UE 803 - Data Science

Session 2: Web scraping

Yannick Parmentier - Université de Lorraine / LORIA

Introduction

Web scraping

> Automatically extracting data from websites.

Source: Wikipedia

What is data ?

- audio, image, video, articles, blogs, sensor values, etc.

  → various *data types* (here we focus on **textual data**)

- text files, spreadsheets (e.g. weather forecast), binary files (e.g. word-processing documents), pre-processed data (e.g. treebanks), etc.

  → *(un)structured data* (here we consider **raw data**)

- freely available data (e.g. songs from public domain) *vs* protected data (e.g. song from a famous artist)

  → *licensed* data (various licenses, to be considered on a case by case basis, here we proritize **open licenses** such as Creative Commons)

Workflow

Setting up web scrapers can be done in a few steps:

(0. **Target** valuable online data *)

1. **Download** corresponding webpages
2. **Parse** these webpages to extract useful parts
3. **Format and store** the extracted data for further processing

Workflow (continued)

- Note:
  - All this *can be done programmatically* with most major programming languages.
  - Steps 1. and 2. *can be made simpler* if the data host includes a *webservice*
(see next class)

Outline

1. Recall Internet basics (HTTP / HTML)
2. Downloading HTML documents (introducing requests)
3. Parsing XML documents (introducing BeautifulSoup4)
4. Case study: extracting quotes
5. The automatic way (introducing Scrapy)

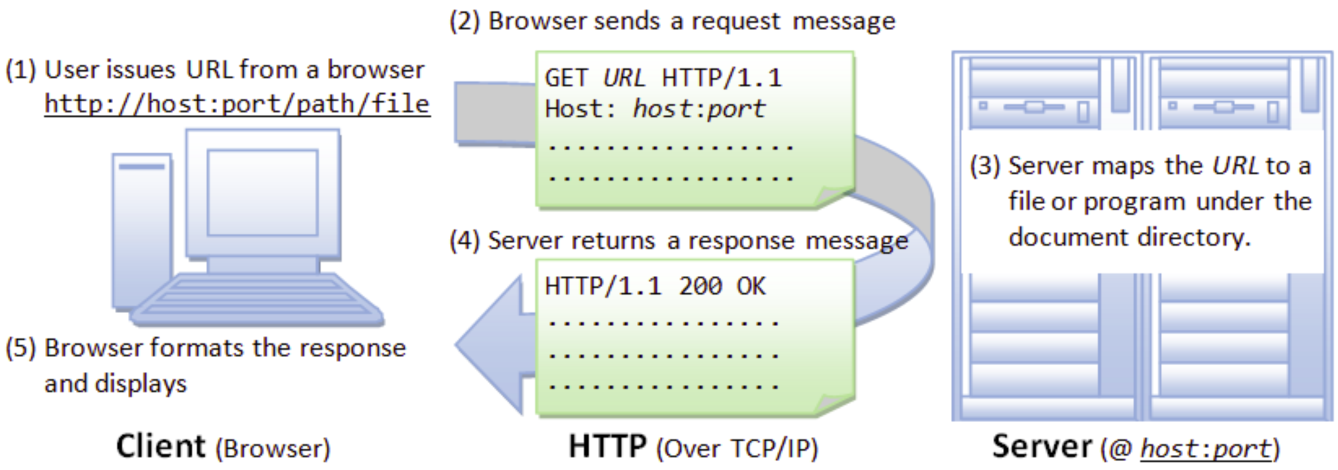
1. Internet basics

Client/Server architecture

- **Client**: program *requesting* a service (e.g. a *web-browser* such as Firefox requesting access to some file)

- **Server**: program *providing* a service (e.g. a *web-server* such as Apache providing access to some file)

- Communication between these two programs relies on a **communication protocol** (here Hyper Text Transfer Protocol - `http`)


(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
....
....
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
....
....
....
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

(Picture from F. Boudin's course)

HTTP queries

- Client queries (and server responses) are **lists of bytes** containing a **header** and a **body**

- Main query **types** (*aka* commands):

  - GET → request data from a specified resource (URL)

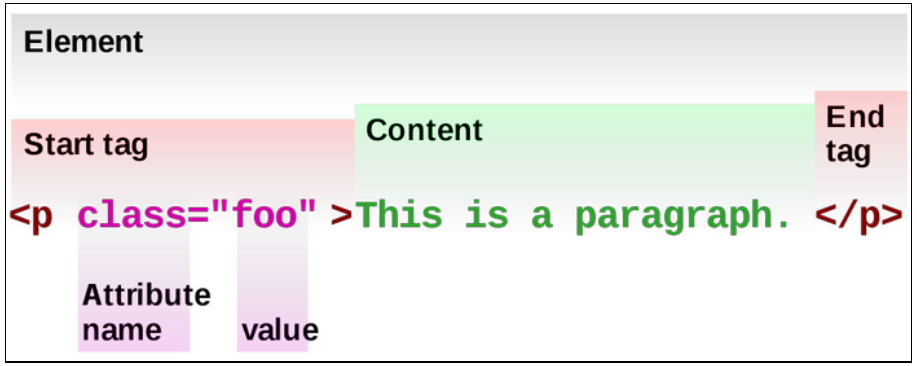
```
GET /test/demo_form.php?name1=value1&name2=value2 HTTP/1.1
Host: w3schools.com
```

