkverbinding.

Om u een idee te geven van de infrastructuur die nodig is om informatie naar uw browser te krijgen, gebruiken we het volgende voorbeeld. Alice bezit een webserver. Bob gebruikt een desktopcomputer die verbinding probeert te maken met de server van Alice. Wanneer een machine met een andere machine wil communiceren, vindt er zoiets als de volgende uitwisseling plaats:

1. Bobs computer stuurt een stroom van 1 en 0 bits, in de vorm van een hoge en lage spanning op een draad. Deze bits vormen een beetje informatie, die een 'header' en een 'body' bevat. De header bevat een directe bestemming van het MAC-adres van zijn lokale router, met als eindbestemming het IP-adres van Alice. De body bevat zijn request voor de serverapplicatie van Alice.
2. De lokale router van Bob ontvangt al deze 1's en 0's en interpreteert ze als een pakket, vanaf het eigen MAC-adres van Bob, bestemd voor het IP-adres van Alice. Zijn router stempelt zijn eigen IP-adres op het pakket als het 'from' IP-adres en verzendt het via internet.
3. Het pakket van Bob doorkruist verschillende tussenliggende servers, die zijn pakket naar het juiste fysieke/bekabelde pad leiden, op weg naar de server van Alice.
4. De server van Alice ontvangt het pakket op haar IP-adres.
5. De server van Alice leest de pakketpoortbestemming in de header en geeft deze door aan de juiste toepassing, de webserverapplicatie. (De pakketpoortbestemming is bijna altijd poort 80 voor webtoepassingen, dit kan worden beschouwd als een appartementnummer voor pakketgegevens, terwijl het IP-adres vergelijkbaar is met het straatadres.)
6. De webserverapp ontvangt een gegevensstroom van de serverprocessor. Deze gegevens zeggen zoiets als het volgende:
   - Dit is een `GET`-request.
   - Het volgende bestand wordt opgevraagd: *index.html*.

7 De webserver lokaliseert het juiste HTML-bestand, bundelt het in een nieuw pakket om naar Bob te verzenden en stuurt het door naar zijn lokale router, voor transport terug naar Bobs machine, via hetzelfde proces.

Et voilà! Dat is nou internet.

Waar in deze uitwisseling kwam de webbrowser in beeld? Helemaal nergens. Browsers zijn in feite een relatief recente uitvinding in de geschiedenis van internet, als we bedenken dat Nexus in 1990 werd uitgebracht.

De webbrowser is een nuttige toepassing voor het aanmaken van deze informatiepakketten, waarbij hij uw besturingssysteem opdraagt ze te verzenden en de gegevens die u terugkrijgt interpreteert als mooie foto's, geluiden, video en tekst. Een webbrowser is echter alleen maar code en code kan uit elkaar worden gehaald, opgedeeld in basiscomponenten, herschreven, opnieuw gebruikt en opgemaakt om alles te doen wat u maar wilt. Een webbrowser vertelt de processor om gegevens naar de toepassing te sturen die uw draadloze (of bekabelde) interface afhandelt, maar u kunt hetzelfde in Python doen met slechts drie regels code:

```
from urllib.request import urlopen
html = urlopen('http://pythonscraping.com/pages/page1.html')
print(html.read())
```

Om dit uit te voeren, kunt u gebruikmaken van het iPython-notebook voor hoofdstuk 1 (**github.com/REMitchell/python-scraping/blob/master/Chapter01_BeginningToScrape.ipynb**) in de GitHub-repository, of u kunt het bestand lokaal opslaan als scrapetest.py en uitvoeren in je terminal met behulp van deze opdracht:

```
$ python scrapetest.py
```

Bedenk dat als u ook Python 2.x op uw machine hebt geïnstalleerd en u beide versies van Python naast elkaar uitvoert, u Python 3.x mogelijk expliciet moet aanroepen door de opdracht op de volgende manier uit te voeren:

```
$ python3 scrapetest.py
```

Met deze opdracht wordt de volledige HTML-code van page1 op de URL **pythonscraping.com/pages/page1.html** weergegeven. Nauwkeuriger gezegd levert dit het HTML-bestand page1.html op, gevonden in de map <web root>/pages, op de server die zich bevindt op de domeinnaam pythonscraping.com.

