Introduction to Webscraping

Static pages

Laurent R. Bergé
B$_x$SE, University of Bordeaux
Fall 2025 ∼ M2-IREF

What is webscraping?

- webscraping is the art of collecting data from web pages
- **anything** you see when browsing the internet *is* data
- any data in a web page can be collected

Why doing that?

Sometimes that’s the only way to get the information you want!

Why doing that?

Sometimes that's the only way to get the information you want!

To consider

Web scraping is time consuming and is also costly in terms of resources (both for you and the server you're scraping). You should think hard to alternative solutions first!*

*: One solution is just to ask the owner, e.g. there's often a dedicated API provided.

Three types of task in webscraping

1. organizing the web scraping (only for large tasks)
2.
3.

Three types of task in webscraping

1. organizing the web scraping (only for large tasks)
2. getting the data from the web (actual web scraping)
3.

Three types of task in webscraping

1. organizing the web scraping (only for large tasks)
2. getting the data from the web (actual web scraping)
3. formatting the data

Typologies of web scraping tasks

	small number of pages	large number of pages
static webpage	direct	need to plan well
dynamic webpage	requires web knowledge	difficult

Objective

- to give you key knowledge to understand how to tackle ambitious web scraping projects

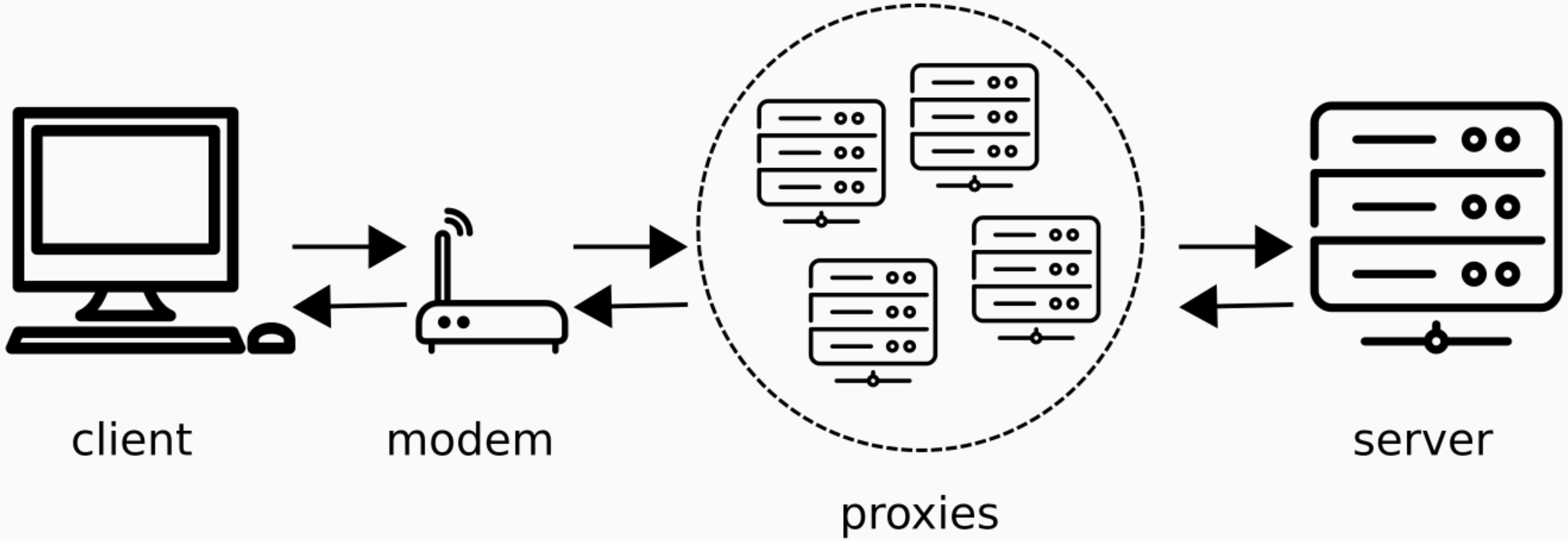
So you'll need to have a correct understanding of how the web works!

Outline

- how static web pages work
- how to scrape static pages
- how dynamic web pages work
- how to scrape dynamic pages

How does the web work?

