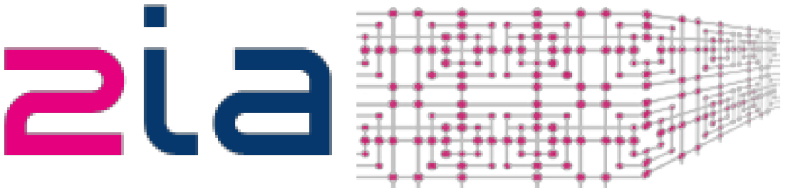
IMT Mines Alès
École Mines-Télécom



WEB SCRAPING

DATA COLLECTION & DESIGN OF EXPERIMENTS

30/02/2019

Andon Tchechmedjiev [1]

[1] LGI2P, IMT Mines Ales

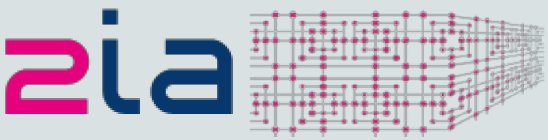
OUTLINE

1. Introduction

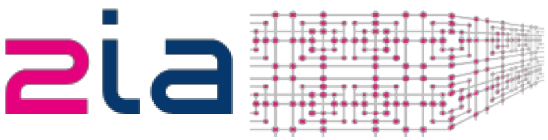
2. Tools for web-scraping

3. Practical application

IMT Mines Alès
École Mines-Télécom



INTRODUCTION



Web Scraping – General Context

For the design of experiments, one often needs to gather data beforehand:

- The “classical” way is to create a survey and collect the answers of individual.
- Nowadays, we can often find enormous amounts of data on the web
  - Some data are available through structured databases or APIs...
  - Whether as a database dump (e.g. SQL) or through a REST API interface, or yet on the Linked Data Cloud (Semantic Web) through SPARQL.
  - However, such data are a great minority of what’s available on the web: there are billions of web-pages with troves of information.

What are web-pages made of?

- Contrarily to databases and APIs that make structured information available, the information in web page is unstructured.
- Unstructured means that the data doesn't follow a formalized data schema (like a SQL schema).
- Web-pages are written in HTML and are structured in a way that facilitates human reading and access but not algorithmic access.
- So the main question is: How do we retrieve web pages and extract content from them?

Legality of Web Scraping

- Copyright law always applies to data found on websites and on the web in general: the copyright law applicable is that where the website is hosted.
- If the website is located within the US, US Copyright law applies, if it's located in the EU, EU copyright law applies.
- In the EU there are further restrictions related to the rights to privacy, in particular where GDPR is concerned.
- In essence: by default copyright applies, so unless it's stated, siphoning data from the web is illegal.
- Always check the Terms and Conditions of websites to make sure you are not legally liable!

Legality of Web Scraping – Exceptions

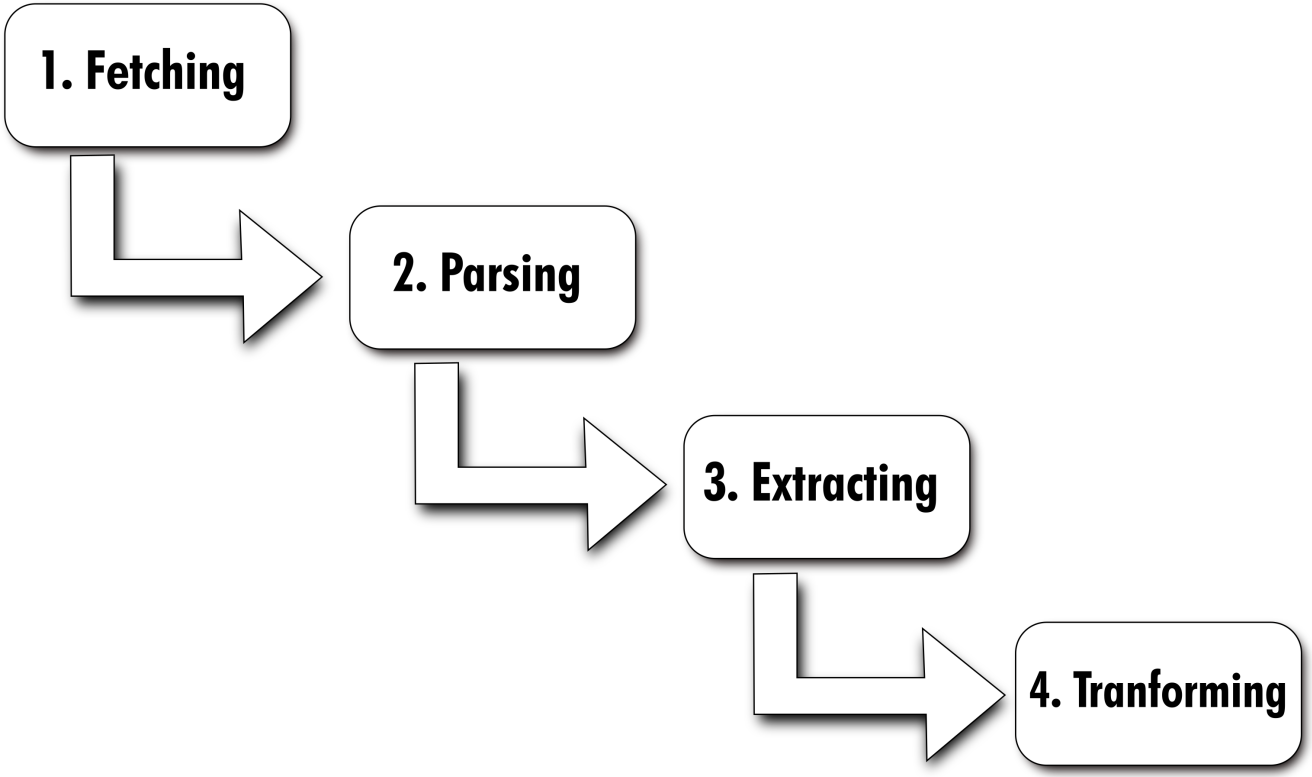
- A lot of government data is available in the public domain: all US administrations are bound to make a lot of their data (unless confidential) available un the public domain, so it's ok to scrape it.
- More and more the case in the EU.
- Creative Commons is your friend! But do check the particular modalities of the CC license (attribution, non commercial, etc.).
- All BSD-like licenses are public-domain.
- Same goes for open documentation licenses (GPL or Apache variants); but beware contagion!
- If in doubt always ask first.

