NOKIA

Crawling the web with Scrapy

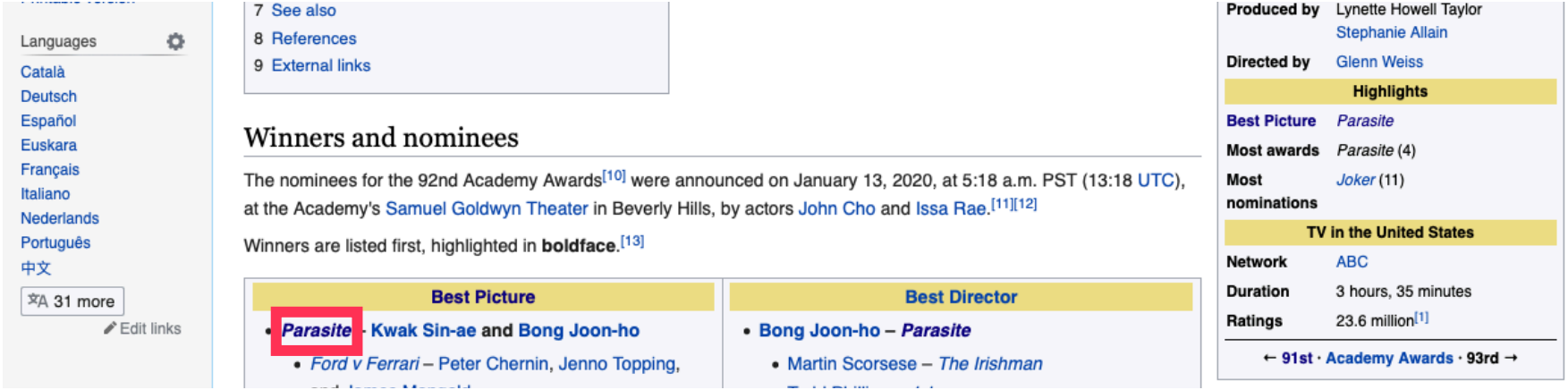
LINCS Python Academy

Quentin Lutz

12-02-2020

Web crawling

- Say you would like to extract information from a website. Doing so can easily prove cumbersome if you proceed with Copy-Paste.



- There are many Python tools that enable web crawling: BeautifulSoup, lxml, **Scrapy**

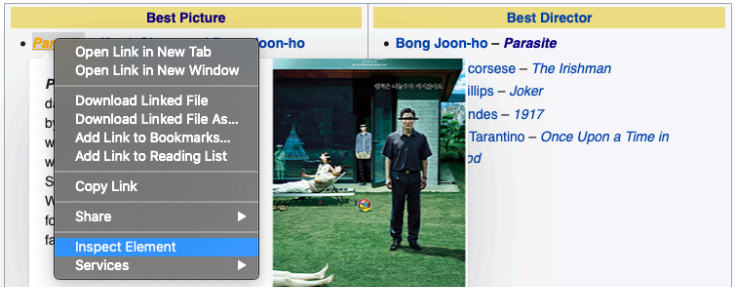
NOKIA

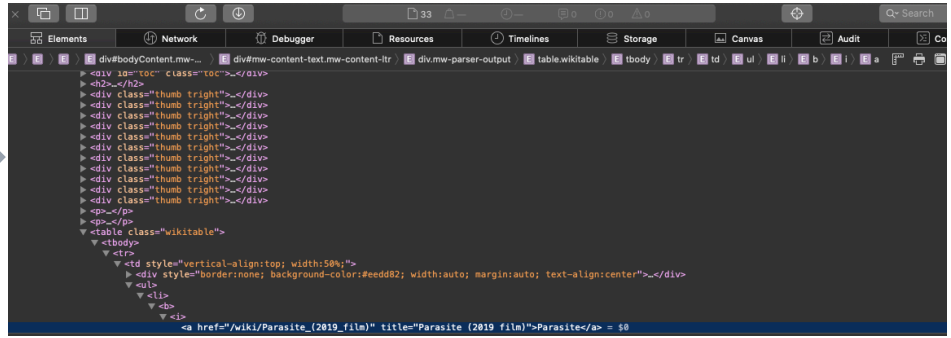
Scrapy

- Is pip installable
- Relies on many libraries to deliver **asynchronous scraping**.
- May be used as a **stand-alone tool** with little to no knowledge of Python…
- … **or** be part of **in a script** as any other library.
- Offers different ways of selecting relevant sections of a webpage (CSS, Xpath, …)