The HTTP protocol



Example of HTTP GET request

```
GET HTTP/1.1
Host: www.bse.u-bordeaux.fr
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/143.0.0.0 Safari/537.36
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Server sends back a response

```
HTTP/1.1 200 OK
Date: Mon, 08 Dec 2025 15:56:07 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Mon, 08 Dec 2025 15:55:16 GMT
Vary: Accept-Encoding
Content-Encoding: gzip
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html... (here comes the requested web page)
```

200 = OK
Q: What are the other codes that you know?

Source: mozilla

What’s a webpage?

- a web page is just code that is interpreted by your browser
- the language in which the content is written is **HTML**
- HTML is just about content!

What’s HTML: Example from the European Parliament

- let’s inspect the Members of the European Parliament https://www.europarl.europa.eu/meps/en/full-list
- right click anywhere on the page and click `View page source` (on Chrome/Firefox)

How does HTML work?

- this is a markup language
- the content of each HTML element is enclosed in tags
- tags can have attributes

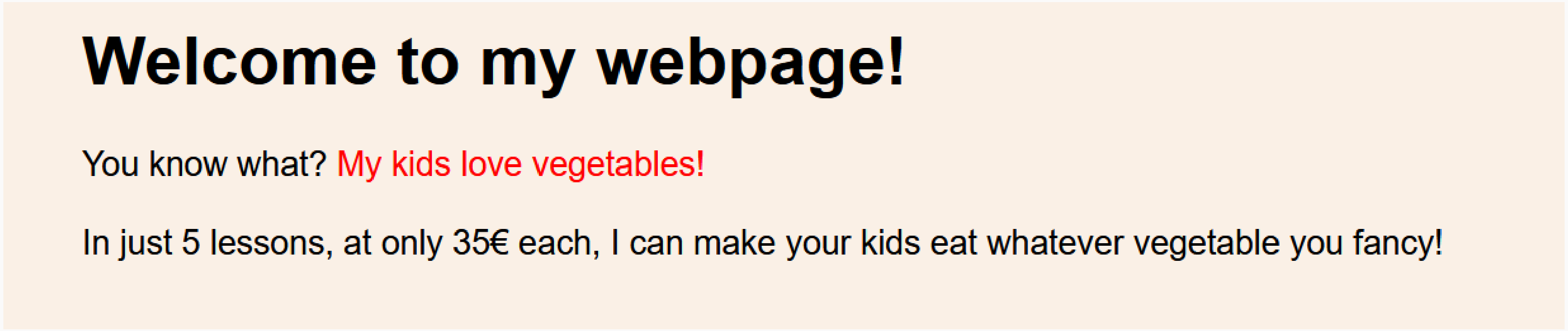
How does HTML work?

- this is a markup language
- the content of each HTML element is enclosed in tags
- tags can have attributes

… that’s basically it!

How does HTML work? Example page

Let’s have a look at this page:



Welcome to my webpage!

You know what? My kids love vegetables!

In just 5 lessons, at only 35€ each, I can make your kids eat whatever vegetable you fancy!

How does HTML work? Example page’s html

```html
<!DOCTYPE html>
<html>
<body>

<h1> Welcome to my webpage! </h1>

<p>
  You know what? <span class="lying">My kids love vegetables!</span>
  <br>
  <br>
  In just 5 lessons, at only 35€ each, I can make your kids eat whatever
vegetable you fancy!
</p>

</body>
</html>
```

How does HTML work? Tags

$$\overbrace{\texttt{<h1>}}^{\texttt{Opening tag}}\ \underbrace{\texttt{Welcome to my webpage!}}_{\texttt{Content}}\ \overbrace{\texttt{</h1>}}^{\texttt{closing tag}}$$

How does HTML work? Attributes

$$\texttt{<p }\overbrace{\texttt{class="lying"}}^{\texttt{Attribute}}\texttt{> My kids love vegetables! </p>}$$

Empty tags

Some tags don’t need closing tags, for exmple:

- `<img>` to insert images
- `<br>` to insert line breaks

Most common tags

`h1` – `h4`	headers
`p`	paragraphs
`a`	link (***a**nchor*)

```html
<a href = "https://cran.r-project.org/"> coding </a>
```

`img`	image

```html
<img src="grumpy_cat.jpg">
```

`strong`	to emphasize text
`ol`, `ul`	ordered and unordered list
`li`	item within a list

```html
<ul>
  <li>first item</li>
  <li>second item</li>
</ul>
```

`div`	generic box (this one is *everywhere*)
`span`	a box *within a paragraph*

HTML: Our webpage

let’s write our first web page!

1. create a folder named `my-website`
2. in that folder, create the file `index.html`
3. in that file, create your personal web page, which should contain:
   - a welcome title (`h1`) where your name should appear within a `span`
   - a brief description of who you are in a paragraph (`p`)
   - a list (`ul` with `li`) containing stuff that you like, with at least one link (`a`)
   - stuff that you don’t like
   - a quote that you like (`q`)
   - an image (`img`)