Legality of Web Scraping

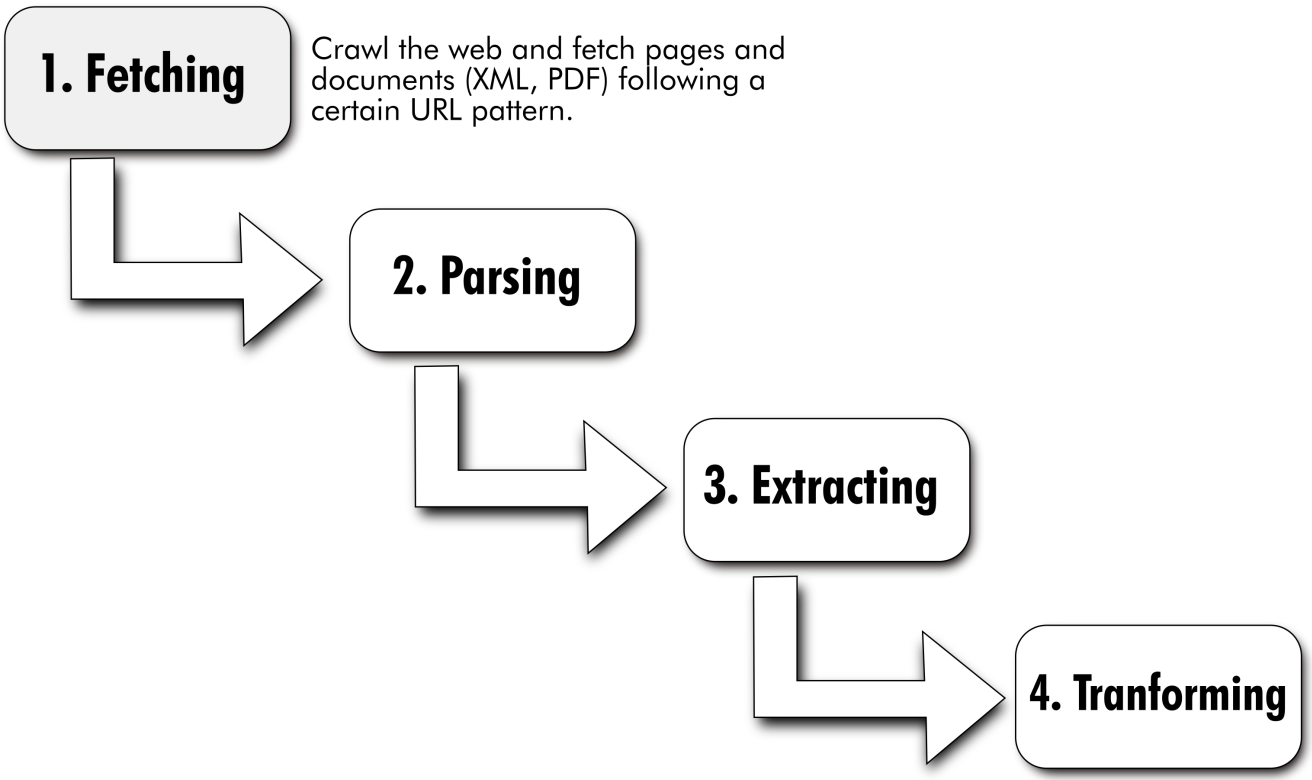
Beyond just whether its legal to exploit data found on websites, storing the data and then making it available can be problematic:

- Any personal data that allows to uniquely identify a user or group of users is very tricky to work with.
- Even if exploiting data is allowed, storing said data after transforming it may not be as it would infringe on data privacy laws.
- Making the data publicly available may not always be permissible although you are allowed to make a private copy.

