Introduction to Scrapy

USI - Università della Svizzera Italiana, Information Retrieval

{fabio.crestani, monica.landoni, federico.ravenda, enrico.verdolotti}@usi.ch

TA: Enrico Verdolotti

Outline

- What is web crawling?
  - Bot/Crawler/Spider
  - Caveats
- Scrapy Demo
  - Installation
  - Scrapy shell
  - Creating a project
  - Our first crawler
- Your turn

Crawling? Scraping?

A **bot**, short for robot, is a generic term for any software program that performs automated tasks on the internet.

A **crawler** is a bot that can navigate autonomously a web site

The focus is on the ability to follow hyperlinks and «move».

The term «**spider**» is often used interchangeably with «crawler». In the context of the internet, a spider is essentially the same as a web crawler.

So, in the web-scraping context, *spiders are bots that crawls the web.*

Caveats - Be careful!

- A spider can send a lot of Requests to a website.
  - Websites don’t like to be overloaded
- Too many requests > overloading the server > ban of your IP Address for future crawling.
- Precaution steps:
  - Respect what states robots.txt (check with: [root-domain-name]/robots.txt)
  - Limit your request frequency
  - Identifiy yourself (AKA User Agent)
  - Disable cookies
  - Use proxies
  - ...

- Documentation: https://docs.scrapy.org/en/latest/topics/practices.html#avoiding-getting-banned

Scrapy

- What is Scrapy?
  - Application framework for crawling web sites and extracting structured data.
  - Written in Python
  - https://docs.scrapy.org/en/latest/

- It has a large variety of features, including:
  - Cookies and session handling
  - HTTP features like: compression, authentication, caching
  - User-agent spoofing (i.e. you can pretend to be a regular browser)
  - Robots.txt handling (default: OBEY to the rules)
  - Crawl depth restriction
  - …

Installing Scrapy

- Scrapy installation guide: https://docs.scrapy.org/en/latest/intro/install.html

- Recomended: with Anaconda (https://www.anaconda.com/)
  - Install Anaconda (or Miniconda)
  - Create an environment (e.g. command: `% conda create –n scrapyenv`
  - [redacted] `% conda activate scrapyenv`
  - [redacted] `% conda install -c conda-forge scrapy`

- [redacted]

Scrapy shell

- An interactive shell console that allow you to:
  - Create your spider projects
  - Run your spiders
  - Debug spider behaviour
  - Test XPath or CSS expressions
  - …
- Will be available once Scrapy is installed

```
% scrapy shell "https://quotes.toscrape.com/"
```

```
% scrapy parse --spider=amzpider -c parse -d 4 'https://www.amazon.it'
```

```
% scrapy crawl quotes -o quotes.jsonl
```

```
% scrapy startproject tutorial
```

Creating a project

- Open terminal and go to the desired folder that will contain your crawler
- Create a project with: `% scrapy startproject <my-project-name>`
- `% scrapy startproject tutorial`

```
tutorial/
    scrapy.cfg            # deploy configuration file

    tutorial/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

Creating a project

- Create a file [any-spider-script-name].py within the spiders folder.

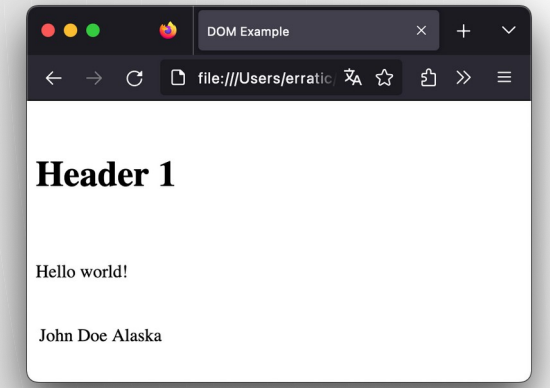
```
tutorial/
    scrapy.cfg            # deploy configuration file

    tutorial/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

quotes_spider.py

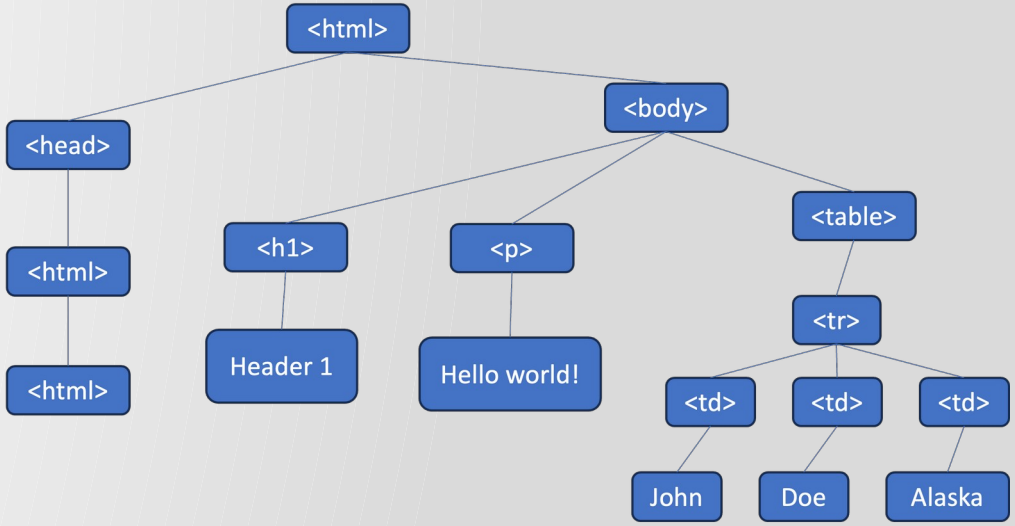
```python
import scrapy

class QuoteSpider(scrapy.Spider):

    name = "quotes"    # Your spider name. Each instance of a QuoteSpider will share the same name.

    start_urls = ["https://quotes.toscrape.com/page/1/",     # URLs list to start crawling.
                  "https://quotes.toscrape.com/page/2/"]
```

