

XPath Navigation

WEB SCRAPING IN PYTHON



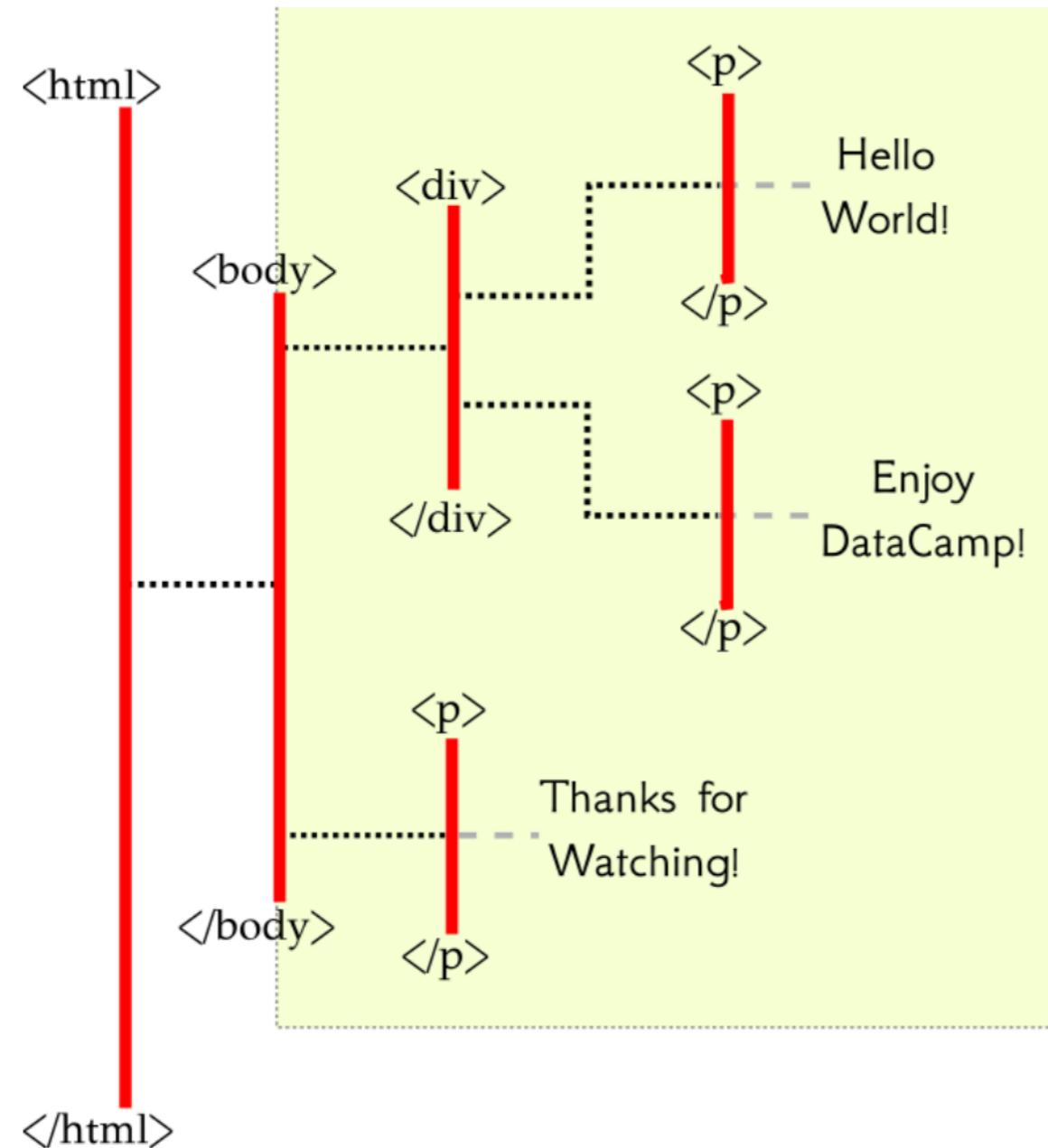
Thomas Laetsch

Data Scientist, NYU

Slashes and Brackets

- Single forward slash / looks forward **one** generation
- Double forward slash // looks forward **all** future generations
- Square brackets [] help narrow in on specific elements

To Bracket or not to Bracket



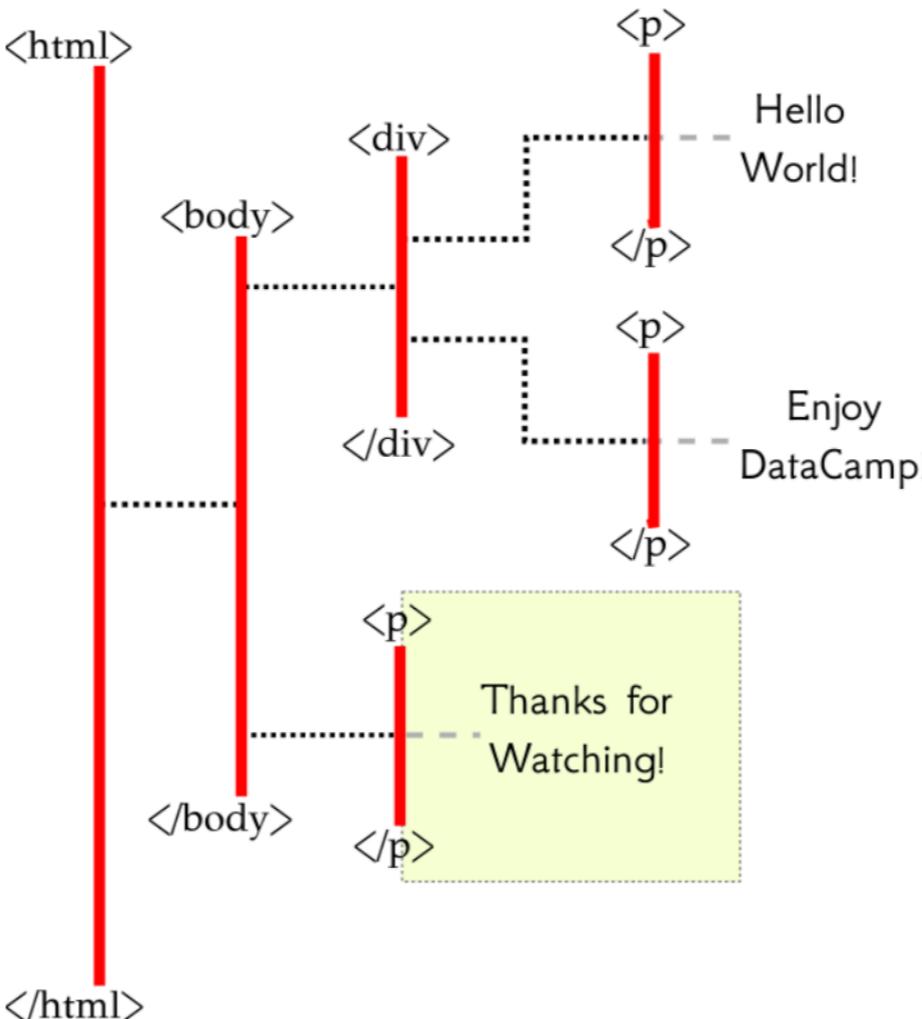
```
xpath = '/html/body'
```

```
xpath = '/html[1]/body[1]'
```