  - POST → send data to a server to update a resource

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

HTML

- Web pages are written in **HyperText Markup Language** (HTML)
- HTML is a **description language**
  → how information *should be* displayed
- HTML is **interpreted** (e.g. by a web-browser)
  → similar to LaTeX, python or pdf
- HTML is based on (embedded) **elements** (delimited by opening and closing **tags**, equipped with **attributes**):



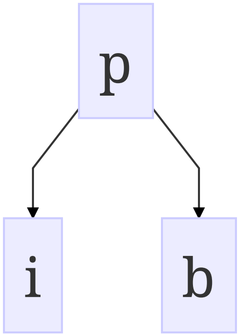
```
<p class="foo" >This is a paragraph. </p>
```

(list of HTML elements 🔗 and attributes🔗)

HTML (continued)

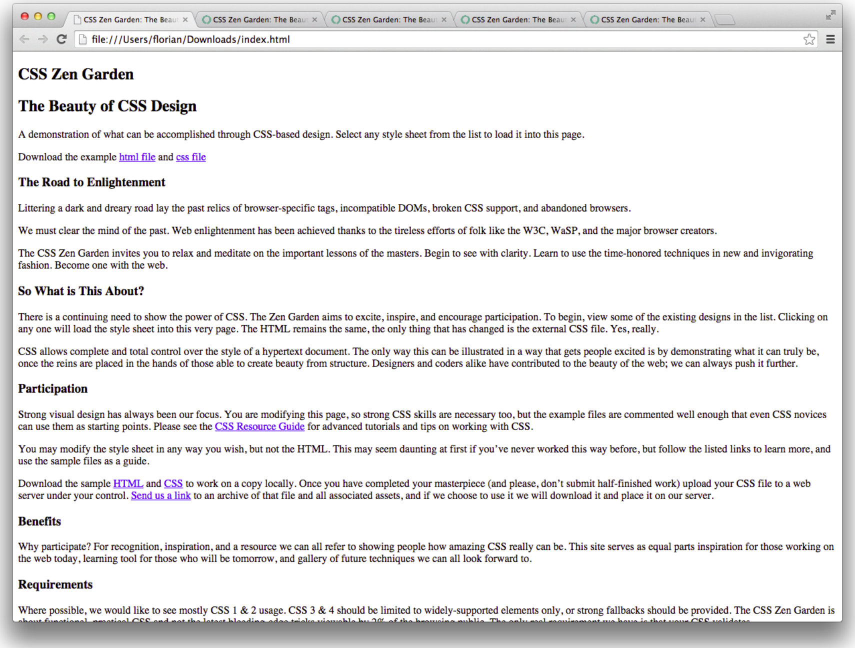
- HTML element *embeddings* follow a **tree structure**:

```
<p>
  <i> ... </i>
  <b> ... </b>
</p>
```



CSS

- Webpages can be associated with **cascading style sheets** (CSS), which contain **layout definitions**


CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
file:///Users/florian/Downloads/index.html

CSS Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.

Download the example html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, broken CSS support, and abandoned browsers.

We must clear the mind of the past. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP, and the major browser creators.

The CSS Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This About?

There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. Designers and coders alike have contributed to the beauty of the web; we can always push it further.

Participation

Strong visual design has always been our focus. You are modifying this page, so strong CSS skills are necessary too, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the HTML. This may seem daunting at first if you've never worked this way before, but follow the listed links to learn more, and use the sample files as a guide.

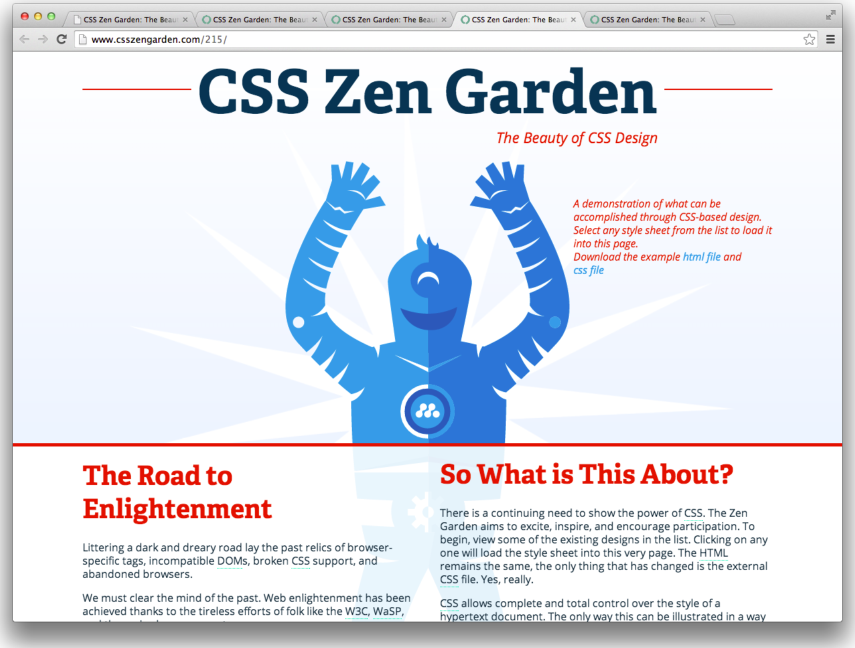
Download the sample HTML and CSS to work on a copy locally. Once you have completed your masterpiece (and please, don't submit half-finished work) upload your CSS file to a web server under your control. Send us a link to an archive of that file and all associated assets, and if we choose to use it we will download it and place it on our server.