```html
<!DOCTYPE html>
<body>
<h1> John Doe's webpage </h1>
<h4> Short presentation </h4>
<p> Brielfly present yourself </p>
<h4> The thnigs I like </h4>
<ol>
  <li> first item </li>
  <li> second item </li>
</ol>
<h4> The thnigs I hate </h4>
<ol>
  <li> first item </li>
  <li> second item </li>
</ol>
<img src="images_path.png">
<a href='URL'> add links with the
              'a' tag</a>
</body>
```

Why is our web page boring?

Questions

- but, wait a minute.. why is our webpage so boring?
- why are others’ webpages so fancy?

Why is our web page boring?

Questions

- but, wait a minute.. why is our webpage so boring?
- why are others’ webpages so fancy?

Answer

- HTML is only about content, not about style
- HTML is nothing without its best friend **CSS**
- CSS is *only about style*

How does CSS work?

- CSS is a language indicating (to the browser) how to style your HTML elements

$$\overbrace{\texttt{span}}^{\texttt{selector}}\ \{$$

$$\underbrace{\texttt{color}}_{\texttt{key}} : \underbrace{\texttt{Crimson}}_{\texttt{value}} ;$$

$$\}$$

Example: HTML

The HTML code...

```
<head>
  <style>
    span {
      color: Crimson;
    }
  </style>
</head>
<body>
<p>
  You know what?
  <span> My kids love vegetables! </span>
</p>
</body>
```

Renders as:

You know what? My kids love vegetables!

See this CSS animation by Adam Kuhn.

CSS: Example

let’s add CSS to our web page!

1. to include CSS code: in the `<head>` tag (before `<body>`), add CSS in a `<style>` tag, ex:

```
<head>
<style>
  body {
    font-size: 16pt;
  }
</>
</head>
<body>
...
</body>
```

2. use CSS to:
   - increase the font-size of all elements in the page
   - set the font-family to sans-serif for all elements in the page
   - change the background-color of the page to Linen★
   - add a border-radius of 30% (or more) to the image

★: There are 140 predefined colors in HTML.

CSS: Can we do more?

I would like to have:

- my name in italic and in a different font
- the stuff that I like in green (**ForestGreen**)
- the stuff that I dislike in red (**Crimson**)
- the things that I *really*, like or dislike in bold

CSS: Can we do more?

I would like to have:

- my name in italic and in a different font
- the stuff that I like in green (**ForestGreen**)
- the stuff that I dislike in red (**Crimson**)
- the things that I *really*, like or dislike in bold

Q: At the moment, can I do that?

A: Not yet, because we need to select precisely some elements!

*We need **selectors!!!***

CSS selectors

CSS selectors

- **CSS selectors** indicate precisely which HTML element you want to style
- typically, HTML tags will contain attributes in order to be found via CSS selectors
- the main attribute used in HTML is the `class`[1]

[1]: The `id` attribute is usually less useful in webscraping.

CSS selectors: Most common selectors

```
p               : all "p" tags
p span          : all "span" contained in "p" tags
p, a            : all "p" and "a" tags
#id1            : all elements with id equal to id1
.class1         : all elements of class "class1"
p.class1        : all "p" elements of class "class1"
p.class1 span : all "span" in "p" tags of class "class1"
p > span        : all "span" that are direct children of p
h1 + p          : all "p" that follow *directly* an "h1" (direct sibling)
h1 ~ p          : all "p" that follow an "h1" (siblings placed after)
[id]            : all elements with an existing "id" attribute
[class^=my]     : all elements whose class starts with "my"
p[class*=low] : all "p" elements whose class contains the string low
etc!
```

See a list in w3schools, and the test page.

Exercise 1

Q: Select the following paragraph.

```
<h2>Who am I?</h2>

<div class="intro">
  <p>I'm <span class="age">34</span> and
    measure <span class="unit">1.70m</span>.</p>
</div>

<div class="info">
  <div>
    <p id="like">What I like:</p>
    <ul>
      <li>Barcelona</li>
      <li>winning the Ballon d'Or every odd year</li>
    </ul>
    <p class="extra">I forgot to say that I like scoring over 100
       goals per season.</p>
  </div>

  <div>
    <p class="dislike">What I don't like:</p>
    <ul>
      <li><span class="foe">Real Madrid</span></li>
      <li>leaving the club in which I've played since
       <span class="age">13</span></li>
    </ul>
  </div>
</div>
```

Exercise 1

Q: Select the following paragraph.

A: `p.extra`

```
<h2>Who am I?</h2>

<div class="intro">
  <p>I'm <span class="age">34</span> and
    measure <span class="unit">1.70m</span>.</p>
</div>

<div class="info">
  <div>
    <p id="like">What I like:</p>
    <ul>
      <li>Barcelona</li>
      <li>winning the Ballon d'Or every odd year</li>
    </ul>
    <p class="extra">I forgot to say that I like scoring over 100
       goals per season.</p>
  </div>

  <div>
    <p class="dislike">What I don't like:</p>
    <ul>
      <li><span class="foe">Real Madrid</span></li>
      <li>leaving the club in which I've played since
       <span class="age">13</span></li>
    </ul>
  </div>
</div>
```

Exercise 2

Q: Select the two highlighted `div`s.