- Give the same selection

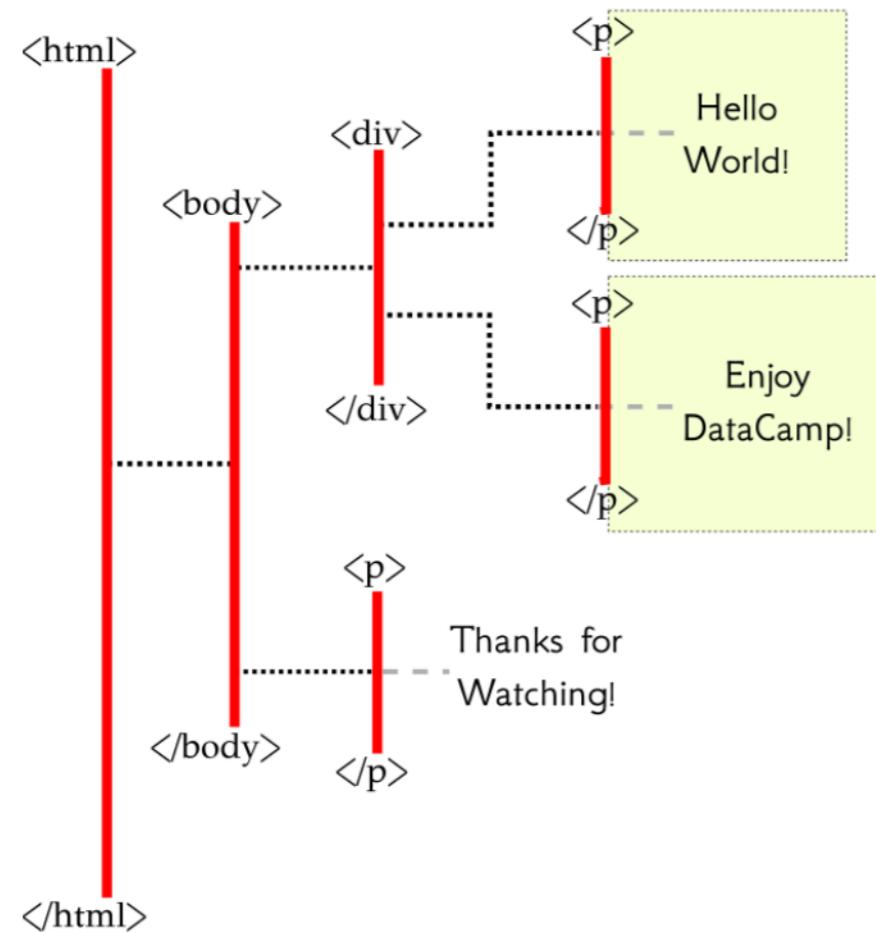
A Body of P

```
xpath = '/html/body/p'
```

