**Immoscraping mit Python: Immobilienangebote automatisiert erfassen und speichern**

05.04.2022 11:38 Uhr Marvin Strathmann

A city skyline at night

Description automatically generated with low confidence

(Bild: Fit Ztudio/Shutterstock.com)

**Mit einem Webscraper wie Scrapy behalten Sie die Immobilienangebote im Blick. Die Daten lassen sich maschinenlesbar speichern und automatisch auswerten.**

Der Immobilienmarkt ist heiß umkämpft. Die begehrten Wunschhäuser oder -wohnungen sind oft nur Minuten auf Immobilienportalen online, bevor der Verkäufer hunderte Mails erhält. Mit einem Scraping-Tool wie Scrapy werten Sie Suchergebnisse für Immobilien automatisch aus und können die Rohdaten weiterverarbeiten.

PROGRAMMIEREN IN PYTHON

* [**Mit Python und Pandas die eigenen Einkaufsdaten analysieren [1]**](https://www.heise.de/ratgeber/Mit-Python-und-Pandas-die-eigenen-Einkaufsdaten-analysieren-6668748.html)
* [**Python-Entwicklungsumgebungen für Einsteiger und Profis [2]**](https://www.heise.de/ratgeber/Python-Entwicklungsumgebungen-fuer-Einsteiger-und-Profis-6360587.html)
* [**Python schnell einrichten: So geht's auf Windows, Linux und macOS [3]**](https://www.heise.de/ratgeber/Python-schnell-einrichten-So-geht-s-auf-Windows-Linux-und-macOS-6355248.html)
* [**Python für alle: Gute Gründe für das Lernen der beliebten Programmiersprache [4]**](https://www.heise.de/hintergrund/Python-fuer-alle-Gute-Gruende-fuer-das-Lernen-der-beliebten-Programmiersprache-6352098.html)
* [**Algorithmus für Linien in Voxelgittern [5]**](https://www.heise.de/hintergrund/Programmieren-mit-Python-Algorithmus-fuer-Linien-in-Voxelgittern-6283373.html)
* [**Dateien und Verzeichnisse beherrschen [6]**](https://www.heise.de/ratgeber/Mit-Python-Dateien-und-Verzeichnisse-beherrschen-4797849.html)
* [**Bedienoberfläche via PyQt erstellen [7]**](https://www.heise.de/ratgeber/Programmieren-mit-Python-Bedienoberflaeche-via-PyQt-erstellen-4949489.html)
* [**Schnittstellen entwickeln mit Pycharm und FastAPI [8]**](https://www.heise.de/ratgeber/Programmieren-mit-Python-Schnittstellen-entwickeln-mit-Pycharm-und-FastAPI-4940182.html)
* [**Daten verarbeiten mit Python: Pandas-Bibliothek für SQL-Umsteiger [9]**](https://www.heise.de/ratgeber/Daten-verarbeiten-mit-Python-Pandas-Bibliothek-fuer-SQL-Umsteiger-4801847.html)
* [**Maschinelles Lernen: Neuronale Netze generieren Inhalte [10]**](https://www.heise.de/hintergrund/Maschinelles-Lernen-Neuronale-Netze-generieren-Inhalte-4971496.html)
* [**Python: Einfache grafische Bedienoberfläche mit Tkinter erstellen [11]**](https://www.heise.de/ratgeber/Python-Einfache-grafische-Bedienoberflaeche-mit-Tkinter-erstellen-4859082.html)
* [**Die besten Python-Projekte: Spiel programmieren, GUI erstellen, Webseite scrapen [12]**](https://www.heise.de/ratgeber/Die-besten-Python-Projekte-Spiel-programmieren-GUI-erstellen-Webseite-scrapen-4869382.html)
* [**RESTful-APIs mit Python und Flask entwickeln [13]**](https://www.heise.de/ratgeber/RESTful-APIs-mit-Python-und-Flask-entwickeln-4940083.html)
* [**Mit Python und Blender große Datenmengen visualisieren [14]**](https://www.heise.de/ratgeber/Programmieren-Mit-Python-und-Blender-grosse-Datenmengen-visualisieren-4986548.html)
* [**Machine Learing: Mit PyTorch ML-Modelle trainieren und deployen [15]**](https://www.heise.de/ratgeber/Machine-Learing-Mit-PyTorch-ML-Modelle-trainieren-und-deployen-4987031.html)

In diesem Artikel erklären wir Ihnen die Grundlagen des Scrapy-Frameworks praktisch. Zunächst soll Scrapy einen kleinen Text in der Konsole ausgeben, dann Daten aus einem Wikipedia-Artikel holen. Dabei stellen wir die Scrapy-Shell vor, die das Debugging erleichtert. Schließlich zeigen wir, wie Sie die Suchergebnisse eines Immobilienportals automatisch scrapen und die Daten zum Weiterverarbeiten als JSON- oder CSV-Datei speichern.