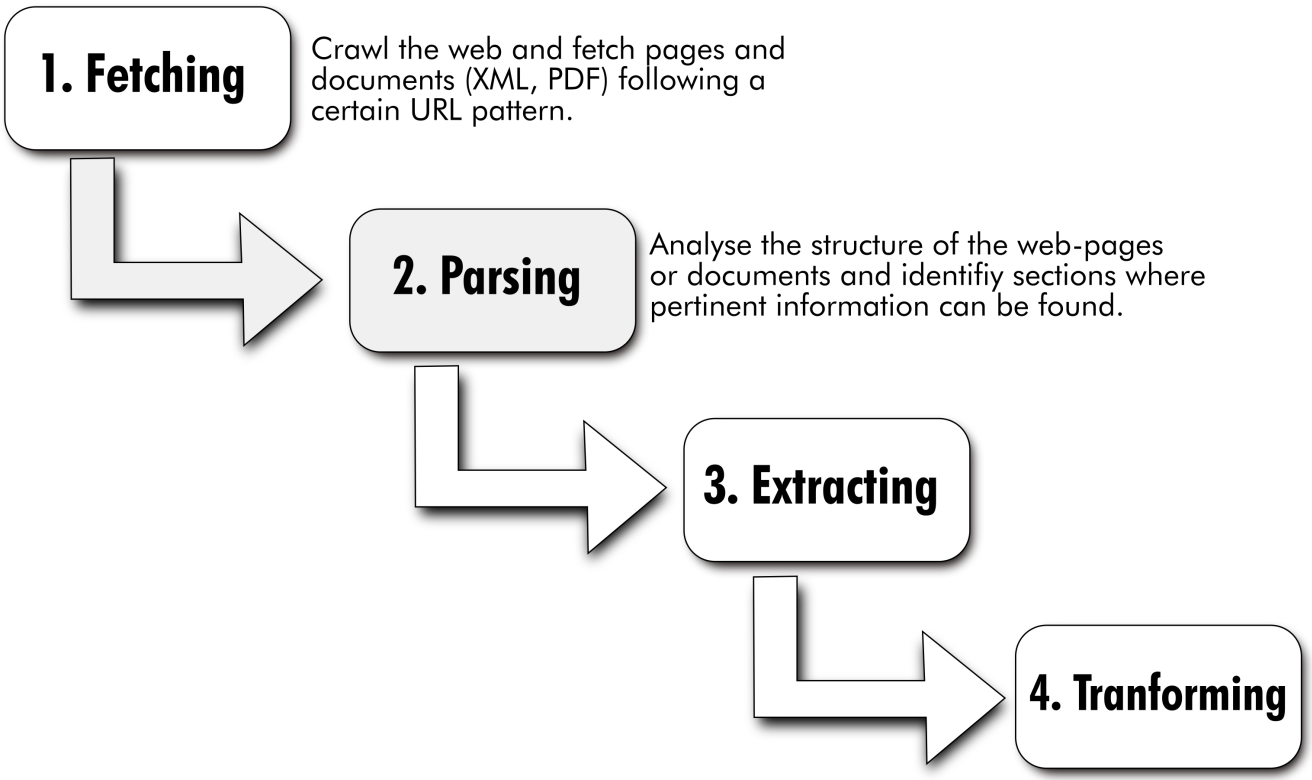
The Web Scraping pipeline



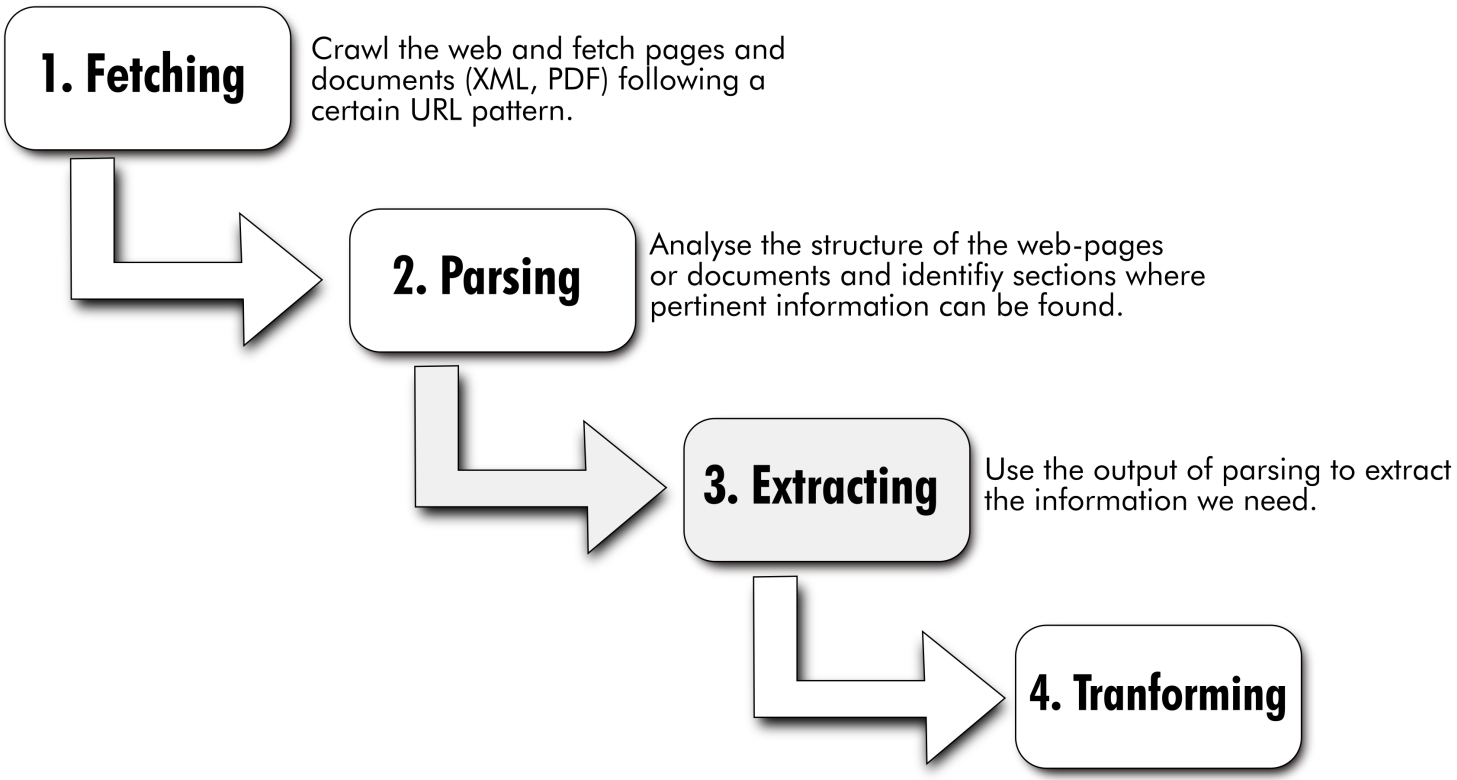
The Web Scraping pipeline



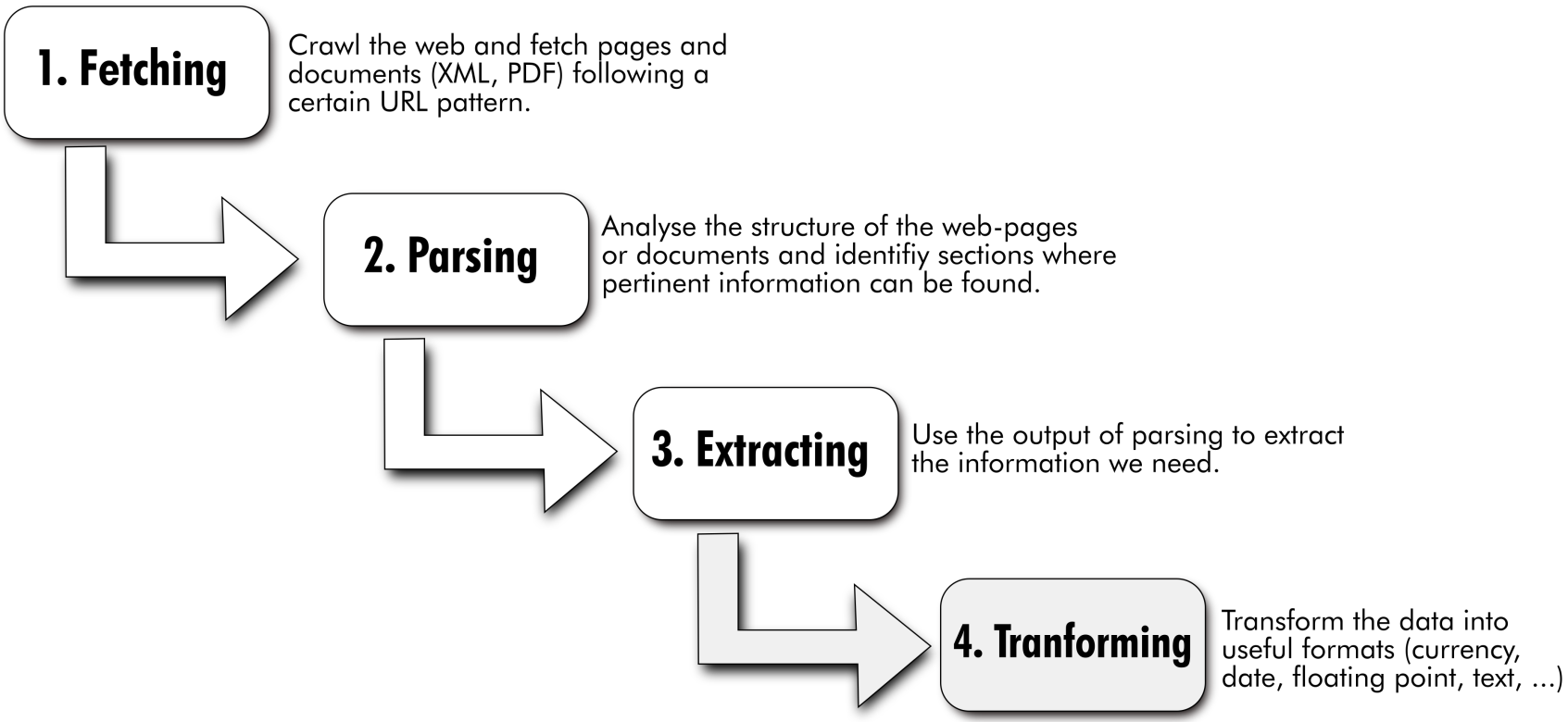
The Web Scraping pipeline



The Web Scraping pipeline



The Web Scraping pipeline



1. Fetching

- Load the document from the web (most often HTML or XML, less often PDF). Possibly as a single string.
- Sometimes a document is split across several pages and must be reassembled.

2. Parsing

- Interpret the raw document to be able to make sense of its structure.
- For HTML or XML, parsing will give you a Document Object Model (DOM).
- For a PDF it's a grid model (coordinates of containers and components).
- Other types of documents may have a different structure yet.
- This step is very sensitive to the documents following a proper syntax and may fail very easily (e.g. A lot of websites have malformed HTML).