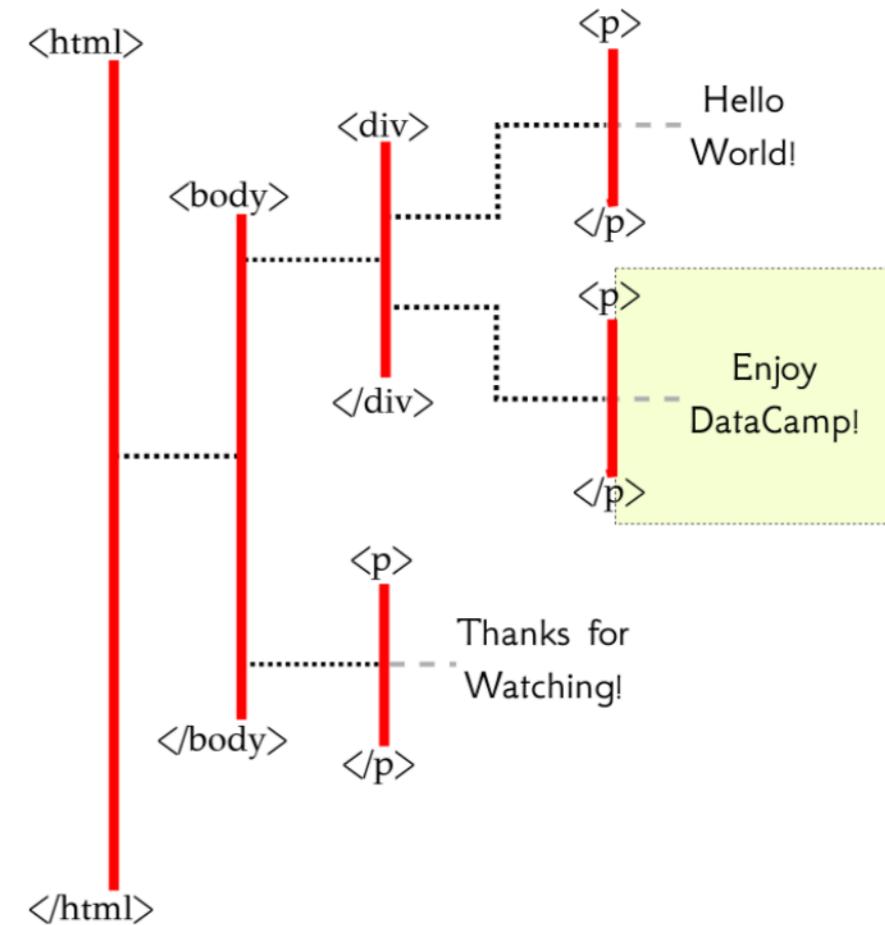


The Birds and the Ps

```
xpath = '/html/body/div/p'
```

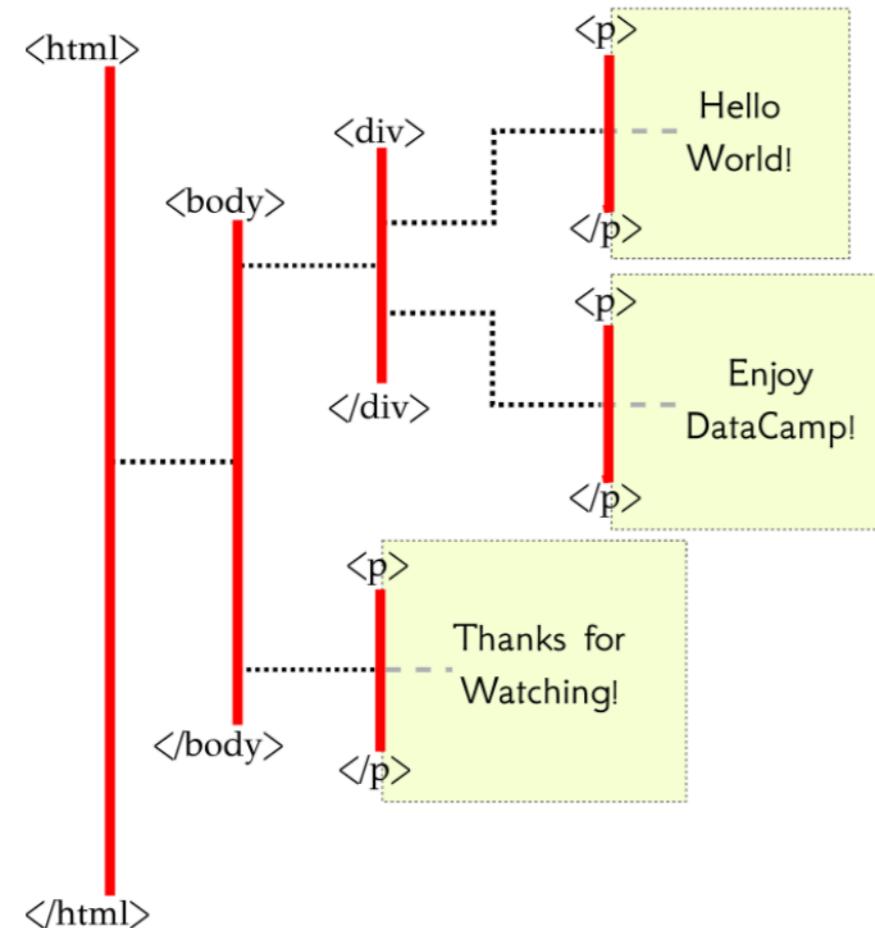


```
xpath = '/html/body/div/p[2]'
```

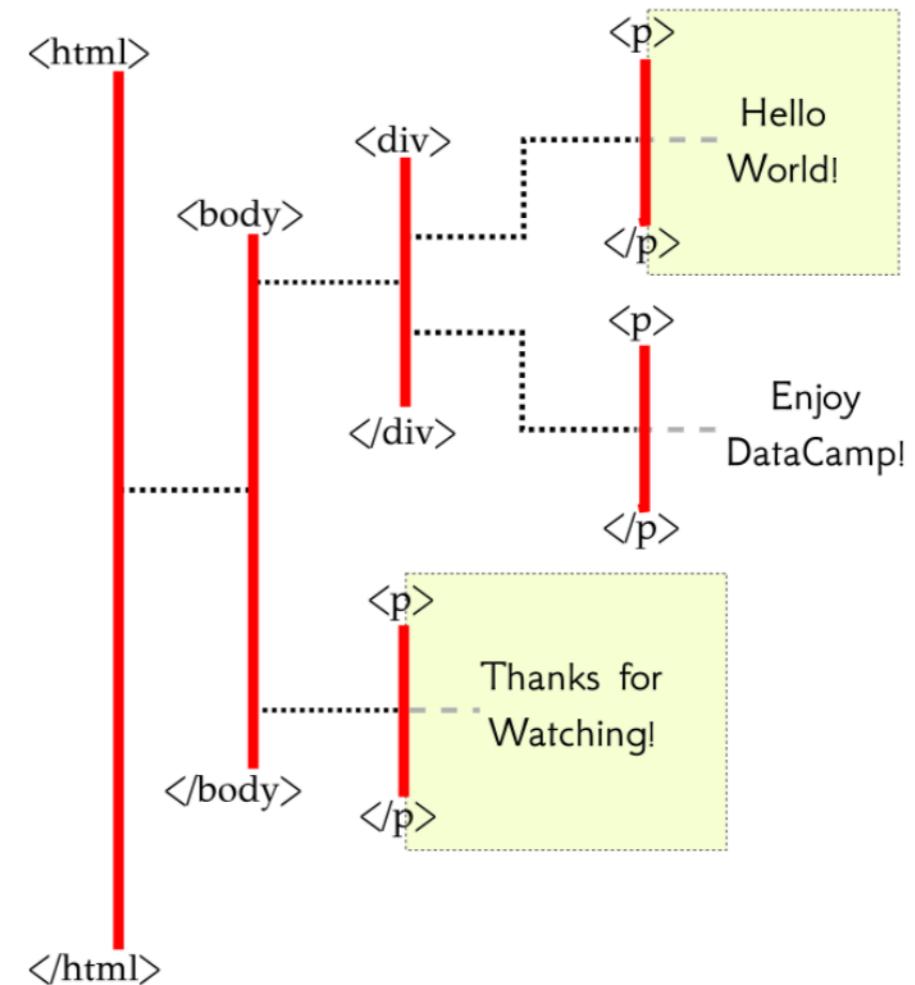


Double Slashing the Brackets

```
xpath = '//p'
```



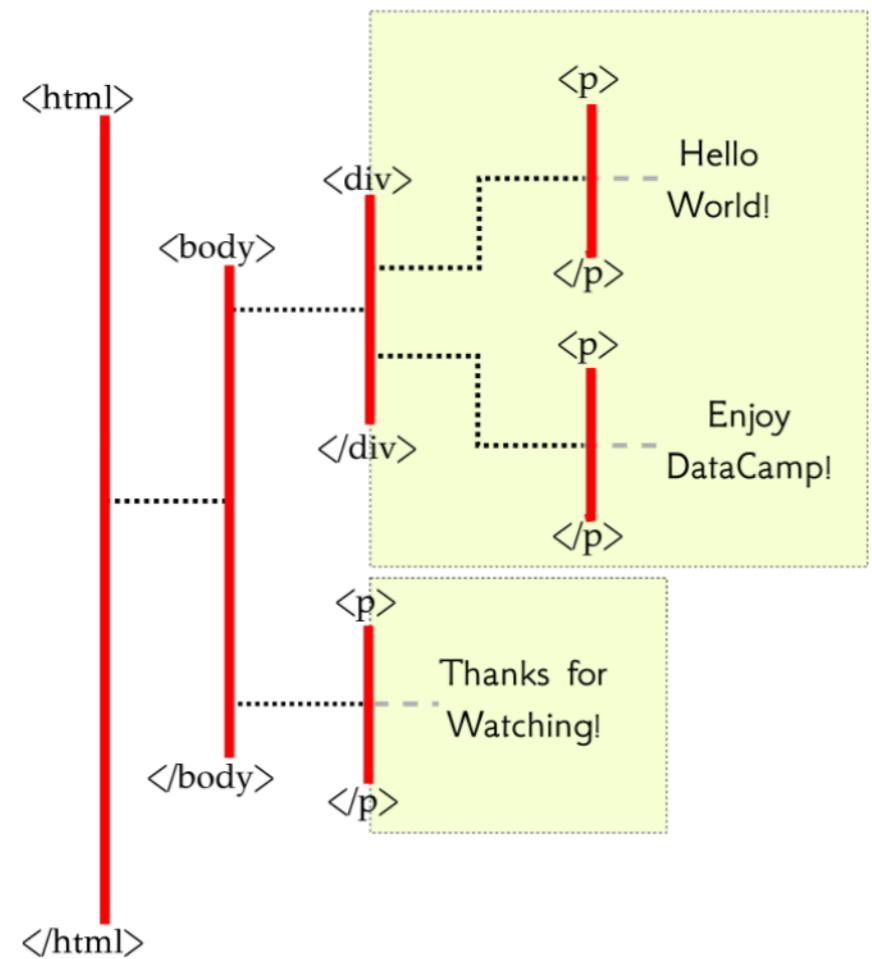
```
xpath = '//p[1]'
```



The Wildcard

```
xpath = '/html/body/*'
```

- The asterisks * is the "wildcard"



