Scrapy

A Fast and Powerful Scraping and Web Crawling Framework

MB, TS, TS

27. Juni 2017

Übersicht

Allgemeines: Definition Scaping

Laut Wikipedia:
Der Begriff Screen Scraping [..] umfasst generell alle Verfahren zum Auslesen von Texten aus Computerbildschirmen. Gegenwärtig wird der Ausdruck jedoch beinahe ausschließlich in Bezug auf Webseiten verwendet [..]. In diesem Fall bezeichnet Screen Scraping speziell die Technologien, die der Gewinnung von Informationen durch gezieltes Extrahieren der benötigten Daten dienen.

Allgemeines: Aha. . . Und was ist dieses Scrapy?

- „A Fast and Powerful Scraping and Web Crawling Framework“
- Also ein freies und open-source-Framework für Python, welches web-scraping beherrscht.
- https://scrapy.org/

Installation

- mit pip (erfordert vorhandene Python-Installation):

  ```
  pip3 install scrapy
  ```

- mit anaconda (in Windows einfacher, da numpy, etc... bereits enthalten):

  ```
  conda install -c conda-forge scrapy
  ```

Extraktion von Daten (1)

```
<div class="quote">
    <span class="text">"The world as we have..."</span>
    <small class="author">Albert Einstein</small>
</div>
<div class="quote">
    <span class="text">"It is our choices,..."</span>
    <small class="author">J.K. Rowling</small>
</div>
...
```

```
def parse(self, resp):
  res1 = resp.css('div small')
  #[<Selector data='<small...">'>, <Selector data='<small...">'>, ...]
  res2 = resp.css('div small::text')
  #[<Selector data='Albert Einstein'>, <Selector data='J.K. Rowling'>, ...]
  res3 = resp.css('div small::text').extract()
  #['Albert Einstein', 'J.K. Rowling', ...]
  res4 = resp.css('div small::text').extract_first()
  #Albert Einstein
```

Extraktion von Daten (2)

- `resp.css('div small')` selektiert mehrere Elemente im DOM mit dem CSS-Selektor `'div small'` (d.h. es findet alle `<small>`-Elemente innerhalb von `<div>`-Elementen)
- `resp.css('div small::text')` selektiert den Text innerhalb jedes gefundenen `<small>...</small>`-Tags
- `resp.css('div small::text').extract()` liefert die selektierten Texte als Liste von Python-Strings
- `resp.css('div small::text').extract_first()` liefert den 1. Treffer davon

Extraktion von Daten (3)

- das Ergebnis einer Selektion kann selbst wieder zum Selektieren genutzt werden
- folgendes Beispiel selektiert alle Zitate einer Website, und gibt je Zitat als Datensatz Autor und Wortlaut zurück:

```
def parse(self, response):
  quotes = response.css('div.quote')
  for q in quotes:
    yield {
      'author': q.css('.author::text').extract_first(),
      'text': q.css('.text::text').extract_first()
    }
```

Shell: Vorführung: E-mails farmen...

- In der ersten Übung wollen wir uns zunächst mit einigen Befehlen von Scrapy in der Shell vertraut machen.
- Die Aufgabe besteht darin die E-mails der HSNR-Dozenten von der offiziellen Homepage zu crawlen.
  - Dazu müssen wir scrapy zunächst mitteilen, mit welcher Website wir arbeiten wollen:

    ```
    scrapy shell 'https://www.hs-niederrhein.de/
    elektrotechnik-informatik/personen/'
    ```

    bzw. unter Windows:

    ```
    scrapy shell "https://www.hs-niederrhein.de/
    elektrotechnik-informatik/personen/"
    ```

Shell: Vorführung: E-mails farmen...