Waarom is het belangrijk om deze adressen te beschouwen als 'bestanden' en niet als 'pagina's'? De meeste moderne webpagina's zijn gekoppeld aan tal van resourcebestanden. Dit kunnen afbeeldings-, JavaScript-, CSS- of andere bestanden zijn waaraan de gevraagde pagina is gekoppeld. Wanneer een webbrowser op een tag stuit zoals `<img src="cuteKitten.jpg">`, weet de browser dat het een nieuwe request moet doen aan de server voor de gegevens in het bestand *cuteKitten.jpg* om de pagina volledig weer te kunnen geven.

Natuurlijk heeft uw Python-script (nog) niet de logica om terug te gaan om nog meer bestanden op te vragen; het kan alleen het enkele HTML-bestand lezen dat u direct hebt opgevraagd.

```
from urllib.request import urlopen
```

doet precies wat het zegt: het kijkt in het Python-module request (te vinden in de *urllib*-bibliotheek) en importeert daarvan alleen de functie `urlopen`.

urllib is een standaard Python-bibliotheek (wat betekent dat u niets extra hoeft te installeren om dit voorbeeld uit te voeren). Hij bevat functies voor het opvragen van gegevens op internet, het verwerken van cookies en zelfs het wijzigen van metadata zoals headers en uw user agent. We zullen *urllib* in het hele boek intensief gebruiken, dus ik raad u aan de Python-documentatie voor die bibliotheek door te lezen (**docs.python.org/3/library/urllib.html**).

Met `urlopen` kunt u een remote object over een netwerk openen en lezen. Omdat het een tamelijk generieke functie is (het kan met gemak HTML- en afbeeldingsbestanden of een andere bestandsstream lezen), zullen we deze vrij vaak door het hele boek toepassen.

Een inleiding tot BeautifulSoup

Beautiful Soup, so rich and green,
Waiting in a hot tureen!
Who for such dainties would not stoop?
Soup of the evening, beautiful Soup!

De *BeautifulSoup*-bibliotheek is vernoemd naar een gedicht van Lewis Carroll met dezelfde naam in *Alice's Adventures in Wonderland*. In het verhaal wordt dit gedicht gezongen door een personage dat de Mock Turtle wordt genoemd (zelf een woordspeling op een populair Victoriaanse gerecht, genaamd Mock Turtle Soup, dat niet bereid is met schildpad maar met rundvlees).

Net als haar naamgenoot in Wonderland, probeert *BeautifulSoup* wijs te worden uit het onwijze; het helpt bij het formatteren en organiseren van het rommelige web door slechte HTML te corrigeren en ons te presenteren met gemakkelijk te doorlopen Python-objecten met een XML-structuur.

BeautifulSoup installeren

Omdat de BeautifulSoup-bibliotheek geen standaard Python-bibliotheek is, moet deze apart worden geïnstalleerd. Als u al ervaring hebt met het installeren van Python-bibliotheken, gebruik dan uw favoriete installatieprogramma en ga verder met de volgende paragraaf, *BeautifulSoup uitvoeren*.

Voor degenen die nog nooit Python-bibliotheken hebben geïnstalleerd (of een opfrisser nodig hebben), wordt deze algemene methode gebruikt voor het installeren van meerdere bibliotheken in het boek, dus misschien wilt u deze paragraaf ook voor het verdere verloop raadplegen.