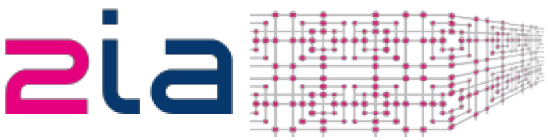
3. Extracting

- Search through the parsed data to retrieve values of interest (e.g. Selecting DOM elements with XPath)
- For example dates, some section in the body of the page, the title, links, the content from tables in the document.
- Separate the data of interest for the next stage in the pipeline.

4. Transformation

- Convert the identified data (strings) into the proper datatypes: e.g. convert dates from their string representation to date objects, convert numerical values to integers or floating point numbers.
- Instantiate into a complex data model (e.g. your Python practical sessions on fake news)
- Normalize data (map to standardized values, completion of missing data, filtering out noise).

TOOLS FOR WEB-SCRAPING



How to implement a web scraper?

1. There are ready to use tools (particularly online) to automatically do web scraping for you on simple use cases: For example ScraperWiki (tables and twitter, single page); Impot.io (crawler, free in some cases, accessible through API); Scraping hub (large-scale scraping with multi-site support, possible to run own scrappers written in python); and many more...
2. Directly in a programming language
   - A throwaway script for simple jobs
   - A more complex scraper using a combination of libraries and tools
   - Through generic web-scraping frameworks

Tools and libraries by programming language

R

- scrapeR (for XML/HTML)
- XML package
- tm to parse pdfs

PHP

- HTML DOM
- PDF Parser

Java

- HTMLUnit
- Many XML/HTML parsers and DOM libraries
- Many PDF parsing libraries

JavaScript

- NodeJS + Request + Cheerio
- jsPDF
- pdf2json

And in Python? – 1. Fetching

Requests

```
1  r = requests.get('https://www.google.com').html
```

urllib2

```
1  html = urllib2.urlopen('http://python.org/').read()
```

httplib2

```
1  h = httplib2.Http(".cache")
2  (resp_headers, content) = h.request("http://pydelhi.org/", "GET")
```

And in Python? – 2. Parsing

Beautiful soup

```
1  tree = BeautifulSoup(html_doc)
2  tree.title
```

lxml

```
1  tree = lxml.html.fromstring(html_doc)
2  title = tree.xpath('/title/text()')
```

re

```
1  title = re.findall('<title>(.*?)</title>', html_doc)
```

And in Python? – 2. Parsing

Beautifulsoup

- A beautiful API

```
1  soup = BeautifulSoup(html_doc)
2  last_a_tag = soup.find("a",{'id':'link3'})
3  all_b_tags = soup.find_all("b")
```

- Very easy to use
- Can handle broken markup (HTML or XML)
- Pure python
- Slow...

And in Python? – 2. Parsing

LXML is a python binding for the libxml and libxslt libraries written in C.

- Very fast
- Not pure python: if you don't have a pure python constraint it's best to use lxml.
- Works with all python versions

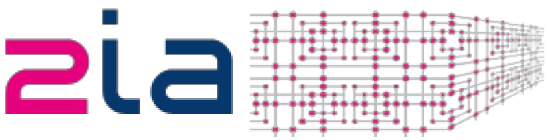
And in Python? – 2. Parsing

RE is the regular expressions library for Python and can be used in cases where there is little information to extracts.

- You need to learn regular expressions.
- Can quickly become complex
- Pure python
- Very fast

PRACTICAL APPLICATION

IMT Mines Alès
École Mines-Télécom



Technical requirements

For this course, we will mix theory with practice seamlessly. We will be working with PyCharm and you will need to install the following packages:

- BeautifulSoup4

Please create a new project in PyCharm and a new python file called extractor.py, we will be working within that file. Then, import the following packages:

```
1  import requests
2  from bs4 import BeautifulSoup
```

1. Fetching

First let us open a web-page and study its structure:
`https://web.archive.org/web/20170131230332/https://www.nga.gov/collection/an.shtm`

- We can see an index of artists by first letter of their last name or band name
- Let us take a simple example and deal with artists with names starting in Z.

1. Fetching

THE COLLECTION

NATIONAL GALLERY OF ART

What's New
Newsletters
Calendar
Recent Acquisitions
Videos & Podcasts
About the Gallery
Shock of the News
George Bellows

The Collection
Exhibitions
Plan a Visit
Programs & Events
Online Tours
Education
Resources
Gallery Shop
Support the Gallery
NGA Images
NGAkids

Search the Site GO

Artist names beginning with Z

Zabaglia, Niccola	Italian, 1664 - 1750
Zaccone, Fabian	American, 1910 - 1992
Zadkine, Ossip	French, 1890 - 1967
Zaech, Bernhard	German, active c. 1650
Zagar, Jacob	Flemish, c. 1530 - after 1580
Zagroba, Idalia	Polish, born 1967
Zaidenberg, A.	American, active c. 1935
Zaidenberg, Arthur	American, 1903 - 1990
Zaisinger, Matthäus	German, active c. 1500
Zajac, Jack	American, born 1929
Zak, Eugène	Polish, 1884 - 1926
Zakharov, Gurii Fillipovich	Russian, born 1926
Zakowortny, Igor	
Zalce, Alfredo	Mexican, born 1908
Zalopany, Michele	American, born 1955
Zammiello, Craig	
Zammitt, Norman	American, born 1931
Zampieri, Domenico	Italian, 1581 - 1641
Zampieri, called Domenichino, Domenico	Italian, 1581 - 1641
Zanartú, Enrique Antunez	Chilean, born 1921
Zanchi, Antonio	Italian, 1631 - 1722
Zanetti, Anton Maria	Italian, 1679/1680 - 1767
Zanetti Borzino, Leopoldina	Italian, 1826 - 1902
Zanetti I, Antonio Maria, conte	Italian, 1680 - 1757
Zanguidi, Jacopo	Italian, 1544 - 1573/1574
Zanini, Giuseppe	Italian, c. 1599 - 1631
Zanini-Viola, Giuseppe	Italian, c. 1599 - 1631
Zanotti, Giampietro	Italian, 1674 - 1765
Zao Wou-Ki	French, born 1921

