

Econom'IA Workshop

Webscraping, an introduction

Laurent Bergé

University of Bordeaux, BSE

6 May 2024

What is web scraping?

- web scraping 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!

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. 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	not fun	need to plan well
dynamic webpage	requires web knowledge	fun

Objective

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

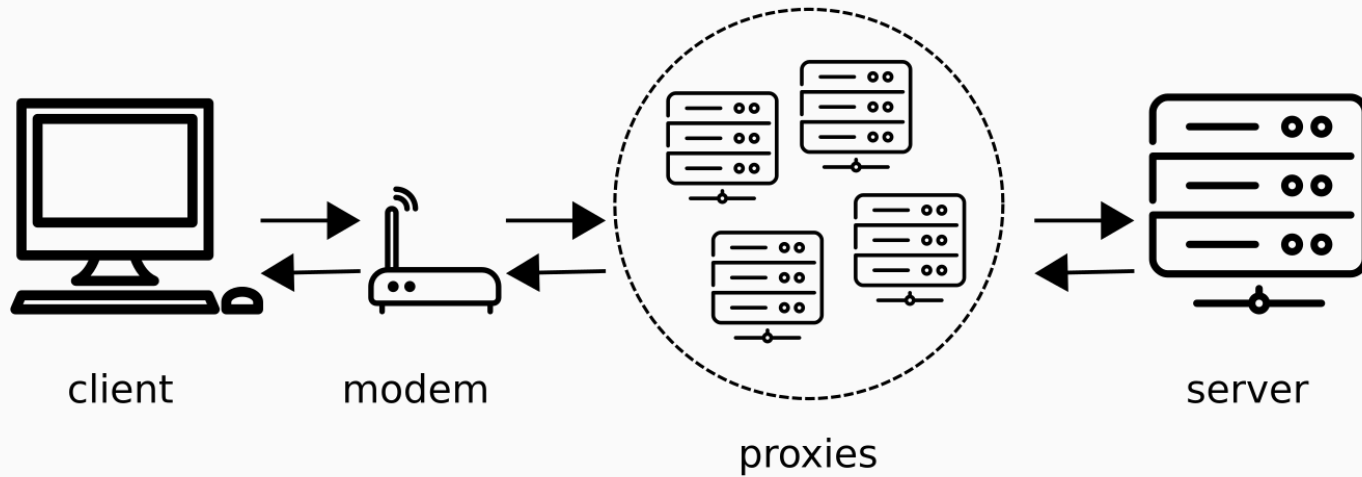
So you'll need to have a correct understanding of how the web works!

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



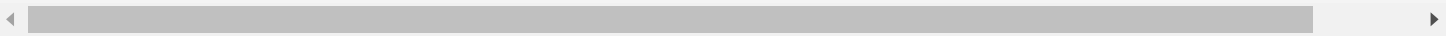
Example of HTTP GET request: The header you send to the webpage

```
GET HTTP/1.1  
Host: developer.mozilla.org  
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)  
Accept-Language: en-us  
Accept-Encoding: gzip, deflate  
Connection: Keep-Alive
```