Benefits

Why participate? For recognition, inspiration, and a resource we can all refer to showing people how amazing CSS really can be. This site serves as equal parts inspiration for those working on the web today, learning tool for those who will be tomorrow, and gallery of future techniques we can all look forward to.

Requirements

Where possible, we would like to see mostly CSS 1 & 2 usage. CSS 3 & 4 should be limited to widely-supported elements only, or strong fallbacks should be provided. The CSS Zen Garden is about functional, practical CSS and not the latest bleeding-edge tricks viewable by 2% of the browsing public. The only real requirement we have is that your CSS validates.


CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
CSS Zen Garden: The Beaut
www.csszengarden.com/215/

CSS Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page. Download the example html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, broken CSS support, and abandoned browsers.

We must clear the mind of the past. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP,

So What is This About?

There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way

- Content is thus separated from layout

CSS (continued)

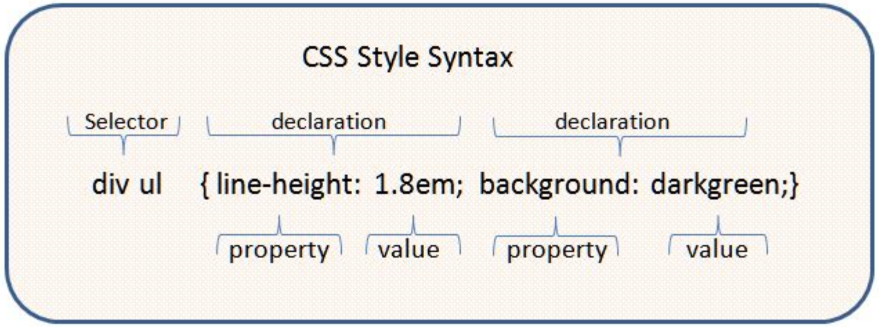
- CSS layout definition is declared in the webpage's header:

```
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Example file</title>

    <link rel="stylesheet" href="/215/215.css">
```

- CSS: list of **rules**

- A rule is made of a **selector** and a list of **declarations** (attribute-values)



CSS Style Syntax

```
div ul  { line-height:  1.8em;  background:  darkgreen;}
```

CSS selectors

- **Elements** (`*`, `div`, ...) or **attributes** (e.g. `[width]`)

```
<div width="50%">
...</div>
```

- **Dominance** constraints on elements (such as `div ul` or `div > ul`)

```
<div><ul>
...
</ul></div>
```

- **Precedence** constraints on elements (such as `div ~ p` or `div + p`)

```
<div>...
</div>
<p>...</p>
```

```
<ul>...</ul>
```