```
<h2>Who am I?</h2>

<div class="intro">
  <p>I'm <span class="age">34</span> and
    measure <span class="unit">1.70m</span>.</p>
</div>

<div class="info">
  <div>
    <p id="like">What I like:</p>
    <ul>
      <li>Barcelona</li>
      <li>winning the Ballon d'Or every odd year</li>
    </ul>
    <p class="extra">I forgot to say that I like scoring over 100
       goals per season.</p>
  </div>

  <div>
    <p class="dislike">What I don't like:</p>
    <ul>
      <li><span class="foe">Real Madrid</span></li>
      <li>leaving the club in which I've played since
       <span class="age">13</span></li>
    </ul>
  </div>
</div>
```

Exercise 2

Q: Select the two highlighted `div`s.

A: `div.info > div`

```
<h2>Who am I?</h2>

<div class="intro">
  <p>I'm <span class="age">34</span> and
    measure <span class="unit">1.70m</span>.</p>
</div>

<div class="info">
  <div>
    <p id="like">What I like:</p>
    <ul>
      <li>Barcelona</li>
      <li>winning the Ballon d'Or every odd year</li>
    </ul>
    <p class="extra">I forgot to say that I like scoring over 100
       goals per season.</p>
  </div>

  <div>
    <p class="dislike">What I don't like:</p>
    <ul>
      <li><span class="foe">Real Madrid</span></li>
      <li>leaving the club in which I've played since
       <span class="age">13</span></li>
    </ul>
  </div>
</div>
```

Exercise 3

Q: Select the two highlighted `li`s.

```
<h2>Who am I?</h2>

<div class="intro">
  <p>I'm <span class="age">34</span> and
    measure <span class="unit">1.70m</span>.</p>
</div>

<div class="info">
  <div>
    <p id="like">What I like:</p>
    <ul>
      <li>Barcelona</li>
      <li>winning the Ballon d'Or every odd year</li>
    </ul>
    <p class="extra">I forgot to say that I like scoring over 100
       goals per season.</p>
  </div>

  <div>
    <p class="dislike">What I don't like:</p>
    <ul>
      <li><span class="foe">Real Madrid</span></li>
      <li>leaving the club in which I've played since
       <span class="age">13</span></li>
    </ul>
  </div>
</div>
```

Exercise 3

Q: Select the two highlighted `li`s.

A: `#like ~ ul > li`

```
<h2>Who am I?</h2>

<div class="intro">
  <p>I'm <span class="age">34</span> and
    measure <span class="unit">1.70m</span>.</p>
</div>

<div class="info">
  <div>
    <p id="like">What I like:</p>
    <ul>
      <li>Barcelona</li>
      <li>winning the Ballon d'Or every odd year</li>
    </ul>
    <p class="extra">I forgot to say that I like scoring over 100
       goals per season.</p>
  </div>

  <div>
    <p class="dislike">What I don't like:</p>
    <ul>
      <li><span class="foe">Real Madrid</span></li>
      <li>leaving the club in which I've played since
       <span class="age">13</span></li>
    </ul>
  </div>
</div>
```

A note on classes

An HTML element can have several classes separated with spaces:

```
<p class="first main low-key"> That's only an example! </p>
```

A note on classes

An HTML element can have several classes separated with spaces:

```
<p class="first main low-key"> That's only an example! </p>
```

- when using `.class`, the class **is not** the full string `"first main low-key"`
- there are three separate classes: `first`, `main` and `low-key`, which can be selected with the `p.class` syntax

A note on classes

An HTML element can have several classes separated with spaces:

```
<p class="first main low-key"> That's only an example! </p>
```

- when using `.class`, the class **is not** the full string `"first main low-key"`
- there are three separate classes: `first`, `main` and `low-key`, which can be selected with the `p.class` syntax

This means that the paragraph can be selected with either:

```
p.first
p.main
p.low-key
p.first.main
```

Selectors: XPath

- **XPath** is a language to make selections in XML documents .comment[it is not linked to CSS]
- it’s like… a path to a document: `/path/to/object` but instead of having folders, you have tags$^{\bigstar}$
- I only mention it since we don’t have the time to cover XPATH. Just be aware the sometimes it’s much easier to select elements with XPATH instead of CSS selectors.
- that said… CSS selectors should be enough for 95% of the use cases.

$^{\bigstar}$: It’s actually more complicated than that, but it’s a fair first approximation.

Why are selectors important?

- when you scrape a web page, you don’t want all the content from the web page: you focus only on specific elements
- you select elements using **CSS selectors** or **XPath**

Wrapping up

Selectors are **powerful tools** to select HTML elements.

Resources

- CSS
- XPath
- a website I created to learn and test selectors

Selectors: Pratcice 1

- 🖱 let's add more CSS to our webpage
- let's have:
  - my name in italic (`font-style: italic;`) and in a different font (`font-family: Arial, sans-serif;`)
  - the stuff that I like in green (`color:` **ForestGreen**)
  - the stuff that I dislike in red (`color:` **Crimson**)

Note that to do this you will need to add attributes to your HTML elements (`class` / `id`)

Selectors: Pratcice 2

Let’s play 5 min to a game to learn selectors that I created:

https://lrberge.github.io/learn_selectors

What I see and what I scrape

To remember

You don’t scrape what you see in the browser, you scrape the HTML code which, after application of CSS styles, renders on your browser.

It’s important to set the CSS aside (that’s why we need to understand what it does)!

Scraping in python

Tools

Good news

You have readily available tools to webscrape in Python.

In python, you'll need:

- requests
- BeautifulSoup (the names of the package is beautifulsoup4)
- and that's it! (for the easy stuff!)

Exercise 1

1. go to the page of the Members of the European Parliament https://www.europarl.europa.eu/meps/en/full-list/all
2. get the name and summary information of all the members

Exercise 1

1. go to the page of the Members of the European Parliament
   https://www.europarl.europa.eu/meps/en/full-list/all
2. get the name and summary information of all the members

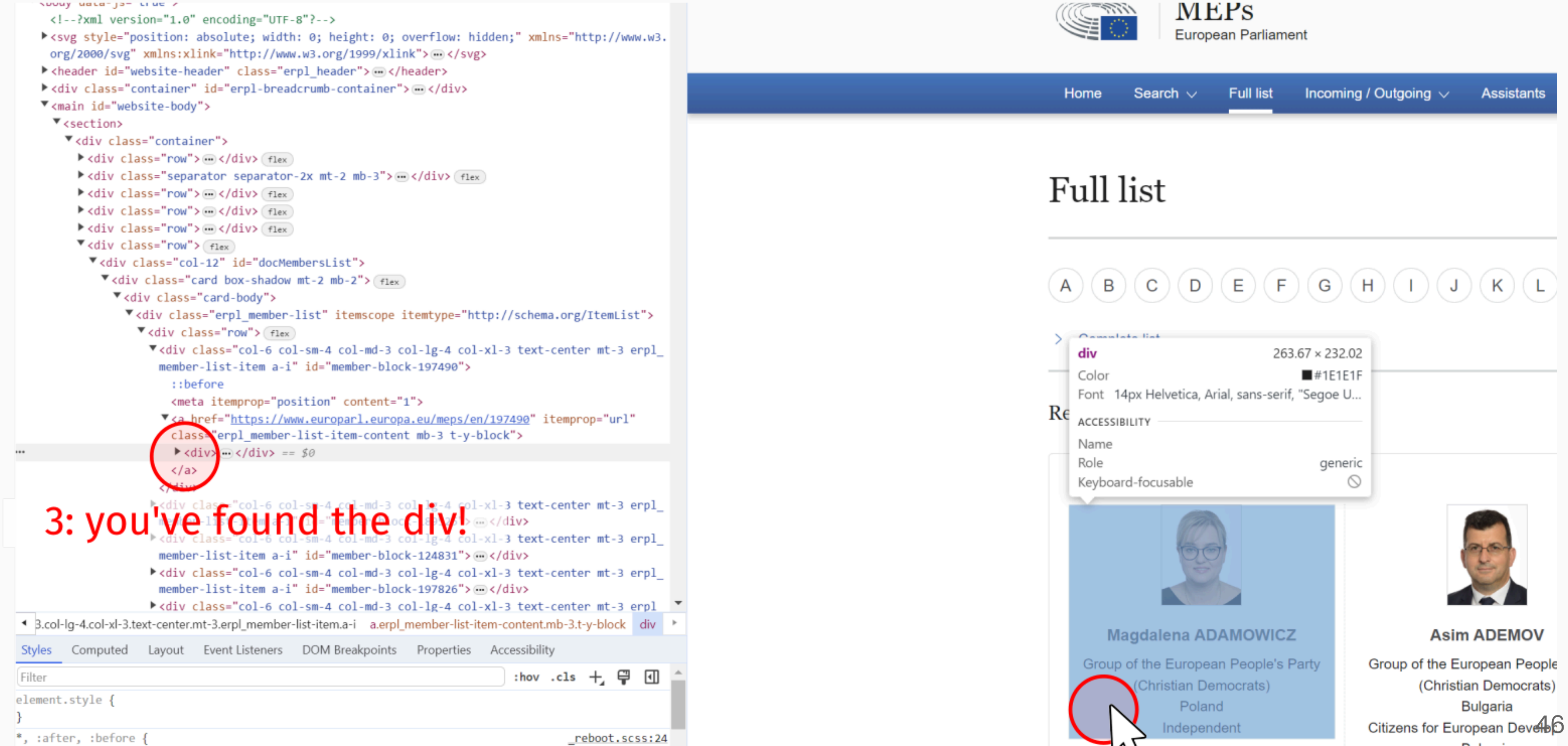
Use this structure...