**Scrapy gegen BeautifulSoup**

Eine andere Bibliothek fürs Webscraping mit Python ist BeautifulSoup. Dabei ist BeautifulSoup eher eine Bibliothek fürs Parsen, Scrapy dagegen ein komplettes Framework fürs Webscraping. Das bedeutet, dass der Entwickler neben BeautifulSoup noch andere Bibliotheken benötigt, etwa Requests, um eine Anfrage an eine Website zu stellen. Scrapy liefert dafür alles mit, was man braucht.

LESEN SIE AUCH

**[[](https://www.heise.de/ratgeber/Scraping-Mit-Python-Daten-von-beliebigen-Websites-auslesen-4659822.html)](https://www.heise.de/ratgeber/Scraping-Mit-Python-Daten-von-beliebigen-Websites-auslesen-4659822.html" \o "Mit Python Daten von beliebigen Websites auslesen am Beispiel Talkshows)**

**[Mit Python Daten von beliebigen Websites auslesen am Beispiel Talkshows](https://www.heise.de/ratgeber/Scraping-Mit-Python-Daten-von-beliebigen-Websites-auslesen-4659822.html" \o "Mit Python Daten von beliebigen Websites auslesen am Beispiel Talkshows)**

**[[16]](https://www.heise.de/ratgeber/Scraping-Mit-Python-Daten-von-beliebigen-Websites-auslesen-4659822.html" \o "Mit Python Daten von beliebigen Websites auslesen am Beispiel Talkshows)**

BeautifulSoup ist dafür etwas einfacher zu lernen. Es eignet sich gut für Projekte, bei denen man mal schnell was aus dem Internet ziehen möchte. Scrapy lässt sich dagegen besser warten und einfacher anpassen, falls eine Website mal seine Struktur ändert. Auch ist es mit Scrapy relativ einfach möglich, mehrere Websites gleichzeitig zu scrapen.

Scrapy arbeitet mit Spiders. Ein Spider ist eine spezielle Klasse, in der der Entwickler festlegt, wie Scrapy eine Seite oder mehrere Seiten scrapen soll. Der Scraping-Weg von Scrapy ist unkompliziert: Es stellt Anfrage an bestimmte URLs, verarbeitet die Antwort der Websites als eigenes Objekt und extrahiert mit Selektoren die gewünschten Daten daraus.

**Hello World**

Scrapy lässt sich wie gewohnt über den Paketmanager Pip installieren:

pip install scrapy

Dann erstellen Sie einen neuen Ordner, in dem später alle Dateien landen sollen. Hier ist es sinnvoll sein, eine virtuelle Umgebung für Python aufzusetzen, um nicht mit anderen Projekten in Konflikt zu geraten.

Man navigiert in der Kommandozeile zum Ordner und führt scrapy startproject helloworld aus. helloworld ist in diesem Beispiel unser Projektname, Sie dürfen sich da natürlich kreativ austoben und einen Namen Ihrer Wahl vergeben.

Scrapy hat nun neue Ordner und Dateien erstellt:

* helloworld
  + scrapy.cfg
  + helloworld
    - \_\_init\_\_.py
    - items.py
    - middlewares.py
    - pipelines.py
    - settings.py
    - spiders
      * \_\_init\_\_.py

Im Ordner helloworld/helloworld/spiders lässt sich nun eine neue Python-Datei erstellen, etwa hellworld\_spider.py. Hier bauen wir jetzt unseren ersten Beispiel-Spider. Es beginnt mit einem klassischen Import, um Scrapy zu nutzen:

import scrapy

Nun baut man sich einen Spider als neue Klasse. Bei uns heißt die Klasse HelloworldSpider:

class HelloworldSpider(scrapy.Spider):

Jeder Spider erbt seine Eigenschaften von scrapy.Spider. Zudem benötigt jeder Spider einen Namen, den man anschließend festlegt:

name = "helloworld"

Zusätzlich ist eine URL-Liste notwendig, damit Scrapy weiß, wo das Programm mit dem Scrapen starten soll:

start\_urls = ["https://www.heise.de/"]