How to address Xpath elements

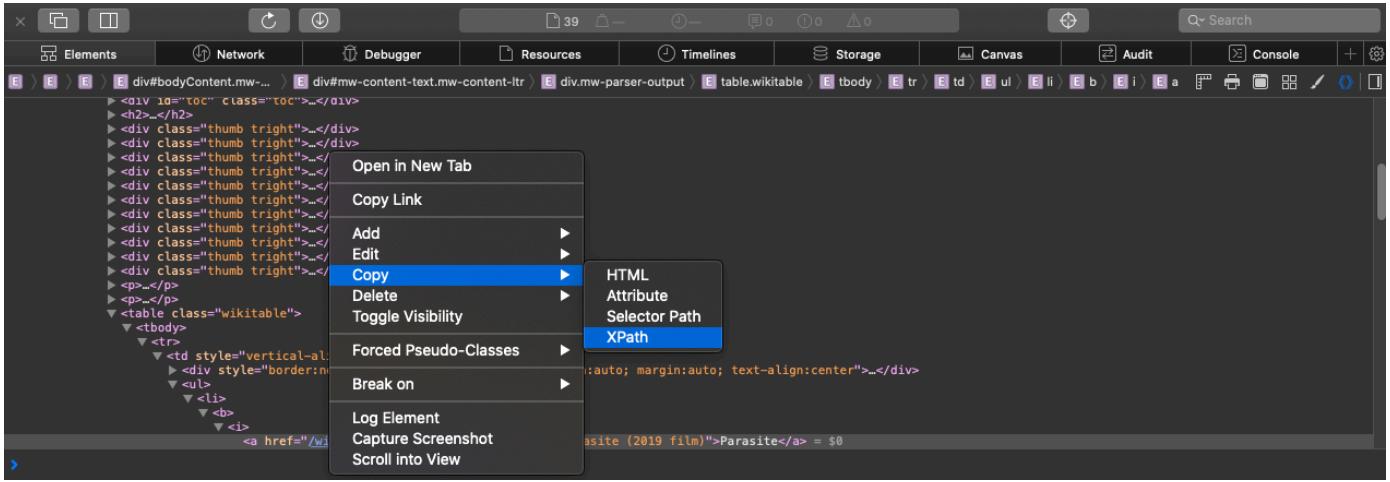
- A webpage is a big container whose elements are itself containers.
- This hierarchical structure make addressing straightforward but the address of one element is often very long.
- The easiest way to find such an address is to use the **Developer tools** embedded in your favorite browser.





How to address Xpath elements

- This gives you the Xpath address of the element you want:




```
//*[@id="mw-content-text"]/div/table[2]/tbody/tr[1]/td[1]/ul/li/b/i/a
```

Scrapy structure

- Scrapy's central element is the **Spider class**.

- To define the how and what of your crawling/scraping, you define a subclass of Spider:

```
films = []


class Aragog(Spider):
    name = "Aragog"
    start_urls = ['https://en.wikipedia.org/wiki/92nd_Academy_Awards']

    def parse(self, response):
        for tr in response.xpath('//*[@id="mw-content-text"]/div/table[2]/tbody/tr[1]/td[1]/ul/li/b/i/a/text()'):
            films.append(tr.extract())


process = CrawlerProcess()
process.crawl(Aragog)
process.start()
```

Previously obtained Xpath

To denote the actual label

NOKIA

Running spiders
Starting from different webpages

```
films = []


class Aragog(Spider):
    name = "Aragog"
    start_urls = ['https://en.wikipedia.org/wiki/%s_Academy_Awards' % ordinal for ordinal in ordinals]

    def parse(self, response):
        for tr in response.xpath('//*[@id="mw-content-text"]/div/table[2]/tbody/tr[1]/td[1]/ul/li/b/i/a/text()'):
            films.append(tr.extract())


process = CrawlerProcess()
process.crawl(Aragog)
process.start()
```

Running spiders
Getting multiple elements from one page

```
films = []


class Aragog(Spider):
    name = "Aragog"
    start_urls = ['https://en.wikipedia.org/wiki/92nd_Academy_Awards']

    def parse(self, response):
        for el in response.xpath('//*[@id="mw-content-text"]/div/table[2]/tbody/tr[1]/td[1]/ul/li/ul/li'):
            film = el.xpath('./i/a/text()').extract_first()
            films.append(film)


process = CrawlerProcess()
process.crawl(Aragog)
process.start()
```

Running spiders
Using hyperlinks and scraping pictures

```
films = []


class Aragog(Spider):
    name = "Aragog"
    start_urls = ['https://en.wikipedia.org/wiki/%s_Academy_Awards' % ordinal for ordinal in ordinals]

    custom_settings = {
        'ITEM_PIPELINES': {'scrapy.pipelines.files.FilesPipeline': 1},
        'FILES_STORE': '/some/path'
    }

    def parse(self, response):
        for el in response.xpath('//*[@id="mw-content-text"]/div/table[2]/tbody/tr[1]/td[1]/ul/li/b/i/a'):
            title = el.xpath('./text()').extract_first()
            films.append(title)
            rel_url = el.xpath('@href').extract_first()
            abs_url = response.urljoin(rel_url)
            yield Request(url=abs_url, callback=self.parse_movie)

    def parse_movie(self, response):
        for img in response.xpath('//*[@id="mw-content-text"]/div/table[1]/tbody/tr[2]/td/a/img'):
            rel_url = img.attrib['src']
            yield {'file_urls': [response.urljoin(rel_url)]}


process = CrawlerProcess()
process.crawl(Aragog)
process.start()
```

Just passing the image URL with a specific field name

NOKIA

Running spiders
Naming files (custom pipeline)

- Unfortunately, Scrapy names the pictures based on the SHA1 hash of their URL.

