Web scraping with the Scrapy Framework

Michał Kołek

What is web scraping?

Difference betweeb scraping and crawling

A web crawler is an Internet bot that systematically browses the World Wide Web and that is typically operated by search engines for the purpose of Web indexing

https://www.webcrawler.com

Robots and sitemaps

```
User-agent: *
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
Disallow: /sdch
Disallow: /groups
Disallow: /index.html?
Disallow: /?
Allow: /?hl=
Disallow: /?hl=*&
Allow: /?hl=*&gws_rd=ssl$
Disallow: /?hl=*&*&gws_rd=ssl
Allow: /?gws_rd=ssl$
Allow: /?pt1=true$
Disallow: /imgres
Disallow: /u/
Disallow: /preferences
Disallow: /setprefs
Disallow: /default
Disallow: /m?
Disallow: /m/
Allow:    /m/finance
Disallow: /wml?
Disallow: /wml/?
Disallow: /wml/search?
Disallow: /xhtml?
Disallow: /xhtml/?
Disallow: /xhtml/search?
Disallow: /xml?
Disallow: /imode?
Disallow: /imode/?
Disallow: /imode/search?
```

https://www.google.com/robots.txt

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = True
```

```
−<sitemapindex>
  −<sitemap>
     <loc>https://www.google.com/gmail/sitemap.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/forms/sitemaps.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/slides/sitemaps.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/sheets/sitemaps.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/drive/sitemap.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/docs/sitemaps.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/get/sitemap.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/flights/sitemap.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/admob/sitemap.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/business/sitemap.xml</loc>
   </sitemap>
  −<sitemap>
     <loc>https://www.google.com/services/sitemap.xml</loc>
   </sitemap>
```

https://www.google.com/sitemap.xml

How does it work?

1. HTTP request

- The web scraper sends HTTP requests to get relevant sites

2. Extracting and parsing website's code

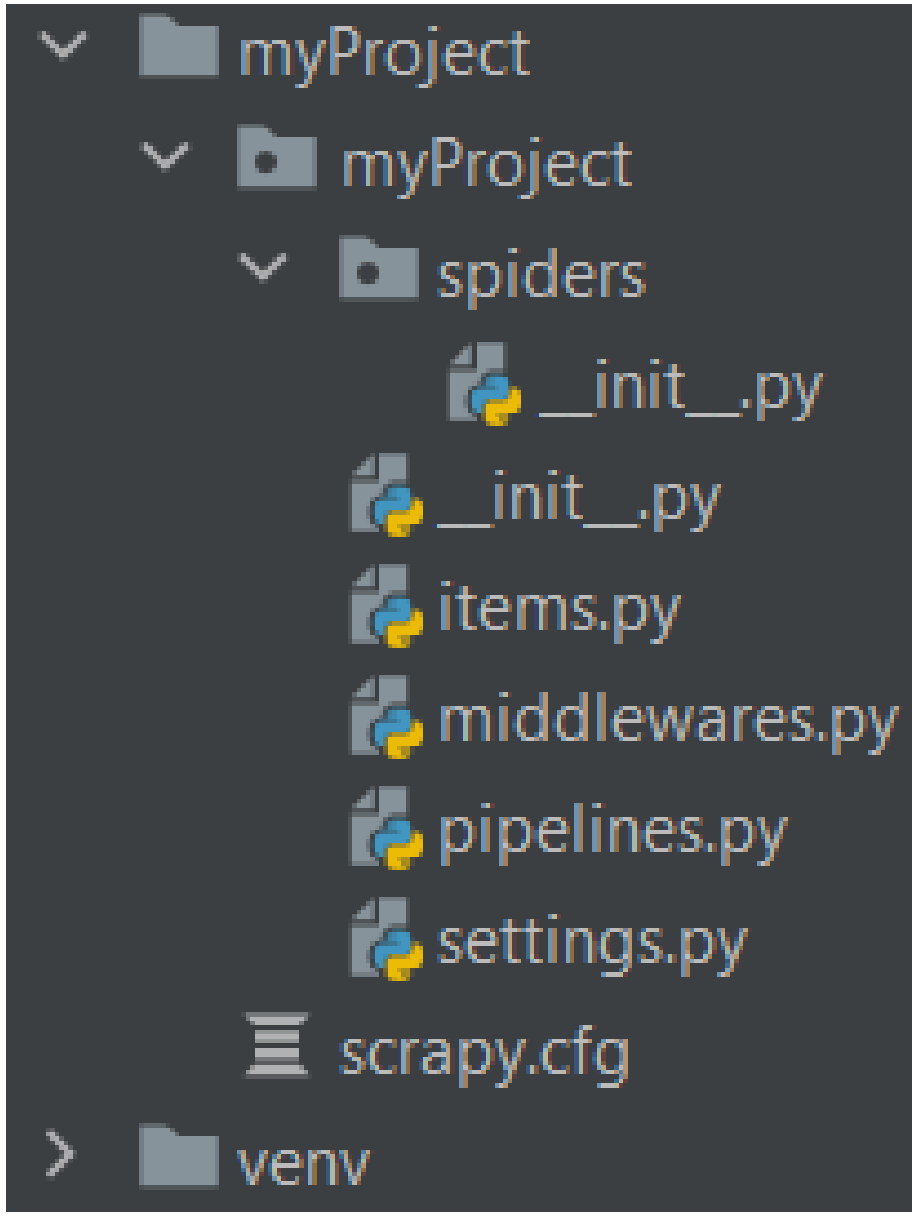
- The scraper looks through HTML or XML code to find predefined elements

3. Saving relevant data

- Formatting data into a structured object (Scrapy supports JSON, JSON lines, CSV, XML)

Project structure

- Spiders go into the spiders directory and have to extend the Spider class
- You can define own Item objects to store data in items.py file
- Objects scraped by the spider go through all pipelines defined in pipelines.py
- You can adjust settings such as pipelines to be used, setting request headers, enabling or disabling cookies, setting max. concurrent request count and many others



Request objects

- Request is (usually) generated by a spider and creates a Response object. Important parameters:
  - **url** (*str*) – URL address of the resource
  - **callback** (*collections.abc.Callable*) – the function to bo called with the response as its' parameter
  - **meta** (*dict*) – additional values which can be passed this way to another function. They can be accessed in the spider from response.meta attribute