```
# 1) get the HTML
import requests
page = requests.get("URL")

# 2) parse the HTML
from bs4 import BeautifulSoup
soup = BeautifulSoup(page.text, 'html.parser')

# 3) select the elements you want
all_elems = soup.select("CSS_SELECTOR")

# 4) obtain the "text content" of the element with .get_text()
all_elems[0].get_text()
```

Exercise 1: Tips

To find out which HTML element you are interested in: use the developer tools of your browers!

- press F12, or ctrl-shift-I on Chrome, or go to `parameters`›`more tools`›`developer tools`, then



```
<body data-js="true">
  <!--?xml version="1.0" encoding="UTF-8"?-->
  ▸<svg style="position: absolute; width: 0; height: 0; overflow: hidden;" xmlns="http://www.w3.
  org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">⋯</svg>
  ▸<header id="website-header" class="erpl_header">⋯</header>
  ▸<div class="container" id="erpl-breadcrumb-container">⋯</div>
  ▾<main id="website-body">
    ▾<section>
      ▾<div class="container">
        ▸<div class="row">⋯</div> flex
        ▸<div class="separator separator-2x mt-2 mb-3">⋯</div> flex
        ▸<div class="row">⋯</div> flex
        ▸<div class="row">⋯</div> flex
        ▸<div class="row">⋯</div> flex
        ▾<div class="row"> flex
          ▾<div class="col-12" id="docMembersList">
            ▾<div class="card box-shadow mt-2 mb-2"> flex
              ▾<div class="card-body">
                ▾<div class="erpl_member-list" itemscope itemtype="http://schema.org/ItemList">
                  ▾<div class="row"> flex
                    ▾<div class="col-6 col-sm-4 col-md-3 col-lg-4 col-xl-3 text-center mt-3 erpl_
                    member-list-item a-i" id="member-block-197490">
                      ::before
                      <meta itemprop="position" content="1">
                      ▾<a href="https://www.europarl.europa.eu/meps/en/197490" itemprop="url"
                      class="erpl_member-list-item-content mb-3 t-y-block">
                        ▸<div>⋯</div> == $0
                      </a>
                    </div>
                    ▸<div class="col-6 col-sm-4 col-md-3 col-lg-4 col-xl-3 text-center mt-3 erpl_
                    member-list-item a-i" id="member-block-...">⋯</div>
                    ▸<div class="col-6 col-sm-4 col-md-3 col-lg-4 col-xl-3 text-center mt-3 erpl_
                    member-list-item a-i" id="member-block-124831">⋯</div>
                    ▸<div class="col-6 col-sm-4 col-md-3 col-lg-4 col-xl-3 text-center mt-3 erpl_
                    member-list-item a-i" id="member-block-197826">⋯</div>
                    ▸<div class="col-6 col-sm-4 col-md-3 col-lg-4 col-xl-3 text-center mt-3 erpl
```

```
3.col-lg-4.col-xl-3.text-center.mt-3.erpl_member-list-item.a-i  a.erpl_member-list-item-content.mb-3.t-y-block  div
```

```
element.style {
}
*, :after, :before {                                                    _reboot.scss:24
```

```
div                                           263.67 × 232.02
Color                                         #1E1E1F
Font   14px Helvetica, Arial, sans-serif, "Segoe U...
ACCESSIBILITY
Name
Role                                          generic
Keyboard-focusable
```

Magdalena ADAMOWICZ
Group of the European People's Party (Christian Democrats)
Poland
Independent

Asim ADEMOV
Group of the European People (Christian Democrats)
Bulgaria
Citizens for European Develo

Exercise 1: Tips.. continued

It can be handy to write the HTML in a file and have a look at it. Do this with the `.prettify()` method from `BeautifulSoup`.