Xposé

WEB SCRAPING IN PYTHON

Off the Beaten XPath

WEB SCRAPING IN PYTHON

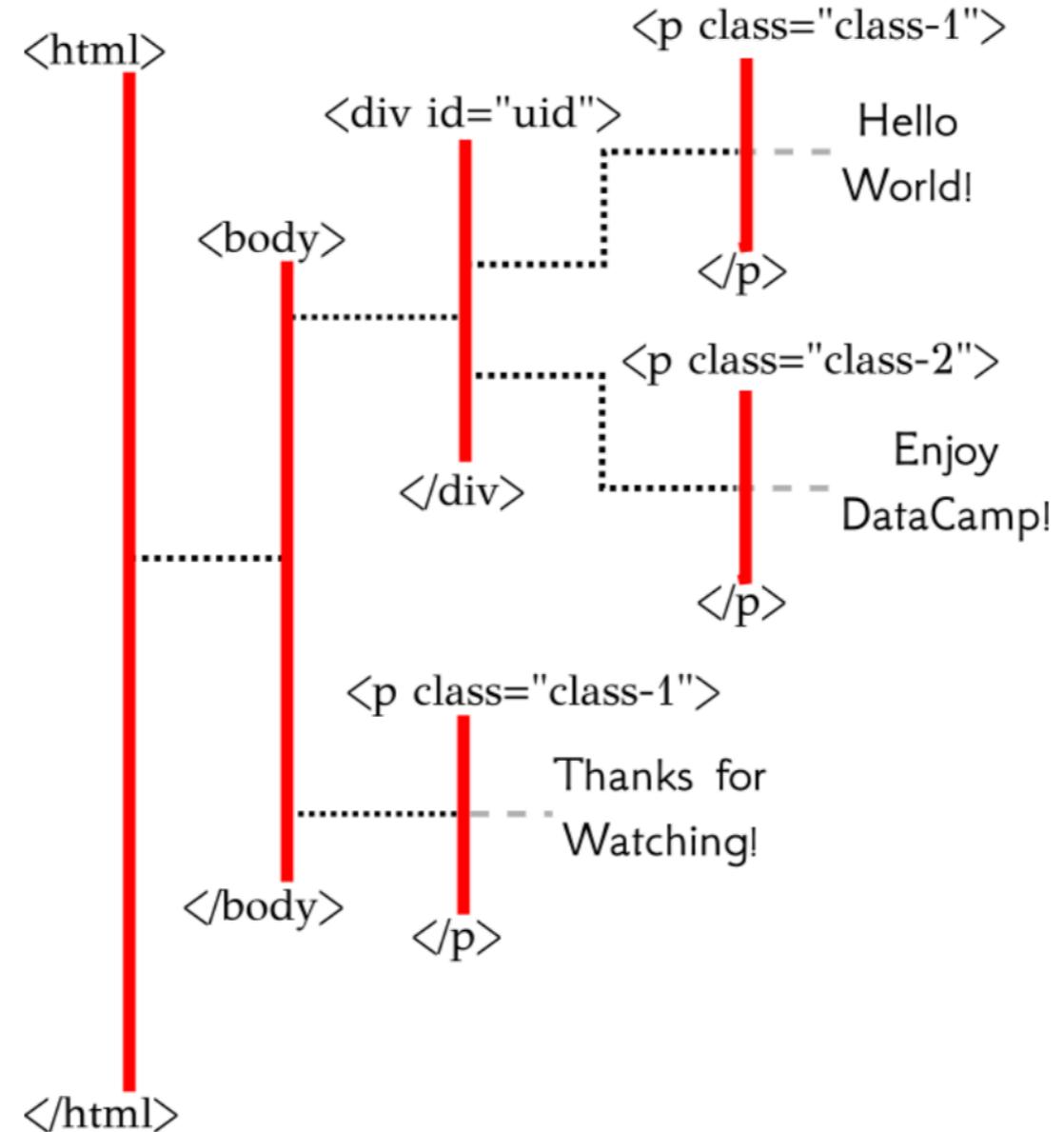


Thomas Laetsch
Data Scientist, NYU

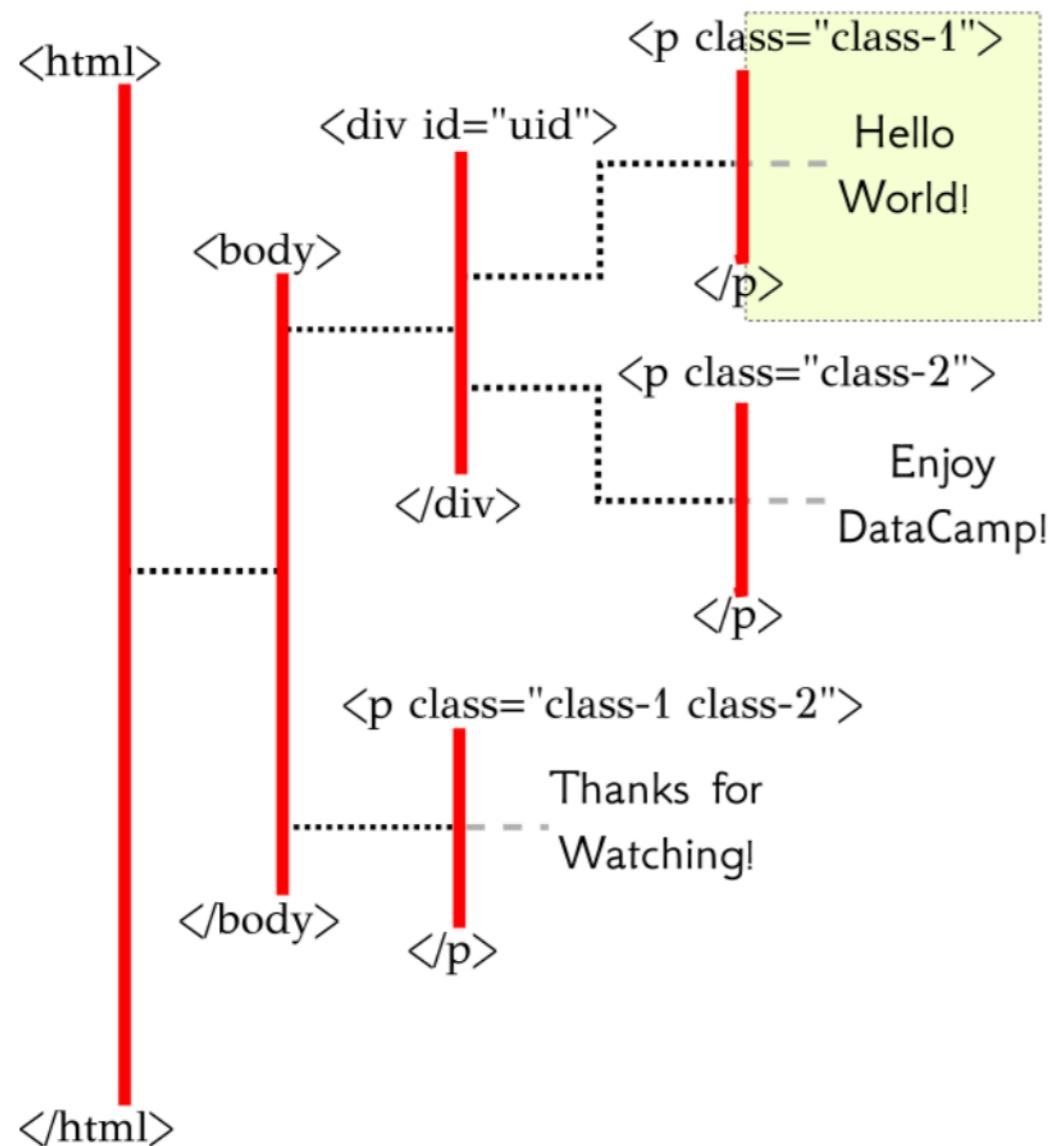
(At)tribute

- @ represents "attribute"
 - @class
 - @id
 - @href

Brackets and Attributes