- Anschließend schauen wir uns den HTML-Code der entsprechenden Seite an und überlegen uns, auf welche Elemente wir zugreifen müssen.
- Offensichtlich besteh die Dozentenliste aus einer Reihe von div-Containern in denen die Personendaten enthalten sind. Wir greifen mittels

  ```
  response.css("div.tx-iwpersonen-pi1-item-box")
  ```

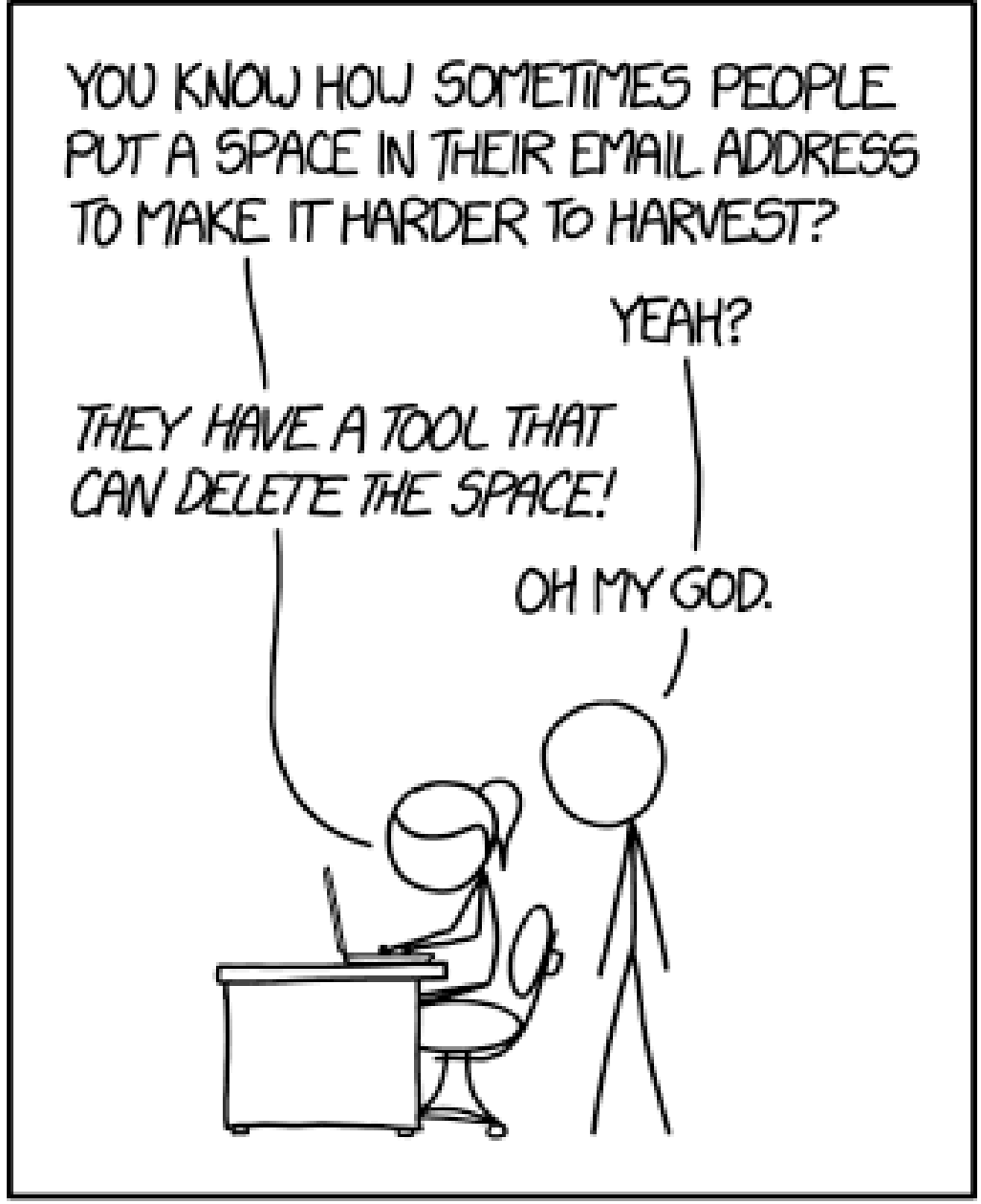
  auf die entsprechenden Container zu und erhalten eine Liste der div-Elemente als Ausgabe.
- Diese kann natürlich auch in eine Variable gespeichert werden...

  ```
  persons = response.css(...)
  ```

Shell: Vorführung: E-mails farmen...