```python
# NOTA: you need the encoding argument
with open("tmp.html", "w", encoding="utf-8") as f:
  f.write(soup.prettify())
```

Then you can look at the HTML directly in your editor.

Exercise 2

Find the dates of birth of the first 5 French MEPs.

Nota: Take your time!

Note that you should be kind when you scrape, it’s always good to add some wait time between two HTTP requests.

Example:

```python
# Have a 3s pause
import time
time.sleep(3)
```

Dynamic web pages

Static vs Dynamic

- static: HTML in source code $=$ HTML in browser
- dynamic: HTML in source code $\neq$ HTML in browser

Q: What makes the HTML in your browser change?

Static vs Dynamic

- static: HTML in source code $=$ HTML in browser
- dynamic: HTML in source code $\neq$ HTML in browser

Q: What makes the HTML in your browser change?

javascript

The language of the web

Web's trinity:

HTML for content

CSS for style

javascript for manipulation

What's JS?

A programming language

- javascript is a regular programming language with the typical package: conditions, loops, functions, classes

Which...

- specializes in modifying HTML content

Why JS?

JS is capable of so many things… It is used to change the content of a webpage based on the user's actions.

Example of user actions:

- clicking
- moving the mouse
- scrolling

Example of events managed by JS:

- loading new data onto the web page
- changing the style (CSS)
- changing the content, like adding/hiding/removing HTML elements
- fancy animations

javascript is indispensable!

JS: what's the connection to webscraping?

- some webpages may decide to display some information only after some event has occurred
- the event can be:
  - the main HTML has loaded
  - an HTML box becomes, or is close to become, on-screen e.g. think to facebook
  - something is clicked
  - etc!

JS: what’s the connection to webscraping?

- some webpages may decide to display some information only after some event has occurred
- the event can be:
  - the main HTML has loaded
  - an HTML box becomes, or is close to become, on-screen e.g. think to facebook
  - something is clicked
  - etc!

Q: So far, we only queried the server to have the source code of the webpage. What’s the problem with that?

JS: what’s the connection to webscraping?

- some webpages may decide to display some information only after some event has occurred
- the event can be:
  - the main HTML has loaded
  - an HTML box becomes, or is close to become, on-screen e.g. think to facebook
  - something is clicked
  - etc!

Q: So far, we only queried the server to have the source code of the webpage. What’s the problem with that?

A: If you wanted to have access to some information that only appears after these events... well... you can’t.

JS: How does it work?

- you can add javascript in an HTML page with the `<script>` tag

Example:

The snippet below hides all paragraphs.

```
<script>
   let all_p = document.querySelectorAll("p");
   for(p of all_p) p.style.display = "none";
</script>
```

1) Use a CSS selector to select all paragraphs in the document.

```
<script>
   let all_p = document.querySelectorAll("p");
   for(p of all_p) p.style.display = "none";
</script>
```

2) Remove all paragraphs from view.

```
<script>
  let all_p = document.querySelectorAll("p");
  for(p of all_p) p.style.display = "none";
</script>
```

Back to our webpage

Let’s go back to the webpage we have created.

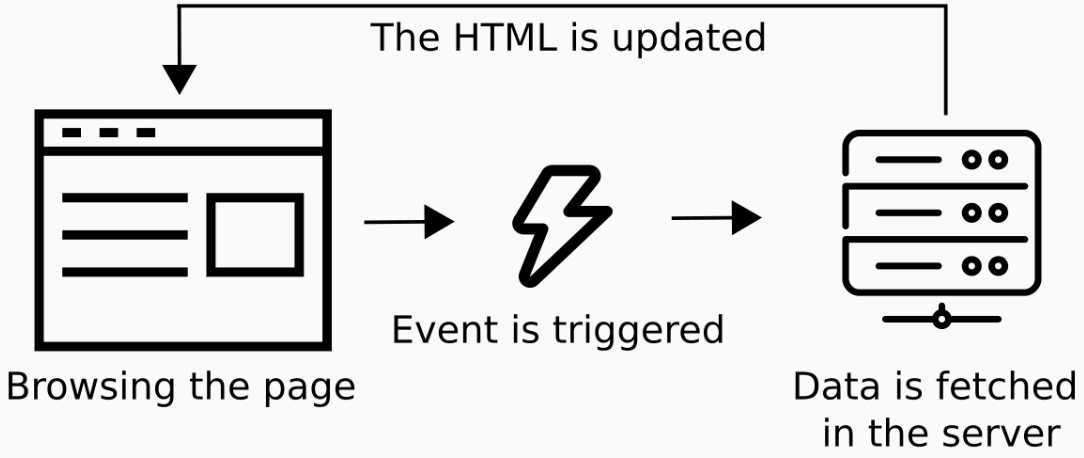
Add the following code which adds a buttom and attaches an event to the button. The event will be triggered when the button is clicked.

When the button is clicked, it will be replaced with a paragraph revealing the author.