Server sends back a response

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web
```



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

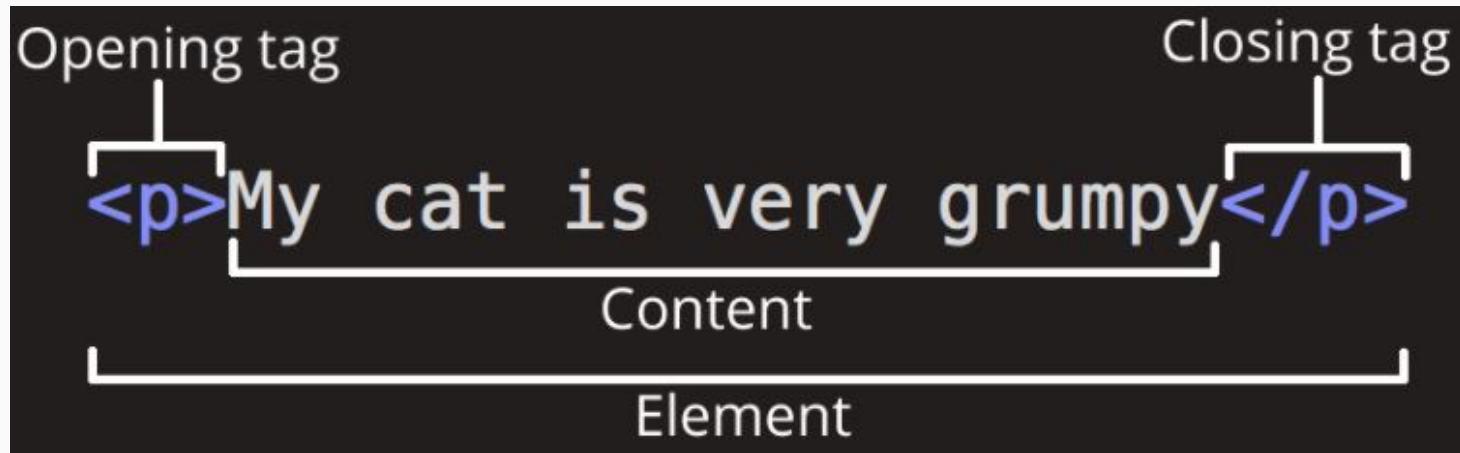
-  let's inspect the Members of the European Parliament
<https://www.europarl.europa.eu/meps/en/full-list>
- right click anywhere on the page and click **View page source** (on Chrome/Firefox)

How does HTML work?

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

... that's basically it!

How does HTML work? Tags



How does HTML work? Attributes



The diagram illustrates an HTML attribute within a code snippet. The code is `<p class="editor-note">My cat is very grumpy</p>`. A bracket above the code spans the `class="editor-note"` portion, with the word "Attribute" written above the bracket. The code is color-coded: `<p` is blue, `class="editor-note"` is yellow, `>` is blue, `My cat is very grumpy` is white, and `</p>` is blue.

```
<p class="editor-note">My cat is very grumpy</p>
```


Empty tags

Some tags don't need closing tags:

- like `` or `
`

Most common tags

- `h1` – `h4`: headers
- `p`: paragraph
- `a`: link
- `img`: image
- `strong`: to emphasize text
- `div`: generic box (`this may be the most popular`)

HTML: Example

🖱️ let's write our first web page!

- create the file `index.html` and open it with the editor
- fill the page according to the following template:

```
<!DOCTYPE html>
<body>
<h1> John Doe's webpage </h1>
<h4> Short presentation </h4>
<p> Briefly 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>