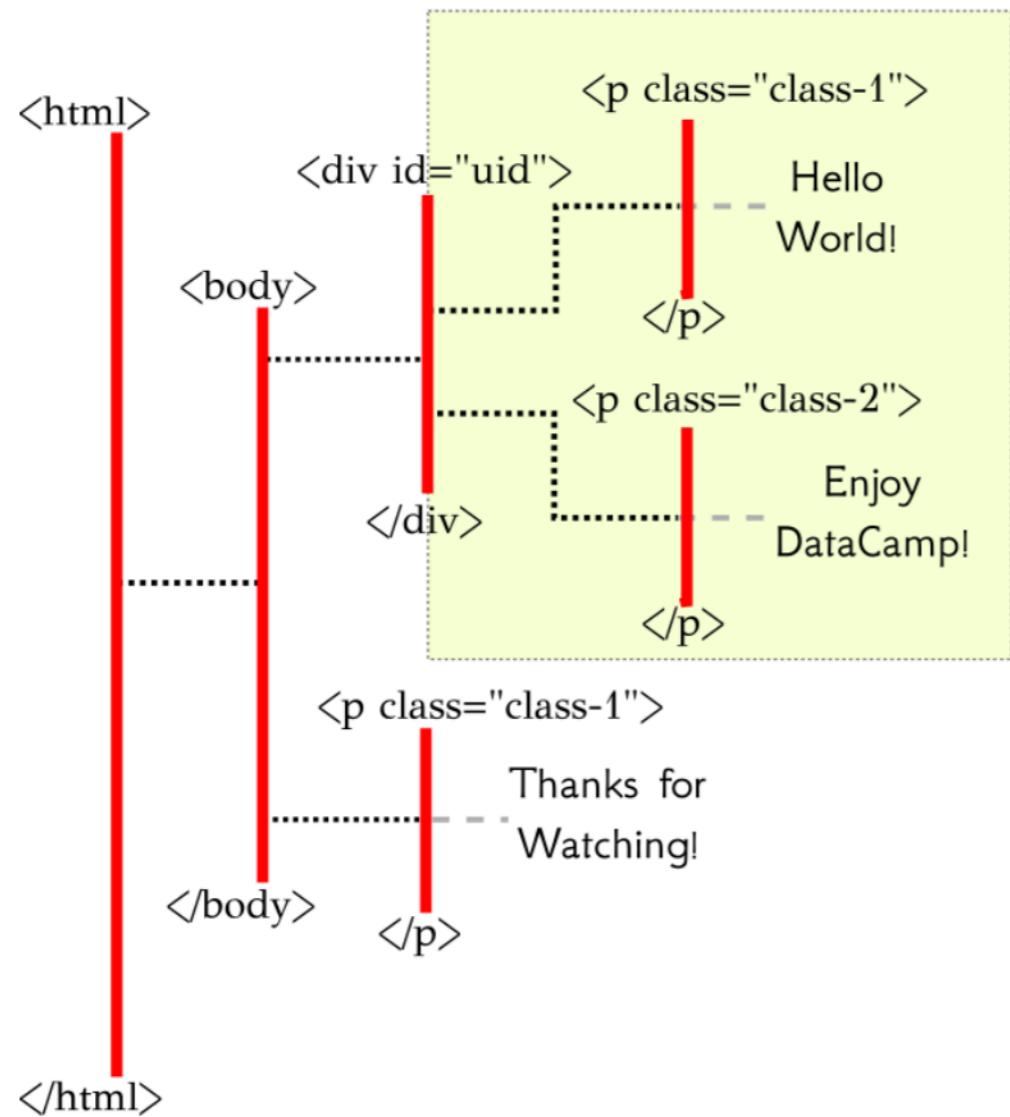


Brackets and Attributes



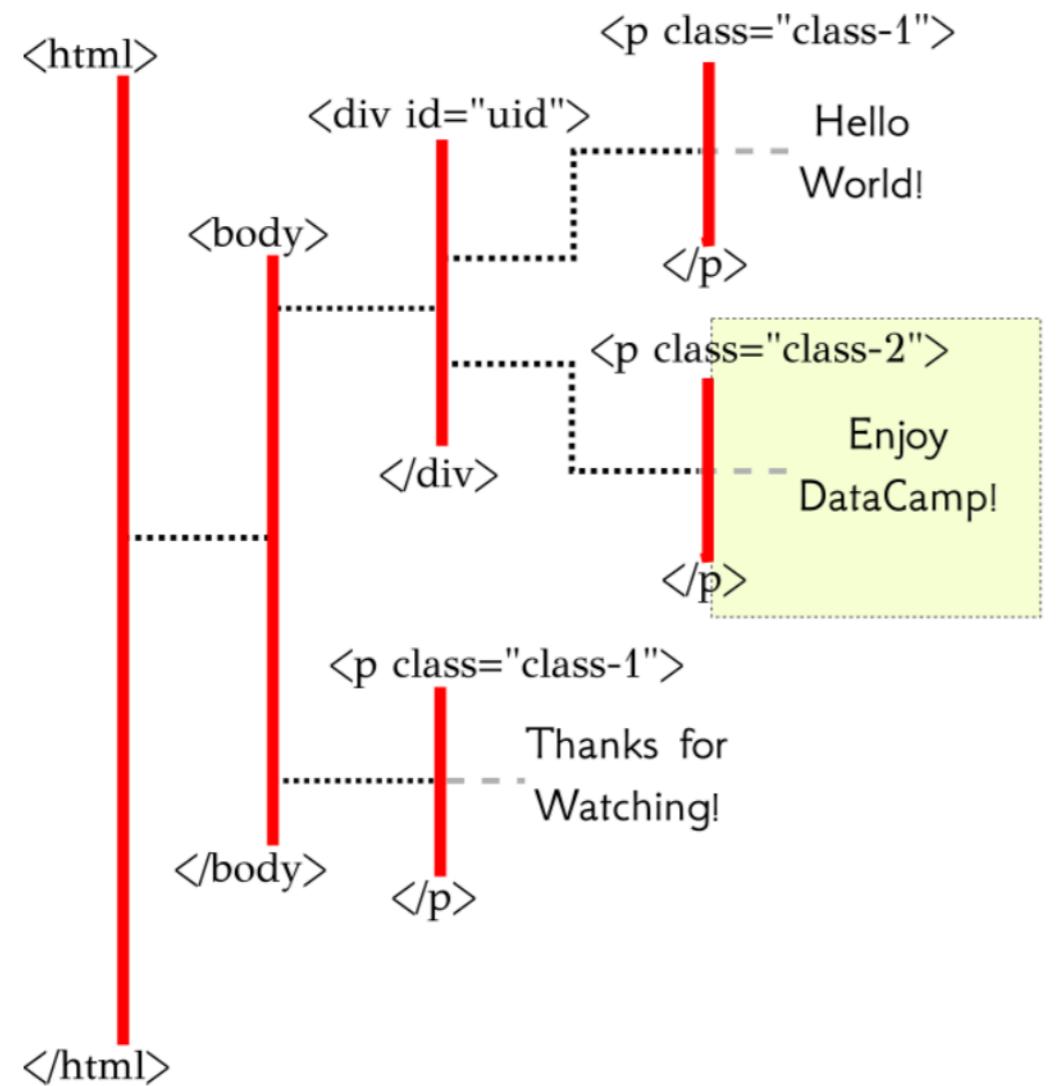
```
xpath = '//p[@class="class-1"]'
```

Brackets and Attributes



```
xpath = '//*[@id="uid"]'
```

Brackets and Attributes



```
xpath = '//div[@id="uid"]/p[2]'
```

Content with Contains

Xpath Contains Notation:

`contains(@attri-name, "string-expr")`

Contain This

```
xpath = '//*[contains(@class,"class-1")]'
```

 <p class="class-1"> ... </p>

 <div class="class-1 class-2"> ... </div>

 <p class="class-1 2"> ... </p>

Contain This

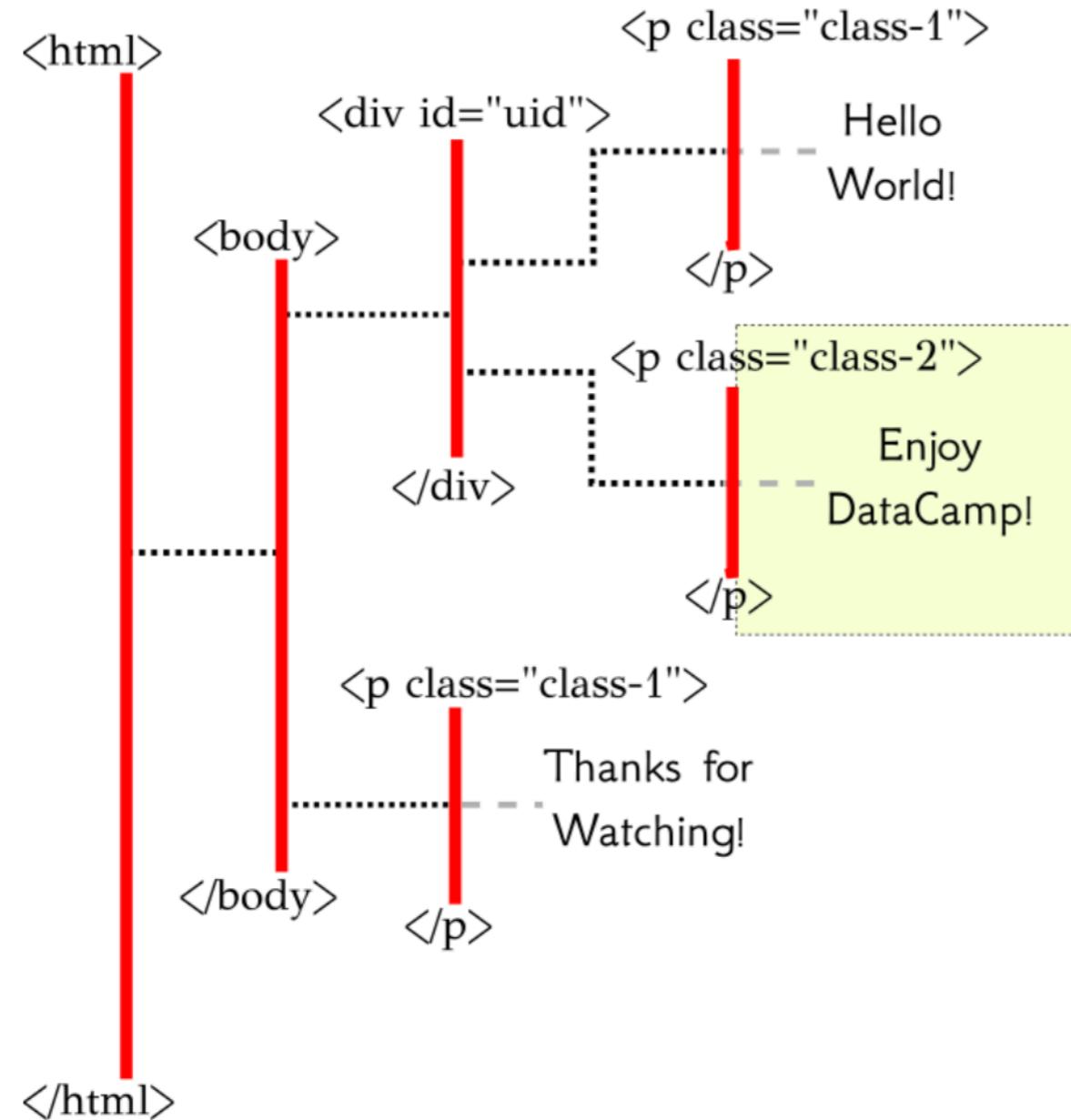
```
xpath = '//*[@class="class-1"]'
```

 <p class="class-1"> ... </p>

 <div class="class-1 class-2"> ... </div>

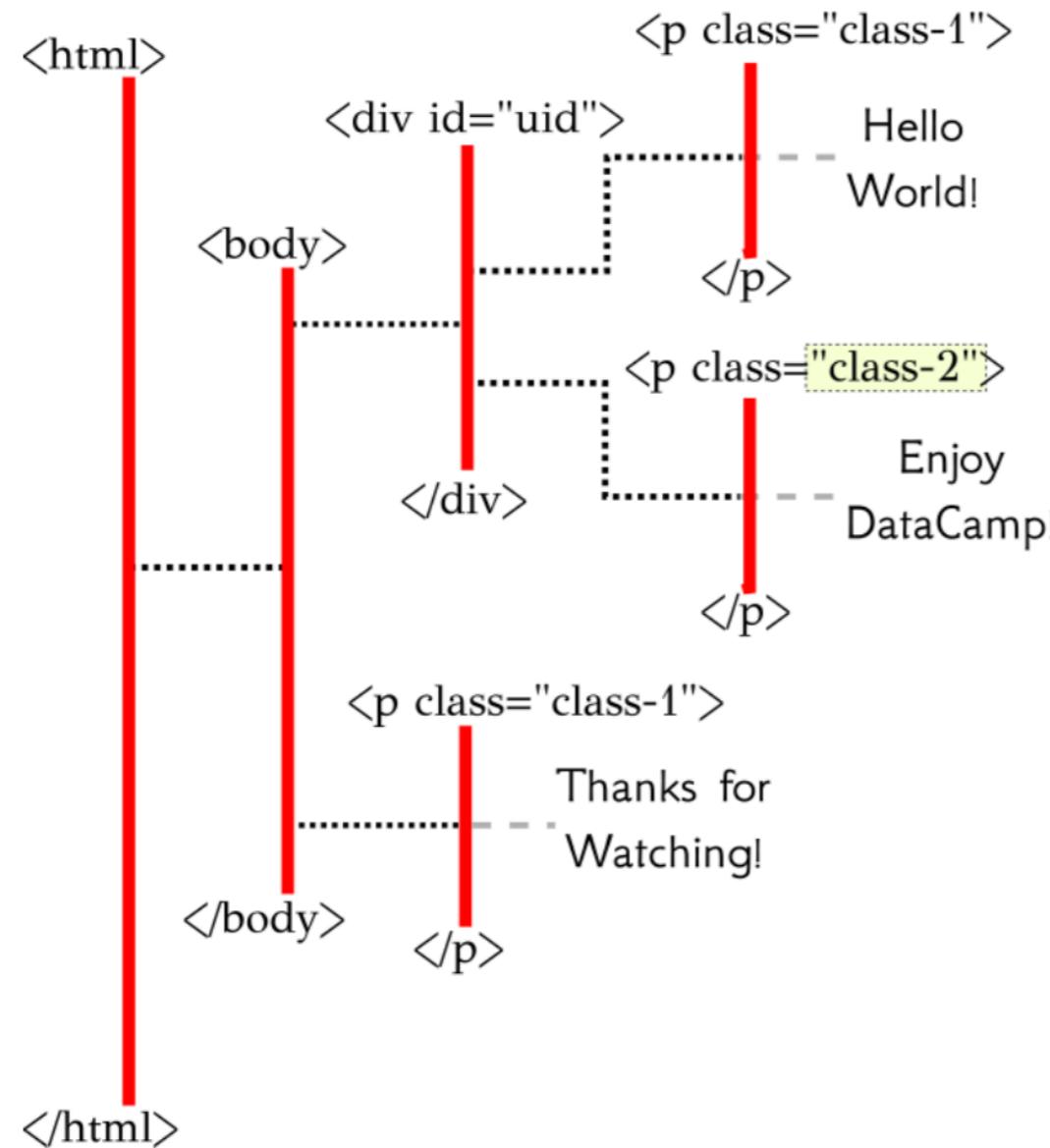
 <p class="class-1 2"> ... </p>

Get Classy



```
xpath = '/html/body/div/p[2]'
```

Get Classy



```
xpath = '/html/body/div/p[2]/@class'
```

End of the Path

WEB SCRAPING IN PYTHON

Introduction to the scrapy Selector

WEB SCRAPING IN PYTHON



Thomas Laetsch
Data Scientist, NYU

Setting up a Selector

```
from scrapy import Selector
```

```
html = '''
<html>
  <body>
    <div class="hello datacamp">
      <p>Hello World!</p>
    </div>
    <p>Enjoy DataCamp!</p>
  </body>
</html>
'''
```

```
sel = Selector( text = html )
```

- Created a scrapy Selector object using a string with the html code
- The selector `sel` has selected the **entire** html document

Selecting Selectors

- We can use the `xpath` call within a `Selector` to create new `Selector`s of specific pieces of the html code
- The return is a `SelectorList` of `Selector` objects

```
sel.xpath("//p")
# outputs the SelectorList:
[<Selector xpath='//p' data='<p>Hello World!</p>>,
 <Selector xpath='//p' data='<p>Enjoy DataCamp!</p>>]
```

Extracting Data from a SelectorList

- Use the `extract()` method

```
>>> sel.xpath("//p")
out: [<Selector xpath='//p' data='<p>Hello World!</p>'>,
      <Selector xpath='//p' data='<p>Enjoy DataCamp!</p>'>]
```

```
>>> sel.xpath("//p").extract()
out: [ '<p>Hello World!</p>',
      '<p>Enjoy DataCamp!</p>' ]
```

- We can use `extract_first()` to get the first element of the list

```
>>> sel.xpath("//p").extract_first()
out: '<p>Hello World!</p>'
```

Extracting Data from a Selector

```
ps = sel.xpath('//p')  
second_p = ps[1]
```

```
second_p.extract()  
out: '<p>Enjoy DataCamp!</p>'
```

Select This Course!

WEB SCRAPING IN PYTHON

"Inspecting the HTML"

WEB SCRAPING IN PYTHON



Thomas Laetsch, PhD

Data Scientist, NYU

"Source" = HTML Code

Web scraping - Wikipedia