Page 1 - Artists 1 thru 29 of 115 »

Zah-Zao Zas-Zie Zie-Zor next

1. Fetching

On this page, the first artist is **Zabaglia, Niccola**, write it down for later. The URL we get when we click on the previous page is: `https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anZ1.htm`

There are 4 pages in total, if you click to go to the last page, you end up with (Note the name of the last artist): `https://web.archive.org/web/20121010201041/http://www.nga.gov/collection/anZ4.htm`

Because of the archive the base URL has changed, but you can use the original base URL:

`https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anZ4.htm`

1. Fetching

In order to actually get the contents of the web page, we will use the Requests library to assign the contents of the html page into a page variable:

```
1  page = requests.get('https://web.archive.org/web/20121007172955'+\
2                      '/https://www.nga.gov/collection/anZ1.htm')
```

2. Parsing

We will now use beautiful soup to create a `BeautifulSoup` object, which contains the parse tree for the html page.

The constructor takes the content of the web page in a single string as its first argument. We will get it from `page.text` returned by the requests library.

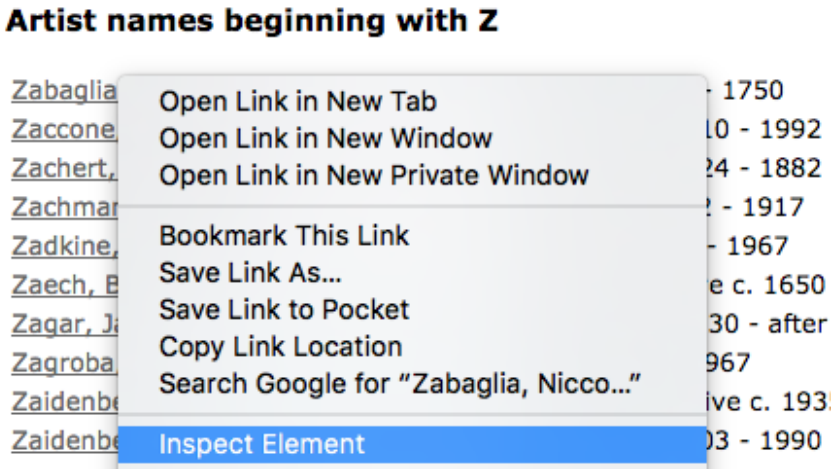
The second argument is the parser to use, here we will use Python's built-in parser, `html.parser`.

```
1  page = requests.get('https://web.archive.org/web/20121007172955'+\
2                      '/https://www.nga.gov/collection/anZ1.htm')
3  soup = BeautifulSoup(page.text, 'html.parser')
```