- Um an die E-mail-Adressen zu kommen, müssen wir nun noch einmal in den Quellcode schauen.
- Anscheinend sind die E-Mailadressen als Link mit dem CSS-Klassenselektor „font-size-11“ hinterlegt.
- `persons.css("a.font-size-11::text").extract()`

  liefert dann eine Liste der E-mailadressen zurück.
- Nun müssen wir nur noch das (at) durch ein @-Zeichen ersetzen und können anschließend die Liste verwenden um liebe Rundmails zu versenden. ;-)



LESS-DRAMATIC REVELATIONS FROM THE CIA HACKING DUMP

Shell: Vorführung: E-mails farmen... Kommandos als Spider (1)

```
import scrapy

class MailSpider(scrapy.Spider):
  name = "emails"
  start_urls = [
    'https://www.hs-niederrhein.de/
    elektrotechnik-informatik/personen/',
  ]

  def parse(self, response):
    persons = response.css(
      "div.tx-iwpersonen-pi1-item-box")
```

Shell: Vorführung: E-mails farmen... Kommandos als Spider (2)

```
for person in persons:
  mail = person.css("a.font-size-11::text")
    .extract_first();
  name = person.css(
    "a.tx-iwpersonen-pi1-detaillink::text").extract_fir
  mail = mail.replace("(at)", "@")
  yield{name : mail,}
```

Projekt anlegen

- Verzeichnisstruktur erzeugen:

  ```
  scrapy startproject mynewproject
  ```

- erzeugt ein gleichnamiges Verzeichnis `./mynewproject`
- künftige Spider werden in `./mynewproject/spiders` angelegt
- Konfiguration erfolgt über `./mynewproject/settings.py`
  - ermöglicht z.B. Konfiguration der Beachtung von `robots.txt`

Spider

- spezielle Klassen in Scrapy-Projekten
- von Klasse `scrapy.Spider` abgeleitet
- führen Crawling durch, spezifisch für Websites programmierbar
- Ablauf:
  1. Spider schickt Requests an initiale URLs
  2. scrapy ruft je Response Callback-Methode auf, mit Inhalt als Parameter
  3. Callback-Methode startet ggf. Requests an weitere URLs
  4. Callback-Methode extrahiert Daten des Response und gibt sie zurück
  5. scrapy sammelt alle zurückgegebenen Daten ein, und speichert sie z.B. in einer Datei

Spider?

Spider! Funfact: Spinne ist im Volksmund ein Synonym für Tiere aus der Gruppe der Arachnide - kleine Tiere mit acht Beinen, welche Insekten mit Netzen oder anderen Fallen fangen.

Einfache Spider (1)

```
import scrapy

class Cars(scrapy.Spider):

    name = 'CarsCrawler'
    start_urls = ['https://suchen.mobile.de/fahrzeuge/searc

    def parse(self, response):
        print('parsing {}'.format(response))
        yield { ... }
```

Einfache Spider (2)

- `name` identifiziert den Spider eindeutig im Projekt
- `start_urls` beinhaltet die URLs für die initialen Requests
- `parse()` wird als Callback aufgerufen, der Parameter `response` enthält das Ergebnis des Response
- die Methode gibt mit `print(...)` die aufgerufene URL mit `response` aus
- zurückgegeben wird das Parsingergebnis als `dict`

Einfache Spider (3)

```
import scrapy

class Cars(scrapy.Spider):

    name = 'CarsCrawler'
    start_urls = ['https://suchen.mobile.de/fahrzeuge/searc

    def parse(self, response):
        print('parsing {}'.format(response))
        yield {
            'text': response.css('.h3
                                  .u-text-break-word::text')
                                  .extract_first()
        }
```