https://en.wikipedia.org/wiki/Web_scraping

Not logged in Talk Contributions Create account Log in

Web scraping

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

Find sources: "Web scraping" – news · newspapers · books · scholar · JSTOR (June 2017) (Learn how and when to remove this template message)

For broader coverage of this topic, see [Data scraping](#).

Web scraping, **web harvesting**, or **web data extraction** is [data scraping](#) used for extracting data from websites. Web scraping software may access the [World Wide Web](#) directly using the [Hypertext Transfer Protocol](#), or through a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a [bot](#) or [web crawler](#). It is a form of copying, in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later [retrieval](#) or [analysis](#).

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for later analysis. Extraction is the process of extracting data from the fetched page. This may involve searching, reformatting, its data copied into a spreadsheet, and so on. Web scraping is often used to find and copy names and phone numbers from a website and store them in a database for further processing. An example would be to find and copy names and phone numbers from a website and store them in a database for further processing.

Web scraping is used for [contact scraping](#), and as a component of applications such as [price comparison](#), [product review scraping](#) (to watch the competition), [social media monitoring](#), [online presence and reputation](#), [web mashup](#) and, [web data integration](#).

[Web pages](#) are built using text-based mark-up languages ([HTML](#) and [XML](#)) designed for human end-users and not for ease of automated use. As a result, web pages are often complex and difficult to parse. Newer forms of web scraping involve listening to data feeds from web servers and the web server.

There are methods that some websites use to prevent web scraping, such as CAPTCHAs and robots.txt files. Web scraping systems that rely on using techniques in [DOM parsing](#), [computer vision](#) and [natural language processing](#) can extract data from these pages without being detected.

Page content for offline parsing.