```
yield scrapy.Request(url=job_url,
                     meta={
                         'company': company,
                         'position': position,
                         'job_url': job_url,
                     },callback=self.parse_job)
```

```
yield JobOpeningItem(
    job_url=response.meta['job_url'],
    position=response.meta['position'],
    company=response.meta['company'],
    seniority=seniority,
    requirements=requirements,
    nice_to_have=nice_to_have,
    salary=salary,
    location=location,
    posting_time=posting_time,
    additional_info=additional_info
)
```

Response objects

- It is the object created after scrapy makes a Request
- Important attributes:
  - **url** (*str*) – the URL of this response
  - **status** (*int*) – the HTTP status of the response
  - **body** (*bytes*) – the response body
  - **request** (*scrapy.Request*) – represents the Request that generated this response
- Important methods:
  - follow(*url*, *callback=None) – makes a new request to given url*
  - *xpath(selector) – returns objects from response matching the selector*
  - *css(selector) – returns objects matching the selector*
  - *urljoin(url) – joins the responses base url with the url given as argument*

Css and Xpath selectors

Goal	CSS 3	XPath
All Elements	*	//*
All P Elements	p	//p
All Child Elements	p>*	//p/*
Element By ID	#foo	//*[@id='foo']
Element By Class	.foo	//*[contains(@class,'foo')]
Element With Attribute	*[title]	//*[@title]
First Child of All P	p>*:first-child	//p/*[0]
All P with an A child	Not possible	//p[a]
Next Element	p + *	//p/following-sibling::*[0]
Previous Element	Not possible	//p/preceding-sibling::*[0]

```
>>> response.css('title::text').get()
'Amazon.com'
```

Zwraca atrubut text pierwszego znacznika title

```
response.css("a[href*='amazon']").getall()
```

Zwraca wszystkie znaczniki ’a’, które w adresie URL mają frazę amazon

```
>>> response.css('img').xpath('@src').getall()
['https://images-na.ssl-images-amazon.com/captcha/uyvnnjxx/Captcha_fmdeaopkbi.jpg', 'https://fls-na.amazon.com/1/oc-csi/1/OP/requestId=0JQFEKC1H8Z7CV460R52&js=0']
```

Połączenie selektorów xpath i css