```
<button type="button" id="btn"> What is my favourite author?</button>

<script>
   let btn = document.querySelector("#btn");

   showAuthor = function(){
       let p = document.createElement("p");
       p.innerHTML = "My favourite author is Shakespeare";
       this.replaceWith(p);
   }

   btn.addEventListener("click", showAuthor);
</script>
```

What happened?

- the favorite author, Shakespeare, is not visible on the webpage.
- the only way for it to appear in the HTML is after an event is triggered (here the click)
- that's how dynamic web pages work! In our example, the information was in the source code, but in general it is fetched from the server.



Dynamic webpages: Can we scrape them?

Yes, but...

Q: What do we need? and I hope the answer will be natural after this long introduction!

Dynamic webpages: Can we scrape them?

Yes, but...

Q: What do we need? and I hope the answer will be natural after this long introduction!

A: Indeed! We need to run javascript on the source code, and keep it running as the page updates.

In other words...

We need a web browser.

Python + Selenium

Requirements

- to scrape dynamic webpages, we'll use python in combination with selenium
- you need:
  - install selenium in python using `pip install selenium` on the terminal
  - download the appropriate driver to the browser (Chrome or Firefox) and put it on the path or in your WD

Checking the install

If the installation is all right, the following code should open a browser:

```
from selenium import webdriver
driver = webdriver.Chrome()
```

How does selenium works?

- selenium controls a browser: typically anything that you can do, it can do
- most common actions include:
  - access to URLs
  - clicking on buttons
  - typing/filling forms
  - scrolling
  - do you really do more than that?★

★: Selenium can do much more actually, it can even execute custom javascript code.

Selenium 101: An example

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("https://stackoverflow.com/")

btn_cookies = driver.find_element(By.CSS_SELECTOR, "button.js-accept-cookies")
btn_cookies.click()

search_input = driver.find_element(By.CSS_SELECTOR, "input.s-input__search")
search_input.send_keys("webscraping")
search_input.send_keys(Keys.RETURN)
```

Importing only the classes we’ll use.

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("https://stackoverflow.com/")

btn_cookies = driver.find_element(By.CSS_SELECTOR, "button.js-accept-cookies")
btn_cookies.click()

search_input = driver.find_element(By.CSS_SELECTOR, "input.s-input__search")
search_input.send_keys("webscraping")
search_input.send_keys(Keys.RETURN)
```

Launching the browser empty at the moment.

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("https://stackoverflow.com/")

btn_cookies = driver.find_element(By.CSS_SELECTOR, "button.js-accept-cookies")
btn_cookies.click()

search_input = driver.find_element(By.CSS_SELECTOR, "input.s-input__search")
search_input.send_keys("webscraping")
search_input.send_keys(Keys.RETURN)
```

Accessing the stackoverflow★ URL.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("https://stackoverflow.com/")

btn_cookies = driver.find_element(By.CSS_SELECTOR, "button.js-accept-cookies")
btn_cookies.click()

search_input = driver.find_element(By.CSS_SELECTOR, "input.s-input__search")
search_input.send_keys("webscraping")
search_input.send_keys(Keys.RETURN)
```

★: I'm sorry to target stackoverflow for webscraping but it's only for instructional purposes!

It's our first visit on the page, so cookies need to be agreed upon. After selecting★ the button to click with a CSS selector, we click on it with the `click()` method.

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("https://stackoverflow.com/")

btn_cookies = driver.find_element(By.CSS_SELECTOR, "button.js-accept-cookies")
btn_cookies.click()

search_input = driver.find_element(By.CSS_SELECTOR, "input.s-input__search")
search_input.send_keys("webscraping")
search_input.send_keys(Keys.RETURN)
```

★: Do not mistaken `find_element` with `find_elements` (the s!). The former returns an HTML element while the latter returns an array.

Finally we search the SO posts containing the term webscraping. We first select the input element containing the search text. Then we type webscraping with the `send_keys()` method and end with pressing enter (`Keys.RETURN`).

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("https://stackoverflow.com/")

btn_cookies = driver.find_element(By.CSS_SELECTOR, "button.js-accept-cookies")
btn_cookies.click()

search_input = driver.find_element(By.CSS_SELECTOR, "input.s-input__search")
search_input.send_keys("webscraping")
search_input.send_keys(Keys.RETURN)
```

Saving the results

To obtain the HTML of an element:

```
body = driver.find_element(By.CSS_SELECTOR, "body")
body.get_attribute("innerHTML")
```

The variable `driver.find_element(By.CSS_SELECTOR, "body").get_attribute("innerHTML")`★ contains the HTML code as it is currently displayed in the browser. It has nothing to do with the source code!

★: Please remind that the term `driver` is only a generic name which was taken from the previous example. It could be anything else.

Saving the results II

To write the HTML in a file, you can still do it Python way:

```
raw_html = body.get_attribute("innerHTML")
soup = BeautifulSoup(page.text, 'html.parser')
with open("tmp.html", "w", encoding="utf-8") as f:
  f.write(soup.prettify())
```

Dynamic webpages: is that it?

Well, that’s it folks!

You just have to automate the browser and save the results.

Then you can do the data processing in your favorite language.

Conclusion

I hope this short workshop has clarified some key concepts in webscraping.

Thanks and have fun!