We gaan in dit boek de BeautifulSoup 4-bibliotheek (ook bekend als BS4) gebruiken. De volledige installatie-instructies van BeautifulSoup 4 zijn te vinden op Crummy.com. Hierna volgt de basismethode voor Linux:

```
$ sudo apt-get install python-bs4
```

En voor de Mac:

```
$ sudo easy_install pip
```

Hiermee installeert u de Python-pakketbeheerder pip. Voer vervolgens het volgende uit om de bibliotheek te installeren:

```
$ pip install beautifulsoup4
```

Bedenk nogmaals dat, als u zowel Python 2.x als 3.x op uw computer hebt geïnstalleerd, u Python3 expliciet moet aanroepen:

```
$ python3 myScript.py
```

Zorg ervoor dat u dit ook gebruikt bij het installeren van pakketten, anders worden de pakketten geïnstalleerd onder Python 2.x, en niet onder Python 3.x:

```
$ sudo python3 setup.py install
```

Als u pip gebruikt, kunt u ook pip3 aanroepen om de Python 3.x-versies van pakketten te installeren:

```
$ pip3 install beautifulsoup4
```

Het installeren van pakketten in Windows is vrijwel identiek aan het proces voor Mac en Linux. Download de meest recente versie van BeautifulSoup 4 van de downloadpagina (**www.crummy.com/software/BeautifulSoup/#Download**), navigeer naar de map waarin u deze hebt uitgepakt en voer dit uit:

```
> python setup.py install
```

En klaar is Kees! BeautifulSoup wordt nu op uw computer herkend als een Python-bibliotheek. U kunt dit uitproberen door een Python-terminal te openen en BeautifulSoup te importeren:

```
$ python
> from bs4 import BeautifulSoup
```

De import moet zonder fouten worden voltooid.

Daarnaast is er een *.exe*-installatieprogramma voor pip in Windows (**pypi.python.org/pypi/setuptools**), zodat u pakketten gemakkelijk kunt installeren en beheren:

```
> pip install beautifulsoup4
```

Bibliotheken integer houden met virtuele omgevingen

Als u van plan bent om aan meerdere Python-projecten te werken, of projecten gemakkelijk wilt kunnen bundelen met alle bijbehorende bibliotheken, of als u zich zorgen maakt over mogelijke conflicten tussen geïnstalleerde bibliotheken, kunt u het beste een virtuele Python-omgeving installeren om alles gescheiden en gemakkelijk beheersbaar te houden.

Wanneer u een Python-bibliotheek zonder een virtuele omgeving installeert, installeert u deze globaal. Meestal vereist dit dat u beheerder of *root* bent en dat de Python-bibliotheek er is voor elke gebruiker en elk project op de machine. Gelukkig is een virtuele omgeving eenvoudig te maken:

```
$ virtualenv scrapingEnv
```

Hiermee creëert u een nieuwe omgeving genaamd scrapingEnv, die u voor gebruik moet activeren:

```
$ cd scrapingEnv/
$ source bin/activate
```

Nadat u de omgeving hebt geactiveerd, ziet u de naam van die omgeving in uw opdrachtregelprompt wat u zegt dat u daar momenteel in werkt. Alle geïnstalleerde bibliotheken of uitgevoerde scripts bevinden zich alleen in die virtuele omgeving.

Als u in de nieuw gecreëerde `scrapingEnv`-omgeving werkt, kunt u daarin BeautifulSoup installeren en gebruiken:

```
(scrapingEnv)ryan$ pip install beautifulsoup4
(scrapingEnv)ryan$ python
> from bs4 import BeautifulSoup
>
```

U kunt de omgeving verlaten met de opdracht `deactivate`, waarna u geen toegang meer hebt tot bibliotheken die in de virtuele omgeving zijn geïnstalleerd:

```
(scrapingEnv)ryan$ deactivate
ryan$ python
> from bs4 import BeautifulSoup
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'bs4'
```

Door alle bibliotheken per project gescheiden te houden, wordt het ook eenvoudig om de volledige omgevingsfolder te zippen en naar iemand anders te sturen. Zolang ze dezelfde versie van Python op hun computer hebben geïnstalleerd, werkt uw code vanuit de virtuele omgeving zonder dat ze zelf bibliotheken hoeven te installeren.