Unser helloworld-Spider kennt bisher nur die Startseite von heise online – sie ist vorerst bloß ein Platzhalter ohne Bedeutung. Anschließend benötigt der Spider standardmäßig eine parse()-Funktion, um die Antwort der angefragten URL zu verarbeiten:

def parse(self, response):

Scrapy holt sich von der angefragten Website eine Antwort. Diese Antwort ist das response-Objekt. Anschließend scrapen wir noch nichts von einer Website, sondern schreiben erstmal in die Kommandozeile mit logger.info():

self.logger.info("Hello World!")​

pass​

Das war's. Der erste Spider ist in wenigen Codezeilen erstellt:

class HelloworldSpider(scrapy.Spider):

name = "helloworld"

start\_urls = ["https://www.heise.de/"]

def parse(self, response):

self.logger.info("Hello World!")

pass

In der Kommandozeile navigiert man nun zum Projektordner und führt den Spider aus:

scrapy crawl helloworld

helloworld ist der Name, den man vorher dem Spider über die name-Variable gegeben hat. Scrapy startet den Scrape-Prozess und ruft heise.de auf. Mit der Antwort der Website macht das Programm aber nichts, es schreibt ja nur "Hello World!" in die Ausgabe. Daher findet sich nun dieser Eintrag in der Konsole:

[helloworld] INFO: Hello World!

Eventuell müssen Sie etwas länger suchen, Scrapy schreibt viele Statusmeldungen in die Konsole. Anschließend beendet Scrapy den Spider.

**Wikipedia scrapen**

Zum Scrapen echter Informationen verwenden wir uns unserem Beispiel nun den Wikipedia-Artikel über Hallo-Welt-Programme unter [**https://de.wikipedia.org/wiki/Hallo-Welt-Programm [17]**](https://de.wikipedia.org/wiki/Hallo-Welt-Programm) als Start-URL:

start\_urls = ["https://de.wikipedia.org/wiki/Hallo-Welt-Programm"]

Aus dieser Website möchte der Entwickler den Titel der Website extrahieren. Dieser Titel steckt schön maschinenlesbar in einem <h1>-Element des HTMLs.

Für ein effizientes Scraping ist es wichtig, die Seiten gut zu analysieren und sich zu überlegen, wie man möglichst eindeutig an die Daten kommt. Dafür eignen sich die Webinspektoren der Browser hervorragend. Im Firefox und in Chrome rufen Sie die Seite auf und drücken F12. Im aufploppenden Webentwickler-Menü erscheint der Quelltext der Seite. Den Webinspektor findet sich oben links: ein Pfeil in einem abgerundeten Rechteck.

Drücken Sie auf den Webinspektor und fahren Sie anschließend auf der Webseite über ein Element, etwa über das Logo oder eine große Überschrift. Wenn man mit aktiviertem Webinspektor auf ein Element klickt, springt das Webentwickler-Menü direkt zum passenden HTML-Quelltext der Website. Dort sehen Sie, in welchen HTML-Tags die Entwickler den Inhalt gepackt haben, also in welche Elemente. Außerdem erfährt man, wie die Tags verschachtelt und über welche Attribute sie ansprechbar sind.

Jedes Element auf einer Website ist über einen CSS-Selektor ansprechbar. [**Die Übersicht auf W3 Schools [18]**](https://www.w3schools.com/cssref/css_selectors.asp) zeigt, wie Sie mit verschiedenen Selektoren auf Inhalte verweisen. Auf der Wikipedia-Seite gibt es zum Glück nur ein <h1>-Element mit dem gesuchten Inhalt: Hallo-Welt-Programm. Mit Scrapy spricht man es daher in der parse()-Funktion so an:

response.css("h1::text").get()

response ist ja das Antwort-Objekt, das Scrapy nach dem Scrapen zurückgibt. Mit css() durchsucht das Programm dieses Objekt anhand eines CSS-Selektors. Der Selektor steht mit h1::text in Anführungszeichen, so holt sich das Programm den Text im <h1>-Element.

response.css("h1::text") würde nur ein weiteres Objekt erzeugen, erst mit get() holt man sich den textlichen Inhalt dieses Objekts – hier ist das der gesuchte Text "Hallo-Welt-Programm". get() gibt nur ein Ergebnis aus, wenn es mehrere gibt, dann zeigt get() auch nur das erste Ergebnis an. Eine Liste mit allen Ergebnissen liefert getall().