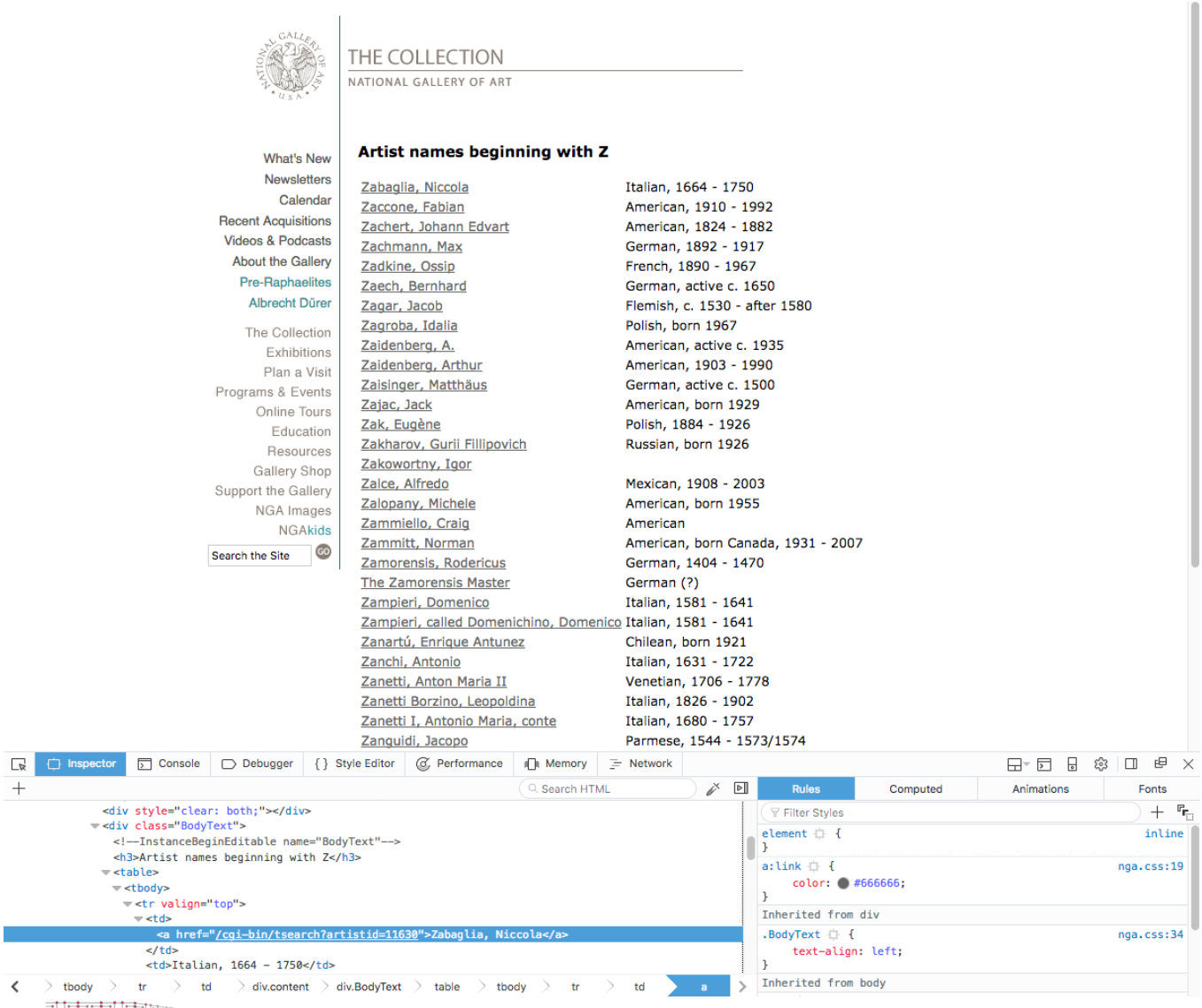
3. Extracting

In order to extract data from the parse tree, we first need to get an idea of how the page is structured. We can use a web browser and its 'Inspect' feature to explore the structure of the page.

Right click on the first artist's name and select the **Inspect** or **Inspect Element**.



3. Extracting



3. Extracting

- The main content of the page is within a `<div>` element that have for css class, `class="BodyText"`
- The name of the first artist, **Zabaglia, Niccola** is within a `<a/>` tag as it's a link to a separate page.
- If we want to get the name and link for each artists we'll need to find a way to access them.

3. Extracting

- Beautiful soup's `find()` and `find_all()` methods will help us access the content of the parsed web page:

```
# Pull all text from the BodyText div
artist_name_list = soup.find(attrs={'class':'BodyText'})

# Pull text from all instances of <a> tag within BodyText div
artist_name_list_items = artist_name_list.find_all('a')
```

3. Extracting

- Once we have the list of links, we will try to display them in the console: we can use a `for` loop on `artist_name_list_items`.
- To print the output, we can use the help of the prettify method to display the HTML is a well indented way.

```
1  # Create for loop to print out all artists' names
2  for artist_name in artist_name_list_items:
3      print(artist_name.prettify())
```

3. Extracting

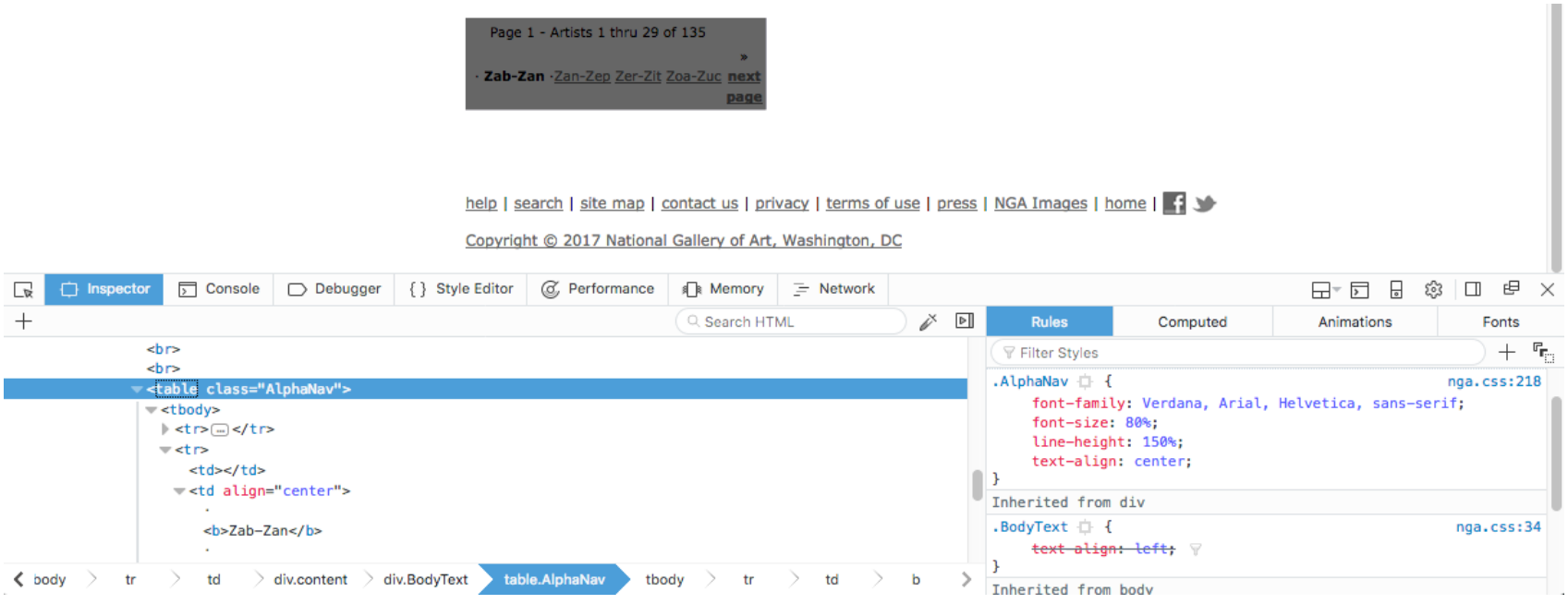
- Let's run it:

```
<a href="/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=11630">
Zabaglia, Niccola
</a>
...
<a href="/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=3427">
Zao Wou-Ki
</a>
<a href="/web/20121007172955/https://www.nga.gov/collection/anZ2.htm">
Zas-Zie
</a>
...
<a href="/web/20121007172955/https://www.nga.gov/collection/anZ3.htm">
Zie-Zor
</a>

<a href="/web/20121007172955/https://www.nga.gov/collection/anZ4.htm">
<strong>
next
<br/>
page
</strong>
</a>
```