- **Union** of elements (such as `div, ul`)

```
<div>...</div>
```

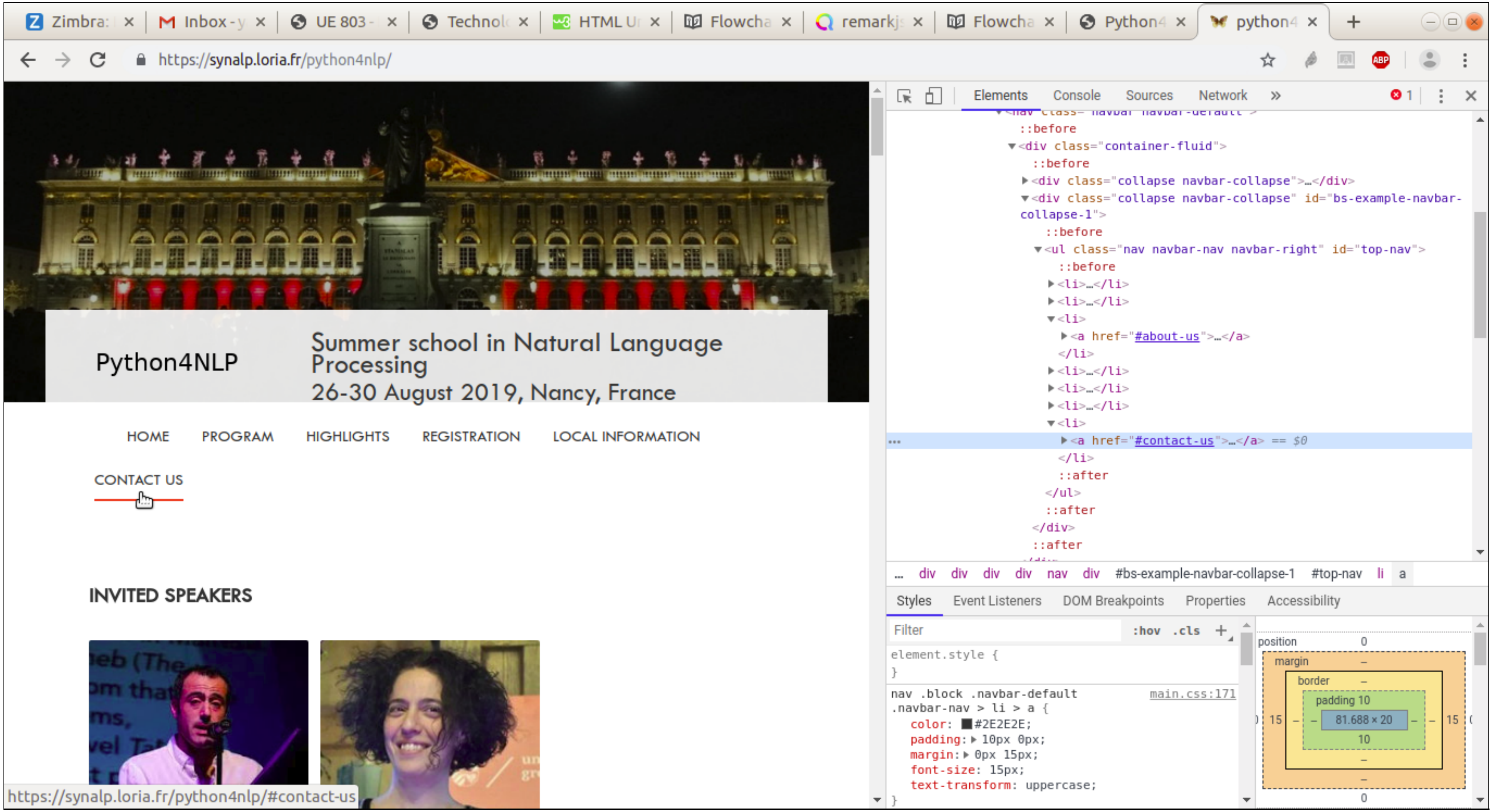
CSS selectors (continued)

- **Class** (referring to *groups* of elements, such as `.intro` or `p.intro`)
- **Identifier** (referring to *unique* elements, such as `#intro`)

```
<p class="intro">
  ...
  ...
</p>
```

```
<p id="intro">
  ...
  ...
</p>
```

To inspect a webpage structure : hit the F12 key



https://synalp.loria.fr/python4nlp/

Python4NLP

Summer school in Natural Language Processing
26-30 August 2019, Nancy, France

HOME PROGRAM HIGHLIGHTS REGISTRATION LOCAL INFORMATION CONTACT US

INVITED SPEAKERS

https://synalp.loria.fr/python4nlp/#contact-us

```
▼<nav class="navbar navbar-default">
   ::before
  ▼<div class="container-fluid">
     ::before
    ▶<div class="collapse navbar-collapse">…</div>
    ▼<div class="collapse navbar-collapse" id="bs-example-navbar-
     collapse-1">
       ::before
      ▼<ul class="nav navbar-nav navbar-right" id="top-nav">
         ::before
        ▶<li>…</li>
        ▶<li>…</li>
        ▼<li>
          ▶<a href="#about-us">…</a>
         </li>
        ▶<li>…</li>
        ▶<li>…</li>
        ▶<li>…</li>
        ▼<li>
...       ▶<a href="#contact-us">…</a> == $0
         </li>
         ::after
       </ul>
       ::after
     </div>
     ::after
```

… div div div div nav div #bs-example-navbar-collapse-1 #top-nav li a

```
element.style {
}
nav .block .navbar-default                main.css:171
.navbar-nav > li > a {
   color: ■#2E2E2E;
   padding: ▶ 10px 0px;
   margin: ▶ 0px 15px;
   font-size: 15px;
   text-transform: uppercase;
}
```

2. Retrieving HTML pages

Introducing requests

Retrieving HTML pages

- Can be done using the `requests` python library:

```
>>> import requests
>>> ua = {'User-agent': 'Mozilla/5.0'}
>>> page=requests.get("http://synalp.loria.fr/index.html", headers=ua)
>>> print(page.status_code)
200
>>> print(page.content)
b'<!DOCTYPE html>\n<html>\n<head>\n  <meta charset="UTF-8">\n
...
```

- Returns an **object** of type `Response` (see doc. 🔗):

```
>>> type(page)
<class 'requests.models.Response'>
```

Retrieving HTML pages (continued)

- NB: The content of the retrieved document (`content` attribute of the `Response` object) is a **binary string**
- If needed, it can be *decoded* (interpreted with a given encoding) using Python's `decode` method (see doc. 🔗):

```
>>> type(page.content)
<class 'bytes'>
>>> print(page.content.decode("utf8"))
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
...
```

3. Parsing an XML file

Introducing BeautifulSoup4

Parsing an XML file

- **Parsing** can be done with the `BeautifulSoup4` library (see documentation 🔗):

```
>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(page.content, 'html.parser')
>>> print(soup.prettify())
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <meta content="width=device-width, initial-scale=1" name="viewport"/>
...
```