Einfache Spider (4)

- das Crawling mit Spider `CarsCrawler` wird über einen eigenen Befehl in der Kommandozeile gestartet:

```
scrapy crawl CarsCrawler -o res.json
```

- scrapy speichert die per `yield` zurückgegebenen Daten in der Datei `res.json` (das Dateiformat ergibt sich aus Suffix)
  - scrapy unterstützt neben json weitere Formate: xml, csv, ...
  - beachte: scrapy leert bei mehrfachem Start des Crawlingbefehls die Datei nicht, sondern hängt die Daten hinten an
- Befehl auch über Pythonskript aufrufbar:

```
if __name__ == '__main__':
  cmd = 'scrapy crawl CarsCrawler -o res.json'
  scrapy.cmdline.execute(cmd.split())
main()
```

- API zum Crawlen?

Einfache Spider (5)

```
import scrapy

class Cars(scrapy.Spider):

    name = 'CarsCrawler'
    start_urls = ['https://suchen.mobile.de/fahrzeuge/searc

    def parse(self, response):
        print('parsing {}'.format(response))
        for offer in response.css('.cBox-body
                                   .cBox-body--resultitem
                                   .rbt-reg.rbt-no-top'):
                yield { ... }
```

Einfache Spider (6)

```
class Cars(scrapy.Spider):

    name = 'CarsCrawler'
    start_urls = ['https://suchen.mobile.de/fahrzeuge/searc

    def parse(self, response):
        print('parsing {}'.format(response))
        for offer in response.css('.cBox-body
                                   .cBox-body--resultitem
                                   .rbt-reg.rbt-no-top'):
            yield {
                'full': offer.css('.h3
                                   .u-text-break-word::text')
                                   .extract_first()
            }
```

Einfache Spider (7)

```
yield {
    'full': offer.css('.h3.u-text-break-word::text')
                 .extract_first(),

    'brand': offer.css('.h3.u-text-break-word::text')
                  .extract_first().split(' ', 1)[0],

    'car-name': offer.css('.h3.u-text-break-word::text')
                     .extract_first().split(' ', 2)[1],

    'price': offer.css('.h3.u-block::text')
                  .extract_first().split(' ', 1)[0]
}
```

Spider: Verschachteltes parsen (1)

```
import scrapy

class Cars(scrapy.Spider):

    name = 'CarsCrawler'
    start_urls = ['https://suchen.mobile.de/fahrzeuge/searc

    def parse(self, response):
        for offer in response.css('.cBox-body.cBox-body--re

            ???
```

Spider: Verschachteltes parsen (2)

```
def parse_details(self, response):
    yield {
    'AC': response.css('#rbt-climatisation-v::text')
                                        .extract_first()
    }

def parse(self, response):
    for offer in response.css('.cBox-body.cBox-body--result
        ???
```

Spider: Verschachteltes parsen (3)

```
def parse_details(self, response):
    yield {
    'AC': response.css('#rbt-climatisation-v::text')
                                    .extract_first()
    }

def parse(self, response):
    for offer in response.css('.cBox-body.cBox-body--result

        details_link = offer.css('a::attr(href)')
                                    .extract_first()
        request = scrapy.Request(details_link,
                            callback=self.parsed_detailes)
```

Spider: Verschachteltes parsen (4)

```
def parse(self, response):
    for offer in response.css('.cBox-body.cBox-body--result

        details_link = offer.css('a::attr(href)')
                                    .extract_first()
        request = scrapy.Request(details_link,
                            callback=self.parsed_detailes)

        if details_link is not None:
            yield request
```