Das Ergebnis von response.css("h1::text").get() lässt sich nun wieder in den Log von Scrapy schreiben, wie eben bei der Hallo-Welt-Ausgabe:

self.logger.info(response.css("h1::text").get())

Wenn im Log von Scrapy [helloworld] INFO: Hallo-Welt-Programm auftaucht, hat alles geklappt.

**Scrapy-Shell**

Um das Debugging zu erleichtern, lässt sich die Scrapy-Shell nutzen. Man startet sie zusammen mit der zu scrapenden URL in Anführungszeichen in der Kommandozeile:

scrapy shell "https://de.wikipedia.org/wiki/Hallo-Welt-Programm"

Scrapy gibt einige Statusinformationen aus und zeigt am Ende, mit welchen Objekten der Nutzer interagieren kann. Gibt man nun response.css("h1::text").get() direkt in die Konsole ein, zeigt die Scrapy-Shell 'Hallo-Welt-Programm' in der Ausgabe an.

Damit lässt sich herumspielen und ausprobieren. So gibt die Eingabe response.css("h1").get() etwa das komplette <h1>-Element aus, nicht nur den Text:

<h1 id="firstHeading">Hallo-Welt-Programm</h**1**>

Zusätzlich ist der Befehl view(response) nützlich. Damit öffnet Scrapy einen Browser und zeigt dort die Antwort der Website an. So erkennt man etwa, ob ein bestimmter Teil einer Website nicht in der Response angekommen ist und kann seinen Code anpassen. Hier wird etwa auch sichtbar, falls eine Website automatische Crawler wie Scrapy blockiert oder ein Captcha verlangt.

STRG+D beendet die Shell unter Linux und macOS, STRG-Z funktioniert unter Windows.

**Immobilienangebote scrapen**

Mit diesem Scrapy-Grundwissen gehen wir jetzt die Immobilienangebote an. In diesem Beispiel wollen wir die Daten der Website immowelt.de durchsuchen.

Dafür benötigt man zuerst eine Start-URL. Immowelt listet verschiedene Anzeigen für Immobilien zum Kaufen oder Mieten auf. Über die Suchmaske auf der Startseite lässt sich die passende Unterseite mit den Ergebnissen anzeigen. So sind etwa Häuser zum Kaufen in Mainz über diese URL erreichbar: [**https://www.immowelt.de/liste/mainz/haeuser/kaufen [19]**](https://www.immowelt.de/liste/mainz/haeuser/kaufen). Das wird unsere Beispiel-URL für dieses Projekt.

Bevor man nun einen Spider dafür baut, sollte man die Seite mit dem F12-Inspektor des Browsers untersuchen, um die passenden CSS-Selektoren zu finden. Mit der Shell von Scrapy lassen sich die Selektoren dann ausprobieren, um die richtigen Inhalte zu extrahieren.

Der Browser-Inspektor verrät dem Nutzer, dass jedes Immobilienangebot der Suche in einem <div>-Element steckt, das mit der Klasse EstateItem-1c115 als Parameter-Wert daherkommt. Der Titel der Immobilienanzeige steckt anschließend verschachtelt in einem <h2>-Element.

**Titel scrapen**

Um den Titel zu extrahieren, wirft man erst die Scrapy-Shell an:

scrapy shell "https://www.immowelt.de/liste/mainz/haeuser/kaufen"

Ein anschließendes response.css("div[class='EstateItem-1c115']").get() holt das erste Angebot in die Konsole und zeigt den Quelltext an. Der CSS-Selektor div[class='EstateItem-1c115'] zeigt also schonmal auf das richtige Element. Um jetzt den Text einer <h2>-Überschrift daraus zu ziehen, packt man ein h2::text mit Leerzeichen getrennt dahinter – wie beim Wikipedia-Beispiel: response.css("div[class='EstateItem-1c115'] h2::text").get().

Als Ergebnis erschien bei unserem Test dieser Immobilientitel in der Konsole:

'FALANGA IMMOBILIEN - SOLIDES und ORDENTLICH vermietetes 5-FAM.-HAUS in zentraler Mainzer NEUSTADT!'

Je nach Zeitpunkt kann der Titel bei Ihnen natürlich anders lauten. Aber der Code funktioniert, den Titel des ersten Suchergebnisses kann Scrapy bereits auslesen. Um alle Ergebnisse auf einmal anzuzeigen, ändert man das get() in getall(). Schon zeigt Scrapy die komplette Liste mit allen Titeln an.