- Parsing an XML document amounts to **storing** its content in a **structured** way (i.e. within a dedicated `BeautifulSoup` object) to make **data access** easier

Parsing an XML file (continued)

- One can **search** through an XML document (that is explore the underlying tree structure, *aka* Document Object Model / DOM) using `BeautifulSoup`'s **dedicated methods** (see documentation 🔗):

```
>>> for para in soup.find_all('p'):
...     print(para.get_text())
...
```

```
Welcome to our website !
The Synalp team is located in Nancy, France and is part of the ...
```

- `find_all(c)` can be used to **extract** the *list* of all **XML elements** (i.e. list of XML **sub-trees**) satisfying a given **constraint** `c`

Searching through an XML document

- Search constraints (**filters**) can be applied:
  - either **on HTML elements**
    (via `find`, `find_all`, `find_parent`, etc.)
  - or **on CSS selectors**
    (via `select`)

- `find`, `find_next`, etc. retrieve the **first element** satisfying some constraint(s), while

- `find_all`, `find_parents`, etc. retrieve the **list of all elements** satisfying it(them)

Searching through an XML document (continued)

- `find` methods accept constraints on:

Searching through an XML document (continued)

- `find` methods accept constraints on:
  - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`

Searching through an XML document (continued)

- `find` methods accept constraints on:
  - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
  - **attribute name**, e.g. `soup.find_all(href=True)`

Searching through an XML document (continued)

- `find` methods accept constraints on:
    - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
    - **attribute name**, e.g. `soup.find_all(href=True)`
    - **attribute name and value**, e.g. `soup.find_all(id='toto')`

Searching through an XML document (continued)

- `find` methods accept constraints on:
  - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
  - **attribute name**, e.g. `soup.find_all(href=True)`
  - **attribute name and value**, e.g. `soup.find_all(id='toto')`
  - **element content**, e.g. `soup.find_all(string='toto')`

Searching through an XML document (continued)

- `find` methods accept constraints on:
  - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
  - **attribute name**, e.g. `soup.find_all(href=True)`
  - **attribute name and value**, e.g. `soup.find_all(id='toto')`
  - **element content**, e.g. `soup.find_all(string='toto')`

- Note that `find` input parameters can be:

Searching through an XML document (continued)

- `find` methods accept constraints on:
    - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
    - **attribute name**, e.g. `soup.find_all(href=True)`
    - **attribute name and value**, e.g. `soup.find_all(id='toto')`
    - **element content**, e.g. `soup.find_all(string='toto')`

- Note that `find` input parameters can be:
    - **list of values**, e.g. `soup.find_all(['a','b'])`

Searching through an XML document (continued)

- `find` methods accept constraints on:
    - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
    - **attribute name**, e.g. `soup.find_all(href=True)`
    - **attribute name and value**, e.g. `soup.find_all(id='toto')`
    - **element content**, e.g. `soup.find_all(string='toto')`

- Note that `find` input parameters can be:
    - **list of values**, e.g. `soup.find_all(['a','b'])`
    - **regular expressions**, e.g. `soup.find_all(href=re.compile('^https'))`

Searching through an XML document (continued)

- `find` methods accept constraints on:
  - **element name**, e.g. `soup.find_all(name='a')` or `soup.find_all('a')`
  - **attribute name**, e.g. `soup.find_all(href=True)`
  - **attribute name and value**, e.g. `soup.find_all(id='toto')`
  - **element content**, e.g. `soup.find_all(string='toto')`

- Note that `find` input parameters can be:
  - **list of values**, e.g. `soup.find_all(['a','b'])`
  - **regular expressions**, e.g. `soup.find_all(href=re.compile('^https'))`
  - **functions**, e.g. `soup.find_all(lambda x: x.has_attr('class'))`

Searching through an XML document (continued)

- To return a **limited number of results**: `soup.find_all('a', limit=3)` and/or to be **non recursive**: `soup.find_all('a', recursive=False)`

- To access element **contents**:

```
>>> [x.get_text() for x in soup.find_all('a')] #[' Synalp ', ...]
```

- Information can eventually be stored in e.g. **dictionaries** and saved in **files**:

```
>>> data = { 'links' : [x.get_text() for x in soup.find_all('a')] }
>>> with open('links.txt', 'w') as f:
...  print(data['links'], file=f)
```

Search examples

```
>>> soup.find('footer') #<footer class="uk-clearfix" ...

>>> soup.find('div', class_='uk-panel') # <div class="uk-panel ...

>>> soup.find('div', attrs={'class':'uk-panel'})

>>> soup.find_all('a') #[<a class="uk-navbar-brand" ...]

>>> soup.find_all(lambda x: x.has_attr('id'), recursive=False)

>>> soup.select('div p') #[<p class="uk-article-meta"></p>...]

>>> soup.select("p:nth-of-type(3)") #[<p>The Synalp team is ...]

>>> soup.select("div ~ .uk-panel") #[<div class="uk-panel"> <div ...]
```