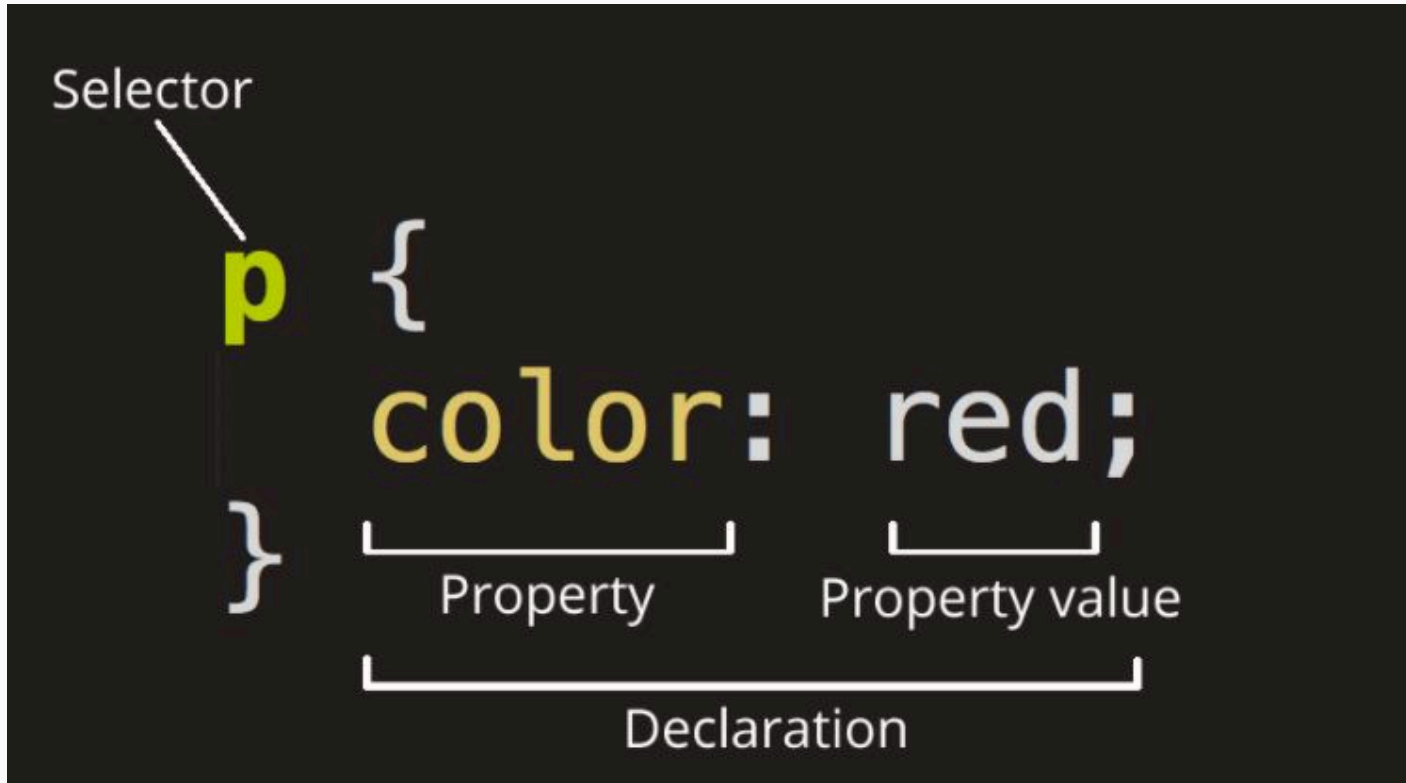
<a href='URL'> add links with the 'a' tag</a>
</body>
```

Yeah: we have created a webpage!

- but, wait a minute.. *why is our webpage so boring?*
- HTML is only about content, **not about style**
- why are others' webpages so fancy?
- HTML is nothing without its best friend **CSS**
- CSS is *only about style* => Let's look at CSS!

How does CSS work?

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



You can do a lot with CSS!



Source: [Adam Kuhn](#), with only HTML and CSS.

CSS: Example

- 🖱️ let's add CSS to our web page!
- we can add CSS using the HTML's `<style>` tag in the header of the HTML document:

```
<!DOCTYPE html>
<head>
  <style>
    /* To increase the font size and change the font family */
    /* + change the background color */
    /* We apply this to the <body> tag */
    body {
      font-size: 15pt;
      font-family: sans-serif;
      background-color: linen;
    }

    /* We add a border radius and a box shadow to the <img> */
    img {
      border-radius: 6px;
      box-shadow: 12px 12px 3px 2px rgba(0,0,255,0.3);
    }
  </style>
</head>
<body>
  etc...
```

CSS example: References

Some references for the previous example:

- font-size
- font-family
- background-color
- There are 140 predefined colors in HTML.
- border-radius
- box-shadow

CSS: Can we do more?

I would like to have:

- the first paragraph in italic
- the stuff that I like in green (**ForestGreen**)
- the stuff that I dislike in red (**Crimson**)

Q: At the moment, can I do that?

A: Not yet, because we need to select precisely some elements! In sum, 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 ways to select HTML elements

```
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 sibl
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 lc
etc!
```


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 divs.

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
lis.

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>
```

- 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
```

A note on classes: `[attr]` selection

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

- even though you can select with `p.main`, you **cannot** with `p[class^=main]`
- in `p[class^=text]` only the full string of the class is considered, not the three classes separately
- you would have to use `p[class*=main]`

Caveat...

- you would also end up selecting the following element:

```
<p class="mainiac"> On the floor. </p>
```

Selectors: XPath

- **XPath** is a language to make selections in XML documents (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[☆]
- 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.

[☆]: 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 + exercise

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

Resources

- [CSS](#)
- [XPath](#)

Exercise

Let's play a bit with a [website](https://lrberge.github.io/learn_selectors) I created to learn and test selectors:
https://lrberge.github.io/learn_selectors

More CSS to our webpage

🖱️ let's add more CSS to our webpage:

- the text of the first paragraphs in italic (use `font-style: italic;`)
- 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`)

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**)!

Basic web scraping in python

Good news

You have readily available tools to webscrape in Python.

In python, you'll need:

- requests
- BeautifulSoup
- and that's it! (for the easy stuff!)

Exercise 1: The European Parliament

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

The only thing you need is...