Selectors – Document Object Model

- A spider need to locate elements of interest inside a web page DOM
  - The Response object will contain also the entire HTML of the requested page

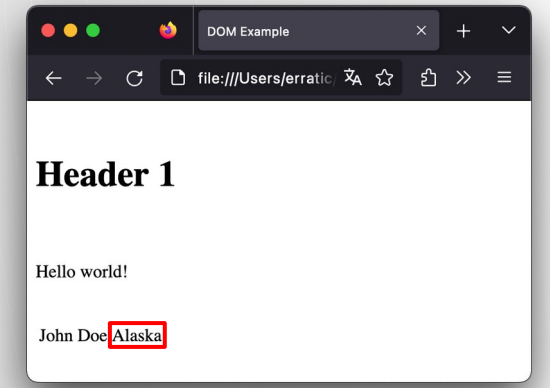



```
<html>
   <head>
       <title>DOM
Example</title>
   <head>
   <body>
      <h1>Header 1</h1>
   <p>Hello world!</p>
   <table>
      <tr class='people'>
         <td>John</td>
         <td>Doe</td>
         <td>Alaska</td>
      </tr>
    </table>
  </body>
</html>
```

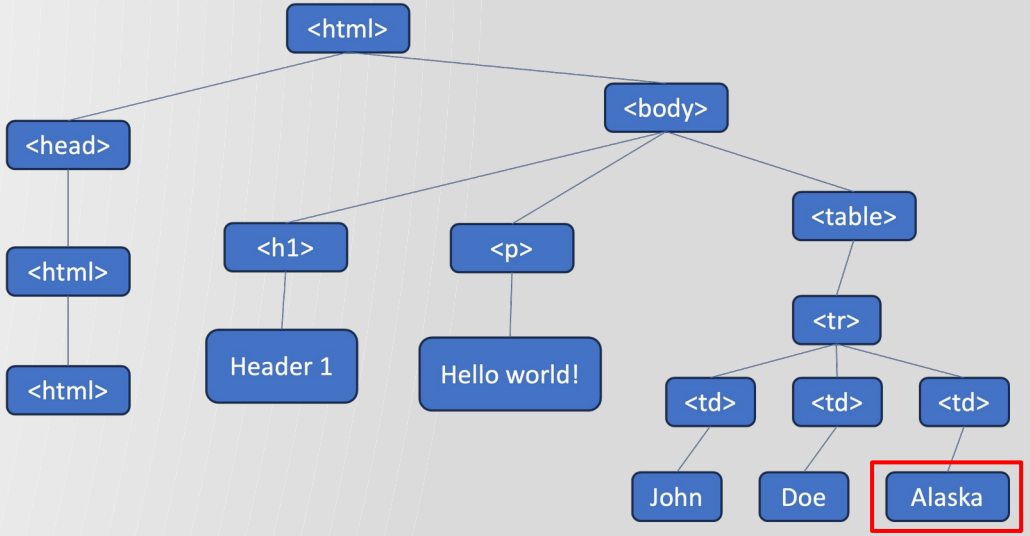


Selectors - XPath

- Two options: CSS selectors or XPath expressions
  - CSS selectors are slightly simpler than XPath expressions
  - XPath expressions are more powerful
- XPath looks like to the regular Windows or Unix folder path expression (e.g. C:\\user\desktop\...)
- Plenty of tutorials available: https://www.w3schools.com/xml/xpath_intro.asp
- For example: `/html/body/table/tr/td[3]/text()` will select 'Alaska' as text information from HTML
- But also: `//tr[@class='people']/td[3]/text()` will do the job