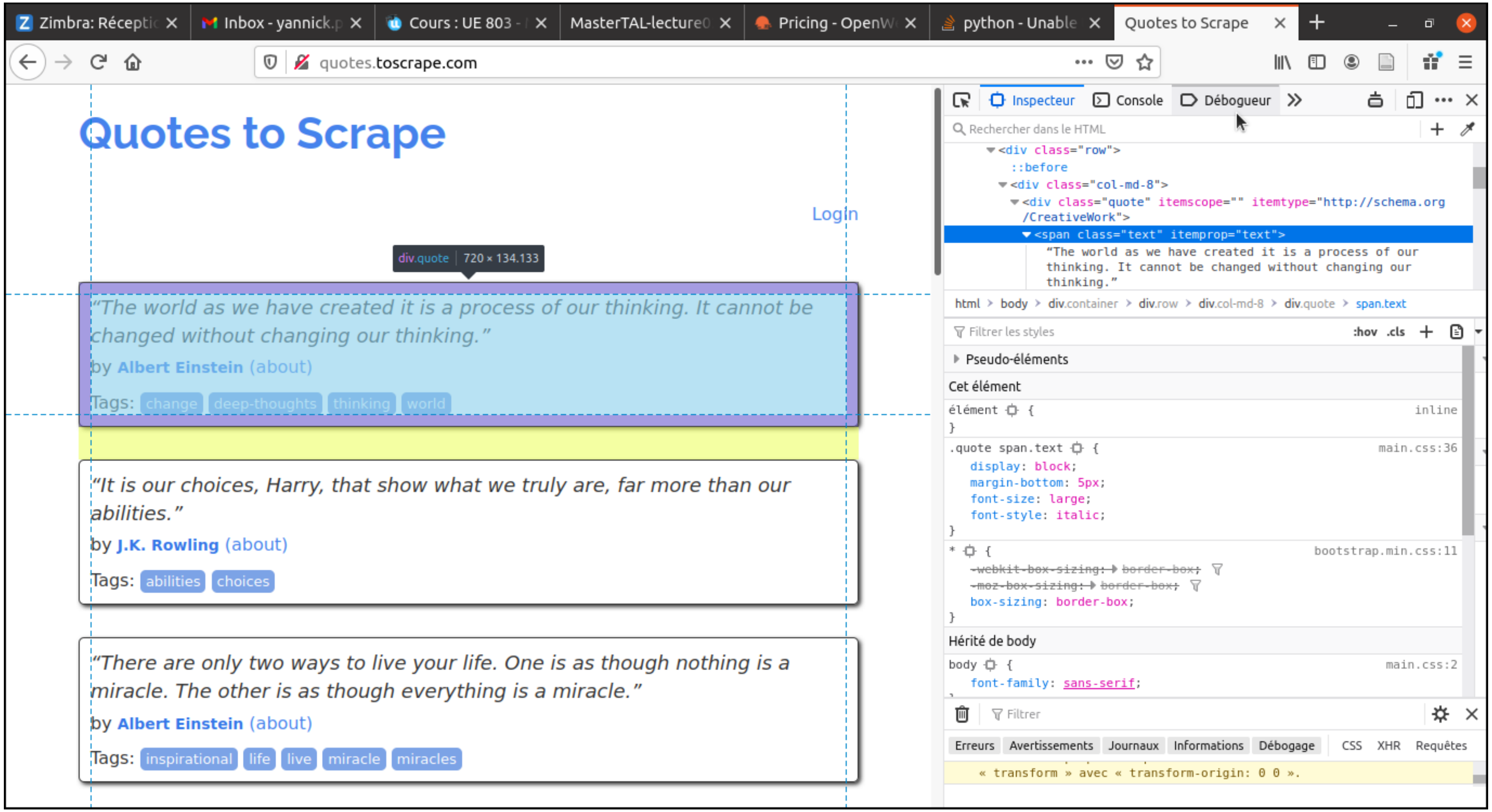
NB: these retrieve XML elements!

4. Case study

Collect quotes

Finding data

- Say we want to extract quotes from `http://quotes.toscrape.com`:



Quotes to Scrape

Login

“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”
by Albert Einstein (about)
Tags: change deep-thoughts thinking world

“It is our choices, Harry, that show what we truly are, far more than our abilities.”
by J.K. Rowling (about)
Tags: abilities choices

“There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”
by Albert Einstein (about)
Tags: inspirational life live miracle miracles

```
<div class="row">
  ::before
  <div class="col-md-8">
    <div class="quote" itemscope="" itemtype="http://schema.org
    /CreativeWork">
      <span class="text" itemprop="text">
        “The world as we have created it is a process of our
        thinking. It cannot be changed without changing our
        thinking.”
```

html > body > div.container > div.row > div.col-md-8 > div.quote > span.text

```
élément {                                   inline
}
.quote span.text {                          main.css:36
  display: block;
  margin-bottom: 5px;
  font-size: large;
  font-style: italic;
}
* {                                         bootstrap.min.css:11
  ~~-webkit-box-sizing: border-box;~~
  ~~-moz-box-sizing: border-box;~~
  box-sizing: border-box;
}
```

```
body {                                      main.css:2
  font-family: sans-serif;
```

```
« transform » avec « transform-origin: 0 0 ».
```