**Spider bauen**

Auf dieser Basis lässt sich ein Spider bauen. Dafür legt man am besten im Ordner spiders eine neue Python-Datei an und nennt sie etwa immo\_spider.py. Die Klasse nennen wir hier ImmoSpider, als Name kommt immo zum Einsatz und die Start-URL ist die Immowelt-URL von eben:

class ImmoSpider(scrapy.Spider):​

name = "immo"​

start\_urls = ["https://www.immowelt.de/liste/mainz/haeuser/kaufen"]​

Anschließend gehts an die parse()-Funktion. Hier sollte man alle Immobilieneinträge in eine eigene Variable auslagern – sie stecken ja alle in <div>-Elementen mit der Klasse EstateItem-1c115:

immobilien = response.css("div[class='EstateItem-1c115']")

Nun lässt sich eine for-Schleife erstellen, die jeden Immobilieneintrag einzeln durchgeht – also jedes <div>-Element mit der Klasse EstateItem-1c115:

for immobilie in immobilien:

Um die Ergebnisse auszugeben, nutzen wir in diesem Beispiel nicht return, sondern yield. yield gibt ein Generator-Objekt zurück, über das man iterieren kann. Das ist hauptsächlich bei großen Datenmengen sinnvoll – nicht unbedingt in Mainz, aber wenn Sie Wohnungen in Berlin mieten wollen, können es sehr viele Einträge werden.

Hier soll yield ein Dictionary zurückgeben, das verschiedenen Informationen über ein Immobilienangebot enthält. Dank der Experimente in der Scrapy-Shell wissen Sie bereits, wie Sie an den Titel einer Anzeige kommen, nämlich mit dem CSS-Selektor div[class='EstateItem-1c115'] h2::text. Der erste Teil dieses Selektors steckt bereits in der Variable immobilien, über die Scrapy in der for-Schleife iteriert. Den Titel des Eintrags erhält man also so:

immobilie.css("h2::text").get()

Der Titel lässt sich nun mit yield in einem Dictionary speichern:

yield {"Titel": immobilie.css("h2::text").get()}

Zusammen mit der for-Schleife sieht die parse-Funktion so aus:

def parse(self, response):​

immobilien = response.css("div[class='EstateItem-1c115']")​

for immobilie in immobilien:​

yield {"Titel": immobilie.css("h2::text").get()}​

Die Ausgabe lässt sich in der Kommandozeile überprüfen, indem man den Spider startet:

scrapy crawl immo

In der Ausgabe sollten nun alle Titel der Einträge erscheinen – und zwar in der vorher festgelegten Form:

{'Titel': 'FALANGA IMMOBILIEN - SOLIDES und ORDENTLICH vermietetes 5-FAM.-HAUS in zentraler Mainzer NEUSTADT!'}​

…​

{'Titel': 'Mehrfamilienhaus in 55116 Mainz'}​

…​

{'Titel': 'Mainz-Drais: Herrliches Grundstück (1.002 qm) für Mehrgenerationenhaus/Zweifamilienhaus'}​

...

**Preis, Wohnfläche, Standort**

Der Titel ist ja schon ganz nett, aber der Nutzer möchte sicher noch weitere Informationen zum Angebot erhalten, etwa den Preis, die Wohnfläche und den Standort. Dabei hilft wieder der Inspektor, mit der Shell von Scrapy überprüfen Sie Ihre CSS-Selektoren.

Der veranschlagte Preis steckt in einem <div>-Element mit dem Parameter data-test und dem Wert price. Die Wohnfläche versteckt sich in einem ähnlichen <div>-Element, nur der Wert lautet area. Den Standort hat Immowelt etwas anders eingebunden, er steckt in einem <span>-Element, das wiederum in einem <div>-Element mit der Klasse IconFact-e8a23 steckt. Ein angefügtes ::text holt wie gewohnt den Text aus den gewünschten Elementen.