3. Extracting

- There's a problem: We also get the links to go from page to page. We need to find a way of removing them! Let's inspect the page again:



- We can see that the navigation links are in an HTML table with the class `class='AlphaNav'`

3. Extracting

- Thankfully, Beautiful soup has a `decompose()` function that we can use to achieve just that:

```
1  page = requests.get('https://web.archive.org/web/20121007172955/'+\
2                      'https://www.nga.gov/collection/anZ1.htm')
3  soup = BeautifulSoup(page.text, 'html.parser')
4
5  # Remove bottom links
6  last_links = soup.find(attrs={'class':'AlphaNav'})
7  last_links.decompose()
8
9  artist_name_list = soup.find(class_='BodyText')
10 artist_name_list_items = artist_name_list.find_all('a')
11
12 for artist_name in artist_name_list_items:
13         print(artist_name.prettify())
```

4. Transforming

- We can extract the segments of interest in the document, but how do we only get the values?
- We can use the `.contents` attribute of Beautiful soup objects to get only the text between a starting and an ending tag. It's always a list even if there is a single element.
- Try to change your programme to take it into account

4. Transforming

- We can extract the segments of interest in the document, but how do we only get the values?
- We can use the `.contents` attribute of Beautiful soup objects to get only the text between a starting and an ending tag. It's always a list even if there is a single element.

```
1  # Use .contents to pull out the <a> tag's children
2  for artist_name in artist_name_list_items:
3          names = artist_name.contents[0]
4          print(names)
```

- Now run it again and check it works out.

4. Transforming

- Now, change the programme to also get the actual links and not just the text.
- You can use the `.get('href')` method.

4. Transforming

- Now, change the programme to also get the actual links and not just the text.
- You can use the `.get('href')` method.

```
...
for artist_name in artist_name_list_items:
        names = artist_name.contents[0]
        links = 'https://web.archive.org' + artist_name.get('href')
        print(names)
        print(links)
```

4. Transforming

- The last step is to now write the results to a csv file.
- We're going to use the `csv` library
  - Open a new csv file:
    ```
    f = csv.writer(open('file.csv', 'w'))
    ```
  - Write a row: `f.writerow(['item1', 'item2', ...])`
  - You need to first write a header and then loop over the items to save and write them one by one.

4. Transforming

```
1  # Create a file to write to, add headers row
2  f = csv.writer(open('z-artist-names.csv', 'w'))
3  f.writerow(['Name', 'Link'])
4
5  for artist_name in artist_name_list_items:
6          names = artist_name.contents[0]
7          links = 'https://web.archive.org' + artist_name.get('href')
8          # Add each artist's name and associated link to a row
9          f.writerow([names, links])
```

1. Fetching (again)

- Let's now improve the retrieval of the pages
- We only retrieved the first page for artists starting with letter Z, but there are 4.
- Now update your code to load all 4 pages so that all the information ends up in the csv file.
- Remember that the URL ends with: `anZ1.htm` and that the URLs for the following pages end with `anZ2.htm`, `anZ3.htm`...

1. Fetching (again)

```
1   pages = []
2   for i in range(1, 5):
3           url = 'https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ' +\
4            str(i) + '.htm'
5           pages.append(url)
6
7   for item in pages:
8           page = requests.get(item)
9           soup = BeautifulSoup(page.text, 'html.parser')
10          last_links = soup.find(attrs={'class':'AlphaNav'})
11          last_links.decompose()
12          artist_name_list = soup.find(attrs={'class':'BodyText'})
13          artist_name_list_items = artist_name_list.find_all('a')
14
15          for artist_name in artist_name_list_items:
16                  names = artist_name.contents[0]
17                  links = 'https://web.archive.org' + artist_name.get('href')
18                  f.writerow([names, links])
```

Sources and references

- Overview of tools for python web scraping
  `https://slides.com/manojp/introws`
- General introduction of the pipeline:
  `https://bit.ly/2RsAZwa`
- Another tutorial for Beautiful Soup:
  `https://bit.ly/2m0Ttt9`
- Two series tutorial for web scraping with Scrapy:
  `https://bit.ly/2CWeVVm` and `https://bit.ly/2TmQszB`