Parsing data

- Useful **HTML element**:

```html
<span class="text" itemprop="text">
  “The world as we have created it is a process of our thinking.
  It cannot be changed without changing our thinking.”
</span>
```

- To get *all* the quotes from the page:

```python
import requests
from bs4 import BeautifulSoup

def get_quotes(url):
  page= requests.get(url=url,headers={'User-Agent': 'Mozilla/5.0'})
  soup= BeautifulSoup(page.content, 'html.parser')
  quotes= soup.find_all(class_="text")
  return list(map(lambda x : x.get_text(), quotes))

url= 'http://quotes.toscrape.com'
l  = get_quotes(url)
```

Parsing data (continued)

- Useful **HTML element** to go to the next page:

```
<li class="next">
    <a href="/page/2/">Next <span aria-hidden="true">→</span></a>
</li>
```

- To get the quotes from the next page:

```
def get_next(url):
  page= requests.get(url=url,headers={'User-Agent': 'Mozilla/5.0'})
  soup= BeautifulSoup(page.content, 'html.parser')
  next= soup.find(class_="next")
  return next

next= get_next(url)
while next is not None:
  link= next.find('a')['href']
  url+= link
  l.extends(get_quotes(url))
  next= get_next(url)
```

Saving data

- To store the corresponding data in a text file:

```python
with open('quotes.txt','w') as f:
     for i in range(len(l)):
        print(i + ": " + l[i], file=f)
```

- Content of the text file (e.g. via `cat quotes.txt`)

```
1: “The world as we have created it is a process of our thinking.
It cannot be changed without changing our thinking.”
2: “It is our choices, Harry, that show what we truly are, far
more than our abilities.”
3: “There are only two ways to live your life. One is as though
nothing is a miracle. The other is as though everything is a
miracle.”
4: “The person, be it gentleman or lady, who has not pleasure in a
good novel, must be intolerably stupid.”
```

5. The automatic way

Introducing Scrapy

Introducint Scrapy

- open-source framework for web scraping
- includes (among others)
    - a crawling engine,
    - an HTML parser,
    - a JSON exporter
- can be used as a standalone command

Retrieving web pages: spiders

- Defining her.his own spider (class inheriting from Scrapy’s Spider class):

```
import scrapy
class QuotesSpider(scrapy.Spider):
```

- A Spider has a unique name, and some start URL(s):

```
class QuotesSpider(scrapy.Spider):
   name = "MyFirstQuoteSpider"
   start_urls = ["http://quotes.toscrape.com/page/1/"]
```

- You can limit scraping to some allowed_domains:

```
    allowed_domains=["quotes.toscrape.com"]
```

Parsing configuration

- Apply a `parse` call-back method on each page:

```python
class QuotesSpider(scrapy.Spider):
   name = "MyFirstQuoteSpider"
   start_urls = ["http://quotes.toscrape.com/page/1/"]

   def parse(self, response):
        page = response.url.split("/")[-2]
        filename = "quotes-%s.html" % page
        with open(filename, "wb") as f:
            f.write(response.body)
```