```
# 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 browsers!**

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

1: click on this icon

3: you've found the div!

2: go here with the mouse and click

Full list

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

Asim ADEMOV
Group of the European People's Party (Christian Democrats)
Bulgaria
Citizens for European Development
Bulgaria

Alex AGIUS SALIBA
Group of the Progressive Alliance of Socialists and Democrats in the European Parliament

Mazaly AGUILAR
European Conservatives and Reformists Group
Spain

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 `.prettyfy()` method from `BeautifulSoup`.

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

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

Exercise 2

Find all dates of birth for French MEPs.

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

Example:

```
# Have a 1s pause  
import time  
time.sleep(1)
```

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?

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/LinkedIn/Twitter infinite scrolling**)
 - 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 button 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"> Who is my favourite author?</button>

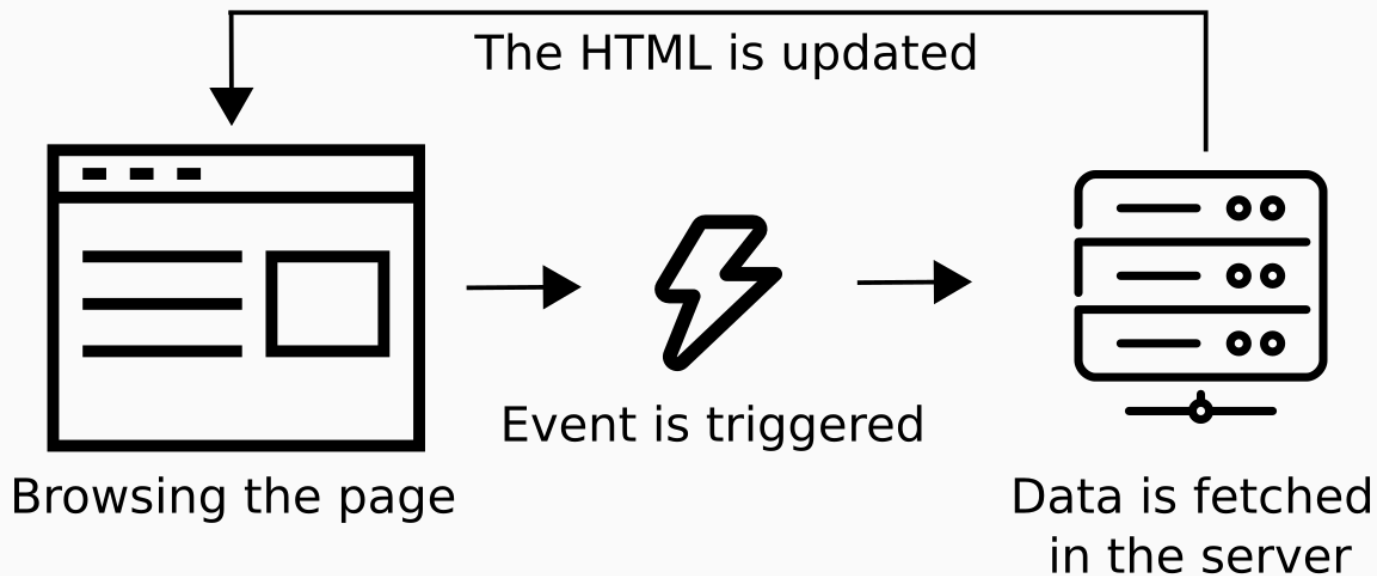
<script>
  let btn = document.querySelector("#btn");

  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!{star} 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!)

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 **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

```
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[id*=acce
btn_cookies.click()

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


Importing only the classes we'll use.

```
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_input.send_keys("web scraping")
search_input.send_keys(Keys.RETURN)
```

Launching the browser (empty at the moment).

```
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_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_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.

```
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_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`).

```
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_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

Now that you dispose of the HTML, you can use `BeautifulSoup` to parse it and do regular data management.

Random tips: Good to know

Scrolling in selenium (executes javascript)

```
# Scrolls down by 1000 pixels
driver.execute_script("window.scrollTo(0,1000)")

# Goes at the top of the page
driver.execute_script("window.scrollTo(0, 0)")
```


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!