```
>>> response.xpath('//div[contains(@class,"a-section a-spacing-extra-large")]//span').get()
'<span class="a-button a-button-primary a-span12">\n                                <span class="a-button-inner">\n                                    <button type="sub
mit" class="a-button-text">Continue shopping</button>\n                                </span>\n                            </span>'
```

Zwraca wszystkie znaczniki ’span’ zawarte w znaczniku ’div’ posiadającym przynajmniej dwie wyznaczone klasy

https://docs.scrapy.org/en/latest/topics/selectors.html

Spiders

- name – name of the spider, used to run the spider
- start_urls – list of URLs to make requests for. You can also override the default start_requests() function:

```
def start_requests(self):
    yield scrapy.Request('http://www.example.com/1.html', self.parse)
```

- parse(self, response) – default method invoked by the spider automatically to parse responses from start_urls
- yield keyword – similar to return keyword, used to store values as a dictionary , Scrapy Item, or to make further requests

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = 'quotes'
    start_urls = [
        'https://quotes.toscrape.com/tag/humor/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'author': quote.xpath('span/small/text()').get(),
                'text': quote.css('span.text::text').get(),
            }

        next_page = response.css('li.next a::attr("href")').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
```

Items

+ Items let you save data in a structured manner

```
class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
    stock = scrapy.Field()
    tags = scrapy.Field()
    last_updated = scrapy.Field(serializer=str)
```

```
>>> product = Product(name='Desktop PC', price=1000)
>>> print(product)
```

```
>>> product['last_updated'] = 'today'
```

```
>>> product['name']
Desktop PC
```

```
yield JobOpeningItem(
    job_url=response.meta['job_url'],
    position=response.meta['position'],
    company=response.meta['company'],
    seniority=seniority,
    requirements=requirements,
    nice_to_have=nice_to_have,
    salary=salary,
    location=location,
    posting_time=posting_time,
    additional_info=additional_info
)
```

Pipelines

+ After an item has been scraped by a spider, it is sent to the Item Pipeline which processes it through several components that are executed sequentially.
+ Typical uses of item pipelines are:
  - cleansing HTML data
  - validating scraped data (checking that the items contain certain fields)
  - checking for duplicates (and dropping them)
  - storing the scraped item in a database

```python
class FormatterPipeline:
    def process_item(self, item, spider):
        adapter = ItemAdapter(item)

        if adapter.get('salary'):
            adapter['salary'] = adapter['salary'].encode().decode()

        if adapter.get('location'):
            for ind, location in enumerate(adapter['location']):
                adapter['location'][ind] = location.encode().decode()
            for ind, location in enumerate(adapter['location']):
                if len(adapter['location'][ind].strip()) < 3:
                    del adapter['location'][ind]

        if adapter.get('posting_time'):
            adapter['posting_time'] = adapter['posting_time'].split(' ')[3]

        if adapter.get('additional_info'):
            adapter['additional_info'] = [info for info in adapter['additional_info'] if len(info) > 3]

        return item
```

Python loops explained

for <variable> in <Iterator>:
    <statements>

```
>>> for i in range(3):
...     print(i)
...
0
1
2
```

```
>>> var = ["var1", "var2", "var3"]
>>> for item in var:
...     print(item)
...
var1
var2
var3
```

```
>>> for char in a:
...     print(char)
...
v
a
r
```

```
>>> dict = {"var1": "val1", "var2": "val2", "var3": "val3"}
>>> for key, value in dict.items():
...     print(key + ": " + value)
...
var1: val1
var2: val2
var3: val3
```

```
>>> vars = ["var1", "var2", "var3"]
>>> for index, value in enumerate(vars):
...     print(index, value)
...
0 var1
1 var2
2 var3
```

```
>>> f = open("vars.txt", "r")
>>> for line in f:
...     print(line)
...
var1

var2
```

Bibliography

+**https://docs.scrapy.org**

+https://github.com/scrapy/quotesbot

+https://docs.python.org/3/library/venv.html

+https://towardsdatascience.com/web-scraping-with-scrapy-practical-understanding-2fbdae337a3b