Alle CSS-Selektoren sehen also so aus:

* Preis: div[data-test='price']::text
* Wohnfläche: div[data-test='area']::text
* Standort: div[class='IconFact-e8a23'] span::text

Diese Daten fügt man nun der yield-Ausgabe hinzu:

yield {​

"Titel": immobilie.css("h2::text").get(), ​

"Preis": immobilie.css("div[data-test='price']::text").get(), ​

"Wohnfläche": immobilie.css("div[data-test='area']::text").get(), ​

"Standort": immobilie.css("div[class='IconFact-e8a23'] span::text").get()​

}​

Läuft der Spider einmal durch, stehen nun für jedes Immobilienangebot die gescrapten Daten zur Verfügung:

{'Titel': 'Mehrgenerationenhaus mit bewohntem Erdgeschoss: MFH mit 3 Einheiten in Bretzenheim', 'Preis': '879.000 €', 'Wohnfläche': '300 m²', 'Standort': 'Mainz (Bretzenheim)'}

**Ausgabe**

Noch stehen diese Daten allerdings nur in der Konsole. Um sie etwa als JSON-Datei zu speichern, tippt der Nutzer scrapy crawl immo -o immobilien.json in die Konsole. -o steht für Output und immobilien.json ist der Dateiname. Auch eine CSV-Datei ist möglich.

Die Datei landet im Projektordner und ist nicht UTF-8 kodiert – was im Deutschen mit seinen Äs, Ös und Üs immer eine schlechte Idee ist und irgendwann garantiert Ärger macht. Um das zu ändern, lässt sich in der Datei settings.py im spiders-Ordner diese Zeile hinzufügen:

FEED\_EXPORT\_ENCODING = 'utf-8'

Wenn Sie scrapy crawl immo -o immobilien.json erneut laufen lassen, werden Sie merken, dass Scrapy die neuen Daten immer hinter die alten hängt. Das ist schön, wenn Sie das so haben wollen, bei manchen Projekten ist das aber nicht sinnvoll, dann sollte der Nutzer die Datei vorher löschen. Und zwar mit dem Befehl rm:

rm immobilien.json; scrapy crawl immo -o immobilien.json

**A screenshot of a computer

Description automatically generated with medium confidence**

***yield* gibt alle Daten im festgelegten Format zurück.**

**Weiterverarbeiten**

Die JSON-Datei lässt sich dann etwa mit Python weiterverarbeiten. Sie können etwa Durchschnittspreise bilden und so den Markt im Auge behalten. Auch ein Preisalarm wäre möglich, etwa indem das Python-Programm Ihnen eine Mail schickt. Zusätzlich können Sie etwa die Daten von verschiedenen Zeitpunkten miteinander vergleichen und so die Anzahl der Angebote im Blick behalten. Schließlich lässt sich mit etwas Aufwand sogar ein Bot erstellen, der automatisch Anfragen verschickt, sobald eine Immobilie unter einem bestimmten Preis zu haben ist.

Scrapy liefert dafür ein eigenes Modul mit, um solche Benachrichtigungen etwa per Mail zu verschicken. Ein einfacher Import reicht:

from scrapy.mail import MailSender

Der Nutzer möchte eine Mail erhalten, wenn Immobilien unter 800.000 Euro zu haben sind. Außerdem möchte er wissen, wie viele Immobilien zu diesem Preis zu haben sind. Dafür benötigt man erst einen Counter für die Anzahl der Immobilien:

anzahl\_guenstige\_immobilien = 0

In der parse-Funktion nimmt man nun den Preis jeder Immobilie, der als String vorliegt, und wandelt ihn in einen Integer um:

for immobilie in immobilien:​

preis = immobilie.css("div[data-test='price']::text").get()​

preis = int(preis[:-2].replace(".", ""))​

Das [:-2] ist notwendig, weil € hinter jedem Preis steht, der Code entfernt die beiden letzten Zeichen. replace(".", "") löscht die in Deutschland üblichen Tausender-Trennzeichen.

Nun folgt die Bedingung und das Programm zählt den Zähler hoch, sobald der Preis unterschritten wird:

if preis < 800000:​

anzahl\_guenstige\_immobilien += 1​

Nach der for-Schleife soll Scrapy die Mail verschicken. Allerdings nur dann, wenn es überhaupt eine Immobilie gibt, die so günstig ist:

if anzahl\_guenstige\_immobilien > 0:

Es gibt zwei Möglichkeiten, wie der Nutzer die Daten für den Mailversand hinterlegen kann. Am einfachsten ist es, die Daten in der MailSender()-Funktion direkt zu hinterlegen und in der Variable mailer auszulagern:

mailer = MailSender(smtphost="smtp.gmail.com", mailfrom="Ihre.Absende-Mailadresse@gmail.com", smtpuser=" Ihre.Absende-Mailadresse @gmail.com", smtpport=465, smtppass="Ihr Passwort", smtpssl=True)

Wir haben in diesem Beispiel einen Versand von einer Google-Mail-Adresse über SMTP eingerichtet. [**Google hat alle relevanten Daten dafür auf einer Supportseite zusammengetragen [20]**](https://support.google.com/mail/answer/7126229?hl=de#zippy=%2Cschritt-smtp--und-andere-einstellungen-im-e-mail-client-%C3%A4ndern), etwa Port und Server. Den smtpuser-Parameter füllt man mit seiner Mailadresse und smtppass möchte das Passwort wissen. Zusätzlich müssen Sie [**in Ihren Gmail-Einstellungen festlegen [21]**](https://myaccount.google.com/lesssecureapps), dass weniger sichere Apps Mails verschicken dürfen – also Apps, die sich nur mit Benutzername und Passwort einloggen.