- Here, starting pages will be retrieved and stored locally

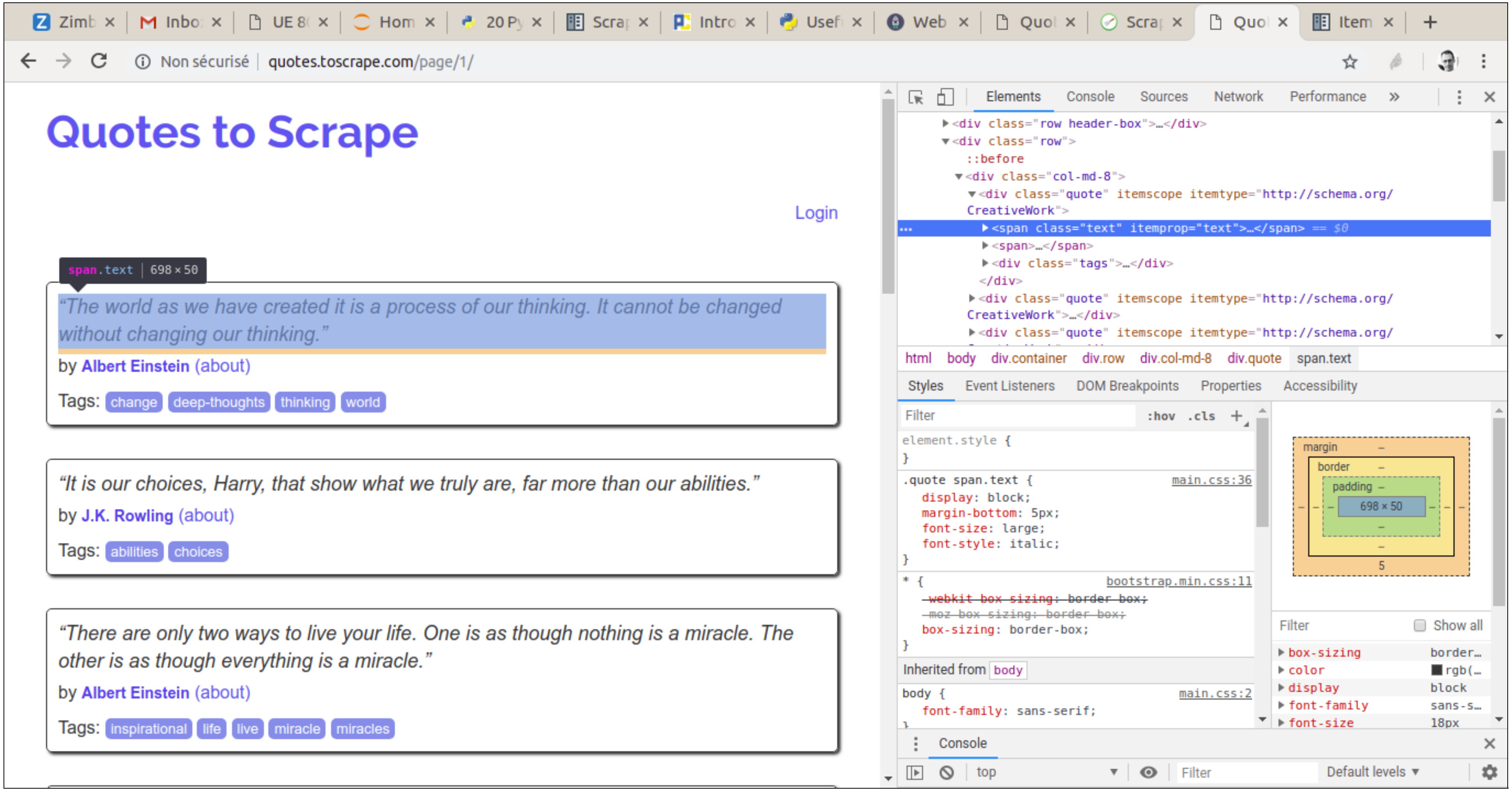
Running the spider

- Finally, to run this scraping engine, one must use a `CrawlerProcess` as follows:

```
if __name__=='__main__':
  import scrapy.crawler
  myspider = QuotesSpider()
  process = scrapy.crawler.CrawlerProcess({
    'USER_AGENT':'Mozilla/5.0'
  })
  process.crawl(myspider)
  process.start()
  process.stop()
```

Retrieving and extracting content

- Back to our case study:



Quotes to Scrape

Login

span.text | 698 × 50

“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”
by Albert Einstein (about)
Tags: change deep-thoughts thinking world

“It is our choices, Harry, that show what we truly are, far more than our abilities.”
by J.K. Rowling (about)
Tags: abilities choices

“There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”
by Albert Einstein (about)
Tags: inspirational life live miracle miracles

```
▶<div class="row header-box">…</div>
▼<div class="row">
   ::before
  ▼<div class="col-md-8">
    ▼<div class="quote" itemscope itemtype="http://schema.org/
    CreativeWork">
      ▶<span class="text" itemprop="text">…</span> == $0
      ▶<span>…</span>
      ▶<div class="tags">…</div>
     </div>
    ▶<div class="quote" itemscope itemtype="http://schema.org/
    CreativeWork">…</div>
    ▶<div class="quote" itemscope itemtype="http://schema.org/
```

html body div.container div.row div.col-md-8 div.quote span.text

```
element.style {
}
.quote span.text {                    main.css:36
  display: block;
  margin-bottom: 5px;
  font-size: large;
  font-style: italic;
}
* {                            bootstrap.min.css:11
  ~~-webkit-box-sizing: border-box;~~
  ~~-moz-box-sizing: border-box;~~
  box-sizing: border-box;
}
Inherited from body
body {                                 main.css:2
  font-family: sans-serif;
```

box-sizing	border…
color	rgb(…
display	block
font-family	sans-s…
font-size	18px

Content extraction

- We can extract quotes by using CSS selectors:

```python
def parse(self, response):
    with open('quotes.txt', 'a') as f:
        quotes = response.css("div.quote")
        for quote in quotes:
            text = quote.css('span.text::text').extract_first()
            print(text,file=f)
```

- Links can be extracted as well:

```html
<ul class="pager">
    <li class="next">
        <a href="/page/2/">Next (...) </a>
    </li>
</ul>
```

Case #1: Extracting links manually

- Using CSS selectors:

```
def parse(self, response):
    with open('quotes.txt', 'a') as f:
    quotes = response.css("div.quote")
    for quote in quotes:
        text = quote.css('span.text::text').extract_first()
        print(text,file=f)
    next_page=response.css('li.next a::attr(href)').extract_first()
    if next_page is not None:
        next_page = response.urljoin(next_page)
        yield scrapy.Request(next_page, callback=self.parse)
```

Case #2: Extracting links using scrapy

- Using the `follow` built-in method:

```python
def parse(self, response):
    with open('quotes.txt', 'a') as f:
    quotes = response.css("div.quote")
    for quote in quotes:
        text = quote.css('span.text::text').extract_first()
        print(text,file=f)
    next_page=response.css('li.next a::attr(href)').extract_first()
    if next_page is not None:
        yield response.follow(next_page, callback=self.parse)
```

Thank you!

Slideshow created using remark.

Exercise sheet #3 (available on Arche)

- Extracting data from the Internet Movie DataBase