Ik ga u niet expliciet instrueren om voor alle voorbeelden van dit boek een virtuele omgeving te gebruiken, maar u kunt dat natuurlijk altijd doen door deze eenvoudig vooraf te activeren.

BeautifulSoup uitvoeren

Het meest gebruikte object in de BeautifulSoup-bibliotheek is, nogal wiedes, het `BeautifulSoup`-object. Laten we eens zien hoe het werkt en het voorbeeld aan het begin van dit hoofdstuk daarvoor aanpassen:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page1.html')
bs = BeautifulSoup(html.read(), 'html.parser')
print(bs.h1)
```

De uitvoer is:

```
<h1>An Interesting Title</h1>
```

Merk op dat alleen de eerste instantie van de h1-tag die op de pagina wordt gevonden, wordt geretourneerd. Volgens afspraak mag er slechts één h1-tag op één enkele pagina voorkomen, maar afspraken worden op internet vaak geschonden. U moet er zich dus van bewust zijn dat alleen de eerste instantie van de tag wordt opgehaald, en niet per se die waarnaar u op zoek bent.

Net als in de vorige webscrape voorbeelden, importeert u de `urlopen`-functie en roept u `html.read()` aan om de HTML-inhoud van de pagina op te halen. Naast de tekststring kan BeautifulSoup ook het bestandsobject gebruiken dat direct door `urlopen` wordt geretourneerd, zonder eerst `.read()` aan te hoeven roepen:

```
bs = BeautifulSoup(html, 'html.parser')
```

Deze HTML-inhoud wordt vervolgens omgezet in een `BeautifulSoup`-object met de volgende structuur:

- **html** → <html><head>...</head><body>...</body></html>
  - **head** → <head><title>A Useful Page<title></head>
    - **title** → <title>A Useful Page</title>
  - **body** → <body><h1>An Int...</h1> Lorem ip... </body>
    - **h1** → <h1>An Interesting Title</h1>
    - **div** → Lorem Ipsum dolor...

Merk op dat de `h1`-tag die u uit de pagina haalt, twee lagen diep genest is in uw `BeautifulSoup`-objectstructuur (`html` → `body` → `h1`). Wanneer u het echter direct van het object ophaalt, roept u de `h1`-tag rechtstreeks aan:

```
bs.h1
```

In feite produceert een van de volgende functie-aanroepen dezelfde uitvoer:

```
bs.html.body.h1
bs.body.h1
bs.html.h1
```

Wanneer u een `BeautifulSoup`-object maakt, worden twee argumenten doorgegeven:

```
bs = BeautifulSoup(html.read(), 'html.parser')
```

Het eerste is de HTML-tekst waarop het object is gebaseerd en het tweede specificeert de parser die BeautifulSoup moet gebruiken om dat object aan te maken. In de meeste gevallen maakt het geen verschil welke parser u kiest.

`html.parser` is een parser die wordt meegeleverd met Python 3 en geen extra installatie vereist. Behalve waar anders vereist is, zullen we deze parser in het hele boek gebruiken.

Een andere populaire parser is lxml. Deze kan via pip worden geïnstalleerd:

```
$ pip3 install lxml
```

lxml kan met BeautifulSoup worden gebruikt door de parser-string te wijzigen:

```
bs = BeautifulSoup(html.read(), 'lxml')
```

lxml heeft een aantal voordelen ten opzichte van html.parser omdat het over het algemeen beter is in het parseren van "rommelige" of verkeerd geformatteerde HTML-code. Het is vergevingsgezind en lost problemen op zoals niet-gesloten tags, onjuist geneste tags en ontbrekende head- of body-tags. Het is ook iets sneller dan html.parser, hoewel snelheid niet per se een voordeel is in webscraping, aangezien de snelheid van het netwerk zelf bijna altijd de grootste bottleneck is.

Een van de nadelen van lxml is dat het afzonderlijk moet worden geïnstalleerd en voor zijn functioneren afhankelijk is van externe C-bibliotheken. Dit kan, in tegenstelling tot html.parser, problemen veroorzaken met betrekking tot overdraagbaarheid en gebruiksgemak.

Een andere populaire HTML-parser is `html5lib`. Net als lxml is html5lib een extreem vergevingsgezinde parser die nog meer initiatief neemt bij de correctie van gebroken HTML. `html5lib` is ook afhankelijk van externe bronnen en is langzamer dan zowel lxml als html.parser. Desondanks kan het een goede keuze zijn als u met rommelige of handgeschreven HTML-sites werkt.

U past het na installatie toe door de string `html5lib` mee te geven aan het BeautifulSoup-object:

```
bs = BeautifulSoup(html.read(), 'html5lib')
```

Ik hoop dat deze kleine proeve van *BeautifulSoup* u een idee heeft gegeven van de kracht en eenvoud van deze bibliotheek. Vrijwel alle informatie kan worden onttrokken uit elk HTML- of XML-bestand, zolang het een identificerende tag bezit. In Hoofdstuk 2 gaan we dieper in op meer complexe BeautifulSoup-functieaanroepen en behandelen we reguliere expressies en hoe u deze samen met `BeautifulSoup` kunt toepassen om informatie uit websites te onttrekken.

Betrouwbaar verbinden en excepties afhandelen

Het web is een rommeltje. Gegevens worden slecht geformatteerd, websites gaan plat en afsluitende tags gaan verloren. Een van de meest frustrerende ervaringen met webscrapen is te gaan slapen met een scraper in uitvoering, dromend van alle gegevens die je de volgende dag in je database zult vinden, om vervolgens te ontdekken dat de scraper een fout maakte bij een onverwacht gegevensformat en de uitvoering staakte kort nadat je voor de laatste keer naar het scherm gekeken had. In situaties als deze zou je in de verleiding kunnen komen om de ontwikkelaar die de website heeft gemaakt (en de vreemd geformatteerde gegevens) te verwensen, maar de persoon die je eigenlijk op zijn falie zou moeten geven, ben je zelf, omdat je niet anticipeerde op mogelijke problemen of excepties!

Laten we eens naar de eerste regel in onze scraper kijken, na de importinstructies, en uitzoeken hoe we om kunnen gaan met excepties die deze regel kan opleveren:

```
html = urlopen('http://www.pythonscraping.com/pages/page1.html')
```

In deze regel kunnen twee dingen mis gaan:

- De pagina wordt niet gevonden op de server (of er is een fout opgetreden bij het ophalen ervan).
- De server wordt niet gevonden.

In de eerste situatie wordt een HTTP-fout geretourneerd. Deze HTTP-fout kan "404 pagina niet gevonden", "500 interne serverfout" enzovoort zijn. In al deze gevallen genereert de `urlopen`-functie de generieke exceptie `HTTPError`. U kunt deze exceptie op de volgende manier afhandelen:

```
from urllib.request import urlopen
from urllib.error import HTTPError

try:
    html = urlopen('http://www.pythonscraping.com/pages/page1.html')
except HTTPError as e:
    print(e)
    # return null, break, of doe iets anders: Plan B
else:
    # programma gaat verder. Opmerking: in geval van return of break in de
    # exceptie-afhandeling, hoeft u het else-statement niet te gebruiken
```

Als een HTTP-foutcode wordt geretourneerd, genereert het programma de fout en voert het de rest onder de `else`-instructie niet uit.

Als de server helemaal niet wordt gevonden (als bijvoorbeeld **www.python-scraping.com** offline is of als de URL verkeerd is getypt), zal `urlopen` een `URLError` genereren. Dit geeft aan dat de server helemaal niet kon worden bereikt, en omdat de externe server verantwoordelijk is voor het retourneren van HTTP-statuscodes, kan er geen `HTTPError` worden gegenereerd en moet de ernstigere `URLError` worden afgevangen. U kunt een controle toevoegen om te zien of dit het geval is:

```
from urllib.request import urlopen
from urllib.error import HTTPError
from urllib.error import URLError

try:
    html = urlopen('https://pythonscrapingthisurldoesnotexist.com')
except HTTPError as e:
    print(e)
except URLError as e:
    print('The server could not be found!')
else:
    print('It Worked!')
```

Als de pagina met succes van de server wordt opgehaald, bestaat er natuurlijk nog steeds het probleem dat de inhoud op de pagina niet helemaal is wat u ervan verwachtte. Telkens wanneer u toegang wilt tot een tag in een `BeautifulSoup`-object, is het slim om te controleren of de tag werkelijk bestaat. Als u toegang probeert te krijgen tot een tag die niet bestaat, retourneert BeautifulSoup het object `None`. Het probleem is dat een poging om toegang tot een tag op een `None`-object zelf te krijgen, resulteert in een `AttributeError`.

De volgende regel (waarbij `nonExistentTag` een verzonnen tag is, niet de naam van een echte BeautifulSoup-functie)

```
print(bs.nonExistentTag)
```

retourneert een `None`-object. U kunt dit object gewoon afhandelen en erop controleren. Het probleem ontstaat als u er niet op controleert, maar in plaats daarvan verder gaat en probeert een andere functie op het `None`-object aan te roepen, zoals geïllustreerd in het volgende:

```
print(bs.nonExistentTag.someTag)
```

Dit levert een exceptie op:

```
AttributeError: 'NoneType' object has no attribute 'someTag'
```

Hoe kun u zich wapenen tegen deze twee situaties? De eenvoudigste manier is om expliciet op beide situaties te controleren:

```
try:
    badContent = bs.nonExistingTag.anotherTag
except AttributeError as e:
    print('Tag was not found')
else:
    if badContent == None:
        print ('Tag was not found')
    else:
        print(badContent)
```

Dit controleren op en afhandelen van elke fout lijkt in eerste instantie arbeidsintensief, maar het is gemakkelijk de code een beetje te reorganiseren om het schrijven daarvan minder moeilijk te maken (en, nog belangrijker, veel minder moeilijk om te lezen). Deze code is dezelfde scraper die op een enigszins andere manier is geschreven:

```
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup

def getTitle(url):
    try:
        html = urlopen(url)
    except HTTPError as e:
        return None
    try:
        bs = BeautifulSoup(html.read(), 'html.parser')
        title = bs.body.h1
    except AttributeError as e:
        return None
    return title

title = getTitle('http://www.pythonscraping.com/pages/page1.html')
if title == None:
    print('Title could not be found')
else:
    print(title)
```

In dit voorbeeld maakt u een functie `getTitle`, die de titel van de pagina retourneert of een `None`-object als er een probleem is met het ophalen ervan. Binnen `getTitle` controleert u op een `HTTPError`, zoals in het vorige voor-

beeld, en kapselt twee van de BeautifulSoup-regels in één `try`-instructie in. Er kan een `AttributeError` door een van deze regels worden gegenereerd (als de server niet bestaat, zou `html` een `None` object zijn en zou `html.read()` een `AttributeError` genereren). Je zou in feite net zo veel regels kunnen insluiten als je wilt binnen één `try`-statement, of een totaal andere functie aanroepen, die op elk moment een `AttributeError` kan genereren.

Bij het schrijven van scrapers is het belangrijk na te denken over het algehele patroon van uw code om excepties af te handelen en de code tegelijkertijd leesbaar te maken. U wilt waarschijnlijk ook veel code opnieuw gebruiken. Generieke functies zoals `getSiteHTML` en `getTitle` (compleet met uitgebreide afhandeling van excepties) maken webscrapen een stuk gemakkelijker, sneller en betrouwbaarder.