LESEN SIE AUCH

**[[A picture containing text

Description automatically generated](https://www.heise.de/ratgeber/SMTP-E-Mails-automatisch-mit-Python-versenden-6282155.html)](https://www.heise.de/ratgeber/SMTP-E-Mails-automatisch-mit-Python-versenden-6282155.html" \o "SMTP: E-Mails automatisch mit Python versenden)**

**[SMTP: E-Mails automatisch mit Python versenden](https://www.heise.de/ratgeber/SMTP-E-Mails-automatisch-mit-Python-versenden-6282155.html" \o "SMTP: E-Mails automatisch mit Python versenden)**

**[[22]](https://www.heise.de/ratgeber/SMTP-E-Mails-automatisch-mit-Python-versenden-6282155.html" \o "SMTP: E-Mails automatisch mit Python versenden)**

Sind alle Parameter ausgefüllt, schreibt man nun den Text der Mail:

mail\_body = f"""Hi,​

es gibt aktuell {anzahl\_guenstige\_immobilien} Immobilien, die günstiger als 800.000 Euro sind.​

Viele Grüße​

Dein Python-Programm​

"""​

Hier nutzen wir einen F-String, das bedeutet, die Variable anzahl\_guenstige\_immobilien in geschwungenen Klammern wird automatisch durch den Wert ersetzt und fügt sich in den Text ein.

Jetzt muss das Programm nur noch auf Senden drücken, mit einem mailer.send():

mailer.send(to=["Ihre.Empfänger-Adresse@Mail.de"], subject="Immobilien-Benachrichtigung", body=mail\_body)

Der Parameter body verweist auf die eben geschriebene Nachricht und der Parameter to möchte mit der Empfänger-Adresse gefüttert werden.

Crawlt man nun mit dem Spider die Adresse, gibts am Ende zusätzlich eine Benachrichtigungsmail. Wer viele Mails verschickt, kann die Daten in der MailSender()-Funktion auch auslagern, nämlich in die settings.py des Projekts. Dort trägt man diese Daten ein:

MAIL\_FROM = "Ihre.Absende-Mailadresse@gmail.com"​

MAIL\_HOST = "smtp.gmail.com"​

MAIL\_PORT = 465​

MAIL\_USER = "Ihre.Absende-Mailadresse@gmail.com"​

MAIL\_PASS = "Ihr Passwort"​

MAIL\_SSL = True​

Um die Settings-Daten im Programm zu nutzen, ist ein weiterer Import nötig:

from scrapy.utils.project import get\_project\_settings​

Die Funktion get\_project\_settings() ruft man nun in der letzten if-Abfrage wieder auf:

settings = get\_project\_settings()​

Anschließend muss nur noch die mailer-Variable angepasst werden:

mailer = MailSender.from\_settings(settings)

MailSender() ist dieses Mal nicht mit Parametern gefüllt, sondern holt sich die vorher festgelegten Daten aus den Einstellungen.

**Fazit**

In Scrapy muss man sich dafür etwas einarbeiten. Es ist komplexer als BeautifulSoup, bringt aber auch mehr Werkzeuge mit. [**Die offizielle Dokumentation ist immer ein guter Anlaufpunkt [23]**](https://docs.scrapy.org/en/latest/). Zusätzlich [**stellen die Entwickler von Scrapy Tutorials bereit [24]**](https://docs.scrapy.org/en/latest/intro/tutorial.html).