Spider: Verschachteltes parsen (5)

```
def parse(self, response):
    for offer in response.css('.cBox-body.cBox-body--result

        details_link = offer.css('a::attr(href)')
                                    .extract_first()
        request = scrapy.Request(details_link,
                             callback=self.parsed_detailes)

        request.meta['full'] = offer.css('.h3.u-text-break-
        request.meta['price'] = offer.css('.h3.u-block::tex
        ...

        if details_link is not None:
            yield request
```

Spider: Verschachteltes parsen (6)

```
def parse_details(self, response):
    yield {
    'AC': response.css('#rbt-climatisation-v::text')
                                        .extract_first()
    'full': response.meta['full'],
    'brand': response.meta['brand'],
    'car-name': response.meta['car-name'],
    'car-name-all': response.meta['car-name-all'],
    'price': response.meta['price']
    }
```

Spider: Verschachteltes parsen (7)

```
def parse_details(self, response):
    AC = response.css('#rbt-climatisation-v::text')
                                        .extract_first()
    if AC is None or AC.startswith('Keine'):
        response.meta['AC'] = 'Keine'
    else:
        response.meta['AC'] = AC

    yield {
        'AC': response.meta['AC'],
        'full': response.meta['full'],
        'brand': response.meta['brand'],
        'name': response.meta['name'],
        'name-all': response.meta['name-all'],
        'price': response.meta['price'],
    }
```

Links folgen (1)

```
def parse(self, response):
  for q in response.css('div.quote'):
    yield {
      'text': q.css('.text::text').extract_first(),
      'author': q.css('.author::text').extract_first()
    }
  a_selector = 'li.next a::attr(href)'
  hrefs = response.css(a_selector).extract()
  for href in hrefs:
    yield response.follow(href, callback=self.parse)
```

- die Methode liefert zunächst von den Zitaten Autor und Wortlaut
- danach sucht sie alle passenden `<a href=...>`-Elemente und selektiert je Treffer das `href`-Attribut mit `'a::attr(href)'`

Links folgen (2)

- zuletzt extrahiert die Methode die enthaltenen URLs und untersucht sie rekursiv mit
  `response.follow(href, callback=self.parse)`
- scrapy speichert intern fingerprints von besuchten URLs zur Vermeidung von "crawling loops"
  - Verhalten über `DUPEFILTER_CLASS` in Konfigurationsdatei änderbar

Übung: Linkcounter

- Erstellung eines Scrapy-Spiders, der alle Vorkommen von in Fefes Blog verlinkten Domains zählt:
  - der Spider soll in der Ausgabedatei (beliebiges Format) eine Datenstruktur als Dictionary anlegen:

```
{
  "www.spiegel.de": 1234,
  "www.heise.de": 567,
  ...
}
```

  - eine Unterscheidung von `"www.spiegel.de"` und `"spiegel.de"` muss hier nicht vorgenommen werden (d.h. 2 Einträge sind ok)
  - der Hostname eines Links kann z.B. mithilfe der Funktion `urlparse` aus dem Modul `urllib.parse` bestimmt werden
  - nur die Links aus eigentlichen Inhalten (d.h. kein Impressum, FAQ, [l]-Links o.Ä.) sollen gezählt werden
  - die Einträge sollen absteigend sortiert sein

Fazit

- Datenzugriff u.a. mithilfe von CSS möglich
  - weniger neues zu lernen
- Python Framework
  - Kann ohne extra portiert zu werden auf anderen Systemen ausgeführt werden
- Asynchron
  - nicht einfach in Rahmenprogramm einzubetten
- Spiderinterface sehr einfach
- Viel automatisiert
  - vermeiden von Link-Dubletten
- Keine möglichkeit mit Scrapy direkt javascript zubenutzen
  - Benötigt externe Programme (Splash)

Quellen

[1] https://doc.scrapy.org/en/latest/intro/tutorial.html

[2] https://doc.scrapy.org/en/latest/intro/install.html

[3] https://doc.scrapy.org/en/latest/topics/spiders.html

[4] https://doc.scrapy.org/en/latest/topics/settings.html