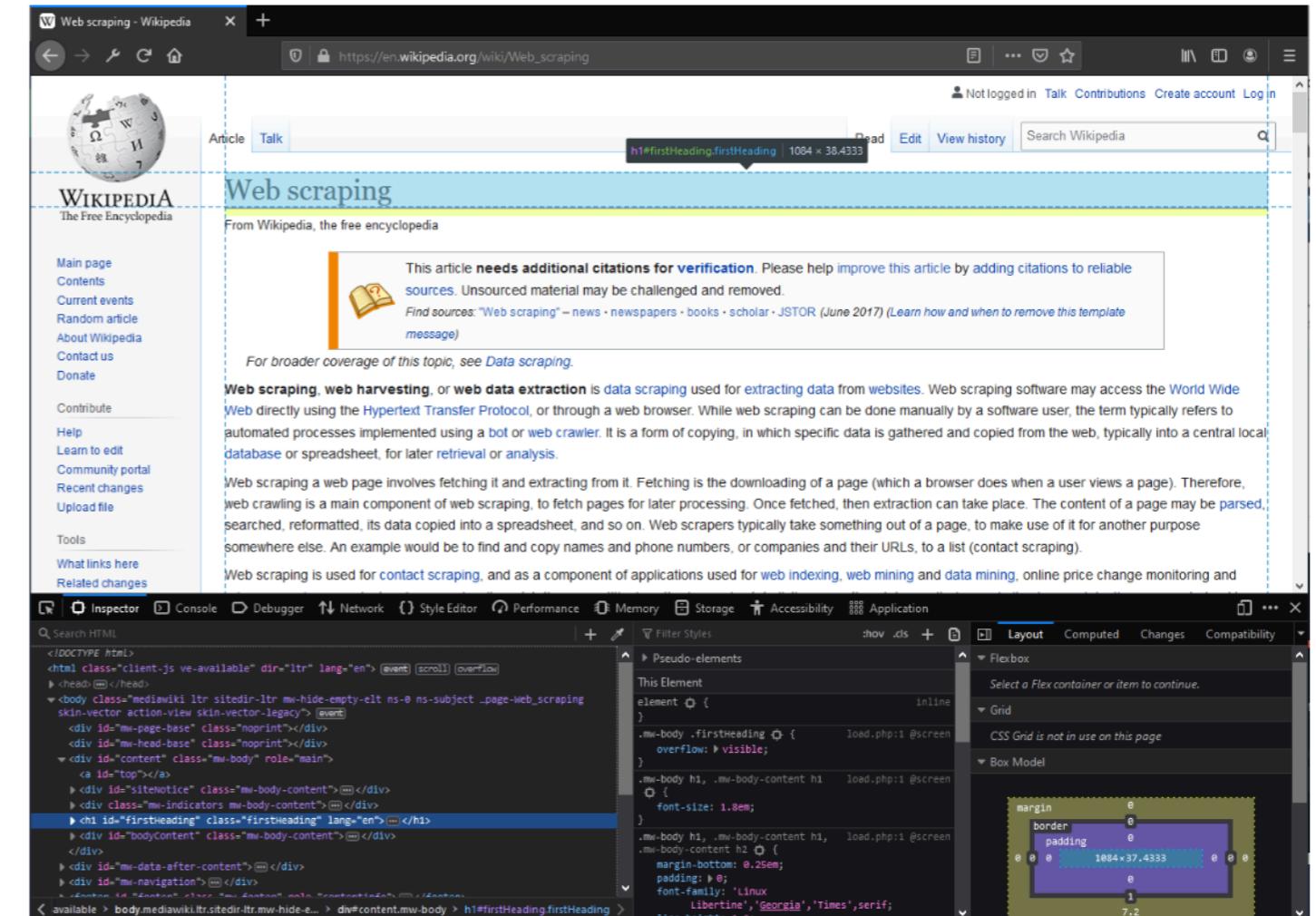
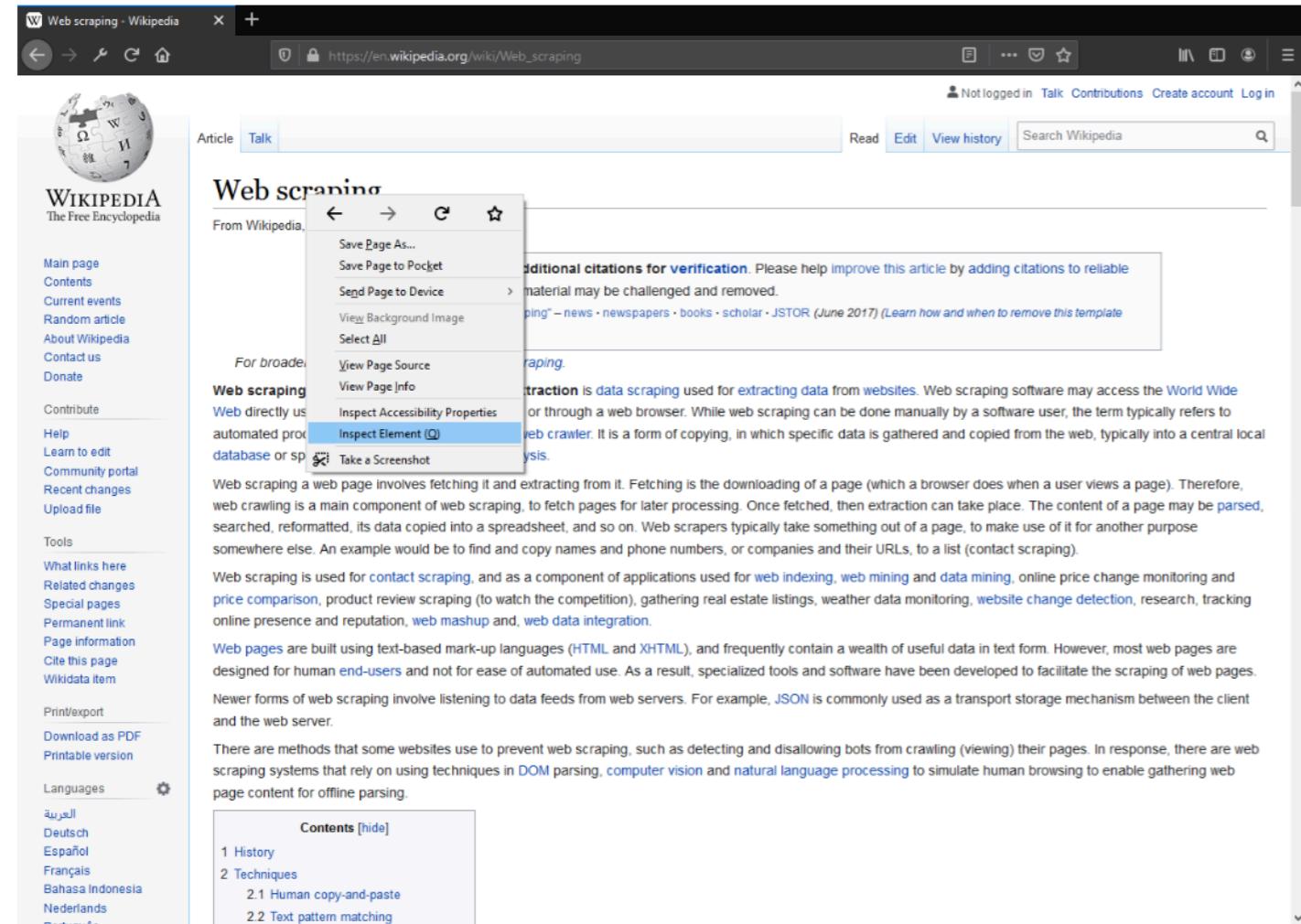
Contents [hide]

- 1 History
- 2 Techniques
 - 2.1 Human copy-and-paste
 - 2.2 Text pattern matching

Read Edit View history Search Wikipedia

Save Page As...
Save Page to Pocket
Send Page to Device
View Background Image
Select All
View Page Source
View Page Info
Inspect Accessibility Properties
Inspect Element (Q)
Take a Screenshot

Inspecting Elements



HTML text to Selector

```
from scrapy import Selector
```

```
import requests
url = 'https://en.wikipedia.org/wiki/Web_scraping'
html = requests.get( url ).content
```

```
sel = Selector( text = html )
```

You Know Our Secrets

WEB SCRAPING IN PYTHON