LESEN SIE AUCH

**[[](https://www.heise.de/ratgeber/Python-Websitedaten-nach-einem-Login-auslesen-4681895.html)](https://www.heise.de/ratgeber/Python-Websitedaten-nach-einem-Login-auslesen-4681895.html" \o "Web-Scraping mit Python: Websitedaten nach einem Login auslesen)**

**[Web-Scraping mit Python: Websitedaten nach einem Login auslesen](https://www.heise.de/ratgeber/Python-Websitedaten-nach-einem-Login-auslesen-4681895.html" \o "Web-Scraping mit Python: Websitedaten nach einem Login auslesen)**

**[[25]](https://www.heise.de/ratgeber/Python-Websitedaten-nach-einem-Login-auslesen-4681895.html" \o "Web-Scraping mit Python: Websitedaten nach einem Login auslesen)**

Egal, welche Scraping-Methode man verwendet, man muss die Zielseite immer genau analysieren und sich zum Teil kreative Wege überlegen, um an die gewünschten Daten zu kommen. Aber wer die richtigen Rohdaten automatisch extrahiert und verarbeiten lässt, kann damit im heiß umkämpften Immobilienmarkt schneller und effizienter sein als die Konkurrenz. (**[str [26]](mailto:str@heise.de" \o "Marvin Strathmann)**)

**URL dieses Artikels:**  
https://www.heise.de/-6659020

**Links in diesem Artikel:**  
**[1]** https://www.heise.de/ratgeber/Mit-Python-und-Pandas-die-eigenen-Einkaufsdaten-analysieren-6668748.html  
**[2]** https://www.heise.de/ratgeber/Python-Entwicklungsumgebungen-fuer-Einsteiger-und-Profis-6360587.html  
**[3]** https://www.heise.de/ratgeber/Python-schnell-einrichten-So-geht-s-auf-Windows-Linux-und-macOS-6355248.html  
**[4]** https://www.heise.de/hintergrund/Python-fuer-alle-Gute-Gruende-fuer-das-Lernen-der-beliebten-Programmiersprache-6352098.html  
**[5]** https://www.heise.de/hintergrund/Programmieren-mit-Python-Algorithmus-fuer-Linien-in-Voxelgittern-6283373.html  
**[6]** https://www.heise.de/ratgeber/Mit-Python-Dateien-und-Verzeichnisse-beherrschen-4797849.html  
**[7]** https://www.heise.de/ratgeber/Programmieren-mit-Python-Bedienoberflaeche-via-PyQt-erstellen-4949489.html  
**[8]** https://www.heise.de/ratgeber/Programmieren-mit-Python-Schnittstellen-entwickeln-mit-Pycharm-und-FastAPI-4940182.html  
**[9]** https://www.heise.de/ratgeber/Daten-verarbeiten-mit-Python-Pandas-Bibliothek-fuer-SQL-Umsteiger-4801847.html  
**[10]** https://www.heise.de/hintergrund/Maschinelles-Lernen-Neuronale-Netze-generieren-Inhalte-4971496.html  
**[11]** https://www.heise.de/ratgeber/Python-Einfache-grafische-Bedienoberflaeche-mit-Tkinter-erstellen-4859082.html  
**[12]** https://www.heise.de/ratgeber/Die-besten-Python-Projekte-Spiel-programmieren-GUI-erstellen-Webseite-scrapen-4869382.html  
**[13]** https://www.heise.de/ratgeber/RESTful-APIs-mit-Python-und-Flask-entwickeln-4940083.html  
**[14]** https://www.heise.de/ratgeber/Programmieren-Mit-Python-und-Blender-grosse-Datenmengen-visualisieren-4986548.html  
**[15]** https://www.heise.de/ratgeber/Machine-Learing-Mit-PyTorch-ML-Modelle-trainieren-und-deployen-4987031.html  
**[16]** https://www.heise.de/ratgeber/Scraping-Mit-Python-Daten-von-beliebigen-Websites-auslesen-4659822.html  
**[17]** https://de.wikipedia.org/wiki/Hallo-Welt-Programm  
**[18]** https://www.w3schools.com/cssref/css\_selectors.asp  
**[19]** https://www.immowelt.de/liste/mainz/haeuser/kaufen  
**[20]** https://support.google.com/mail/answer/7126229?hl=de#zippy=%2Cschritt-smtp--und-andere-einstellungen-im-e-mail-client-%C3%A4ndern  
**[21]** https://myaccount.google.com/lesssecureapps  
**[22]** https://www.heise.de/ratgeber/SMTP-E-Mails-automatisch-mit-Python-versenden-6282155.html  
**[23]** https://docs.scrapy.org/en/latest/  
**[24]** https://docs.scrapy.org/en/latest/intro/tutorial.html  
**[25]** https://www.heise.de/ratgeber/Python-Websitedaten-nach-einem-Login-auslesen-4681895.html  
**[26]** mailto:str@heise.de

*Copyright © 2022 Heise Medien*