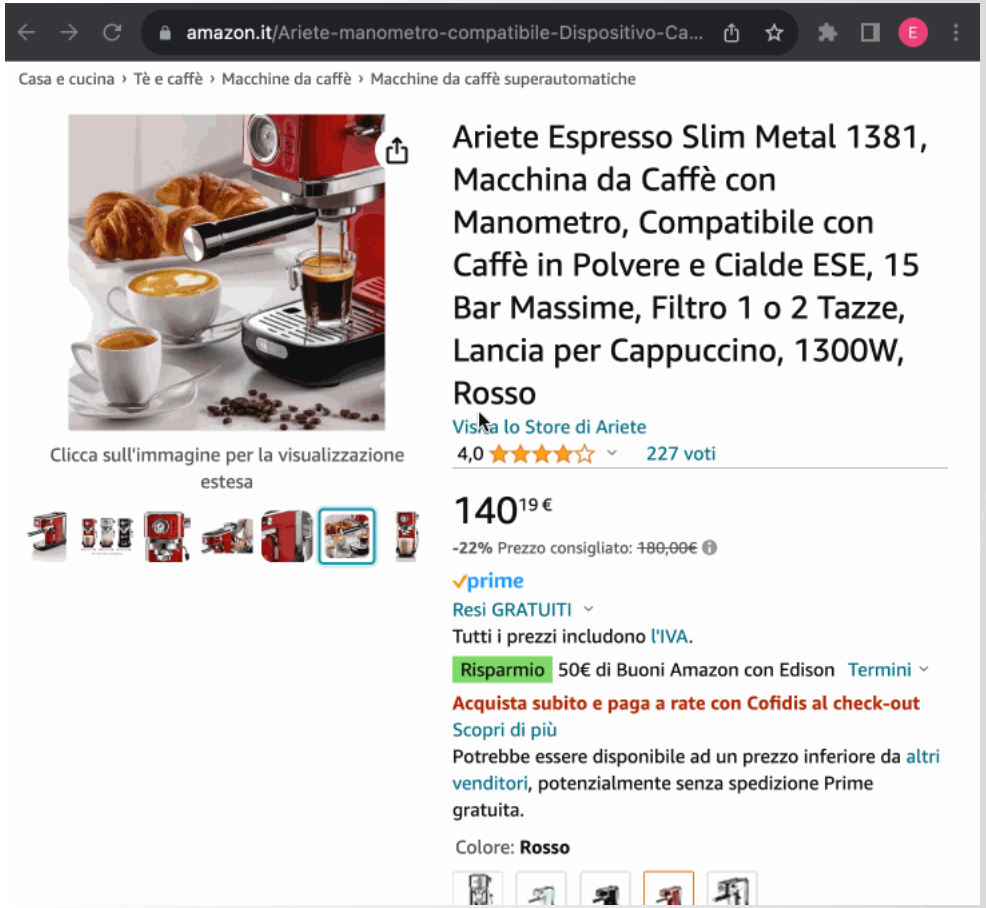



```
<html>
   <head>
      <title>DOM
Example</title>
   <head>
   <body>
      <h1>Header 1</h1>
   <p>Hello world!</p>
   <table>
      <tr class='people'>
         <td>John</td>
         <td>Doe</td>
         <td>Alaska</td>
      </tr>
    </table>
  </body>
</html>
```



Selectors - Tricks

- A quick way to get your selectors



Our first crawler

- Target web site: https://quotes.toscrape.com

```
import scrapy


class QuoteSpider(scrapy.Spider):

    name = "quotes"    # Your spider name. Each instance of a QuoteSpider will share the same name.


    start_urls = ["https://quotes.toscrape.com/page/1/",     # URLs list to start crawling.
                  "https://quotes.toscrape.com/page/2/"]


    def parse(self, response):    # Funcion called every crawled web page. The response parameter will contain the web site response.

        for quote in response.xpath("//div[@class='quote']"):    # For each quote element in the current page...

            text = quote.xpath(".//span[@class='text']/text()").get()                    # Extract the quote text as a string.
            author = quote.xpath(".//small[@class='author']/text()").get()         # Extract the author name as a string.
            tags = quote.xpath(".//div[@class='tags']/a[@class='tag']/text()").getall()  # Extract the quote tags as a list of strings.

            yield {'text': text, 'author': author, 'tags': tags}    # Return extracted data as a Python dict.

        next_page = response.xpath("//li[@class='next']/a/@href").get()    # Extract next page link as a string.

        if next_page:    # If next page is not None, that is, if we have a next page to visit...

            yield response.follow(next_page, callback=self.parse)    # Follow the next page link. Return the next page response object.
```

Scrapy Shell - Run a Spider

- Place your spider code in: `tutorial/tutorial/spiders`
- From tutorial **root folder** run: `% scrapy crawl quotes -o results.jsonl`
  - [redacted]
- [redacted]

Your Turn

- Try to use *scrapy shell* to analyze the response from the website you want to scrape.
- Example command: `% scrapy shell "https://quotes.toscrape.com/page/1/"`

```
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x1074c6690>
[s]   item       {}
[s]   request    <GET https://quotes.toscrape.com/page/1/>
[s]   response   <200 https://quotes.toscrape.com/page/1/>
[s]   settings   <scrapy.settings.Settings object at 0x1062bd150>
[s]   spider     <DefaultSpider 'default' at 0x11022de10>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are followed)
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser
2023-10-22 23:47:02 [asyncio] DEBUG: Using selector: KqueueSelector
In [1]:
```