- Create a custom pipeline to use metadata:

```
class MyFilesPipeline(FilesPipeline):
    def file_path(self, request, response=None, info=None):
        original_path = super(MyFilesPipeline, self).file_path(request, response=None, info=None)
        extension = original_path.split('.')[-1]
        return request.meta.get('filename', '') + "." + extension

    def get_media_requests(self, item, info):
        file_url = item['file_urls'][0]
        meta = {'filename': item['name']}
        yield Request(url=file_url, meta=meta)
```


NOKIA

Running spiders
Naming files (passing metadata)

- Then pass metadata when using links:

```
class Aragog(Spider):
    name = "Aragog"
    start_urls = ['https://en.wikipedia.org/wiki/%s_Academy_Awards' % ordinal for ordinal in ordinals]

    custom_settings = {
        'ITEM_PIPELINES': {'__main__.MyFilesPipeline': 1},
        'FILES_STORE': '/Users/qlutz/Downloads/posters'
    }

    def parse(self, response):
        for el in response.xpath('//*[@id="mw-content-text"]/div/table[2]/tbody/tr[1]/td[1]/ul/li/b/i/a'):
            title = el.xpath('./text()').extract_first()
            films.append(title)
            rel_url = el.xpath('@href').extract_first()
            abs_url = response.urljoin(rel_url)
            yield Request(url=abs_url, callback=self.parse_movie, meta={'Title': title})

    def parse_movie(self, response):
        for img in response.xpath('//*[@id="mw-content-text"]/div/table[1]/tbody/tr[2]/td/a/img'):
            rel_url = img.attrib['src']
            yield {'file_urls': [response.urljoin(rel_url)], 'name': response.meta.get('Title')}
```

```
12 Years a Slave.jpg
Argo.jpg
Birdman or (The Unexpected Virtue of Ignorance).png
Green Book.png
Moonlight.png
Parasite.png
Spotlight.jpg
The Artist.jpg
The King's Speech.jpg
The Shape of Water.png
```

Pass the title when making the request

Get the data back from the dict

Command line commands

- Scrapy has its own project format with separate files for the configuration and the spiders.

```
scrapy startproject myproject
```

```
scrapy genspider myspider https://that_site_id_like_to_scrape.com
```

- Spider parsing functions are generators.

- CLI arguments make it possible to choose many things including output files

```
scrapy crawl myspider -o myfile.csv
```

Command line commands

```
class SynapsesSpider(scrapy.Spider):
    name = 'synapses'
    allowed_domains = ['synapses.telecom-paristech.fr']
    start_urls = ['https://synapses.telecom-paristech.fr/catalogue/2019-2020/diplome/1991/ING-diplome-d-ingenieur']

    def parse(self, response):
        libs = response.xpath('//td[@class="libelle"]')
        for lib in libs:
            cours = lib.xpath('span[@class="label label-ue"]/text()').extract_first()
            rel_url = lib.xpath('a/@href').extract_first()
            abs_url = response.urljoin(rel_url)
            yield scrapy.Request(abs_url, callback=self.parse_course,meta={'Cours':cours})

    def parse_course(self,response):
        humans = response.xpath('//a[@class="annuaire-modal-link"]/text()').extract()
        for human in humans:
            yield{'Cours':response.meta.get('Cours'),'Human':human}
```

myfile.csv

Cours	Human
ACCQ201	Hugues RANDRIAMBOLOLONA
ACCQ201	Carole PELTIER
ALL-P-022	Karin ILLNER-TOLEDAN
ALL-P-022	Emma SCHWABEDISSEN
ALL-P-022	Marie BAQUERO
ALL-P-022	Karin ILLNER-TOLEDAN
ALL-P-022	Emma SCHWABEDISSEN
ALL-B2.1	Emma SCHWABEDISSEN
ALL-B2.1	Karin ILLNER-TOLEDAN
...	...

Scrapy shortcomings

- Scrapy can be used on most static pages but is useless on dynamic pages (like train/plane booking websites).

- Scrapy is designed to work from the command line rather than a Python script. That shows a lot in the documentation (=> do not hesitate to check the source code for workarounds).

References

- https://scrapy.org
- https://python.gotrained.com/scrapy-tutorial-web-scraping-craigslist/#Scrapy_Tutorial_Getting_Started
- https://docs.scrapy.org/en/latest/index.html