

# Web Scraping With Python

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Business Savvy

## What are businesses looking for?

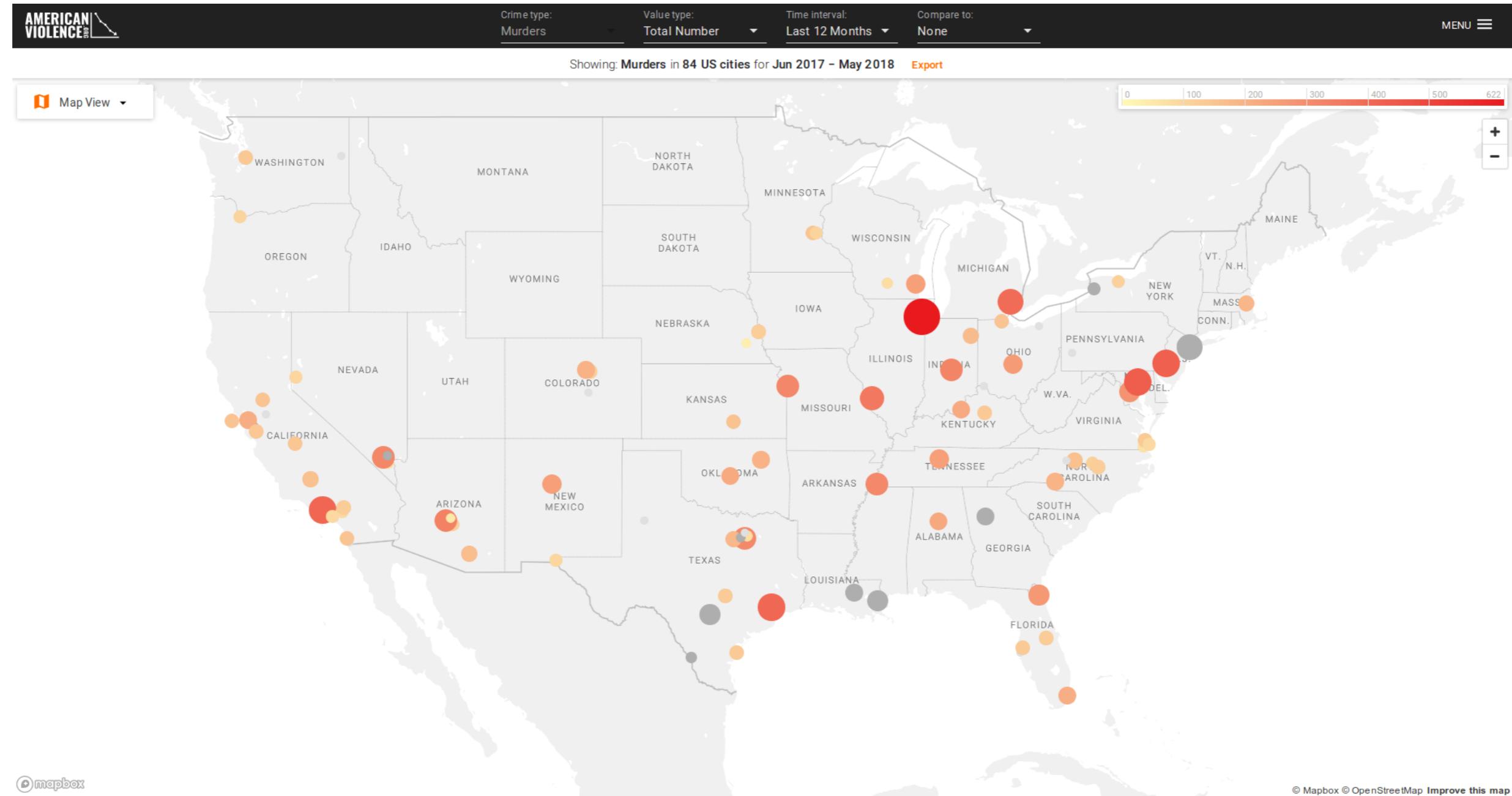
- Comparing prices
- Satisfaction of customers
- Generating potential leads
- ...and much more!

# It's Personal

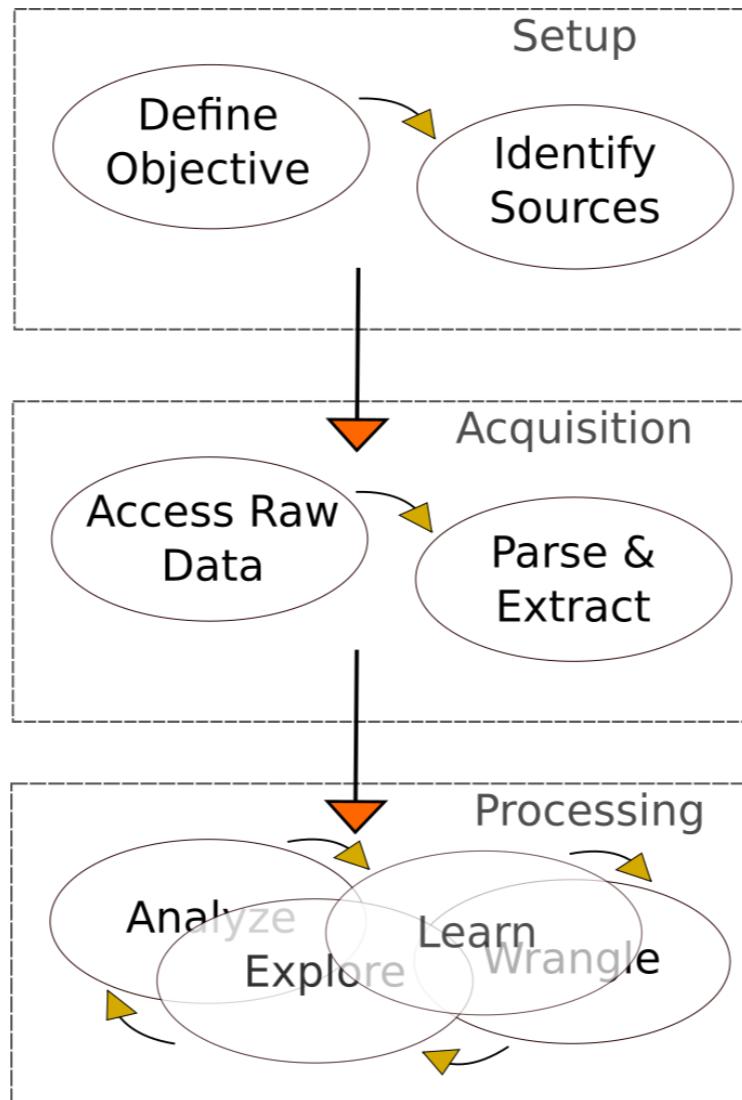
## What could you do?

- Search for your favorite memes on your favorite sites.
- Automatically look through classified ads for your favorite gadgets.
- Scrape social site content looking for hot topics.
- Scrape cooking blogs looking for particular recipes, or recipe reviews.
- ...and much more!

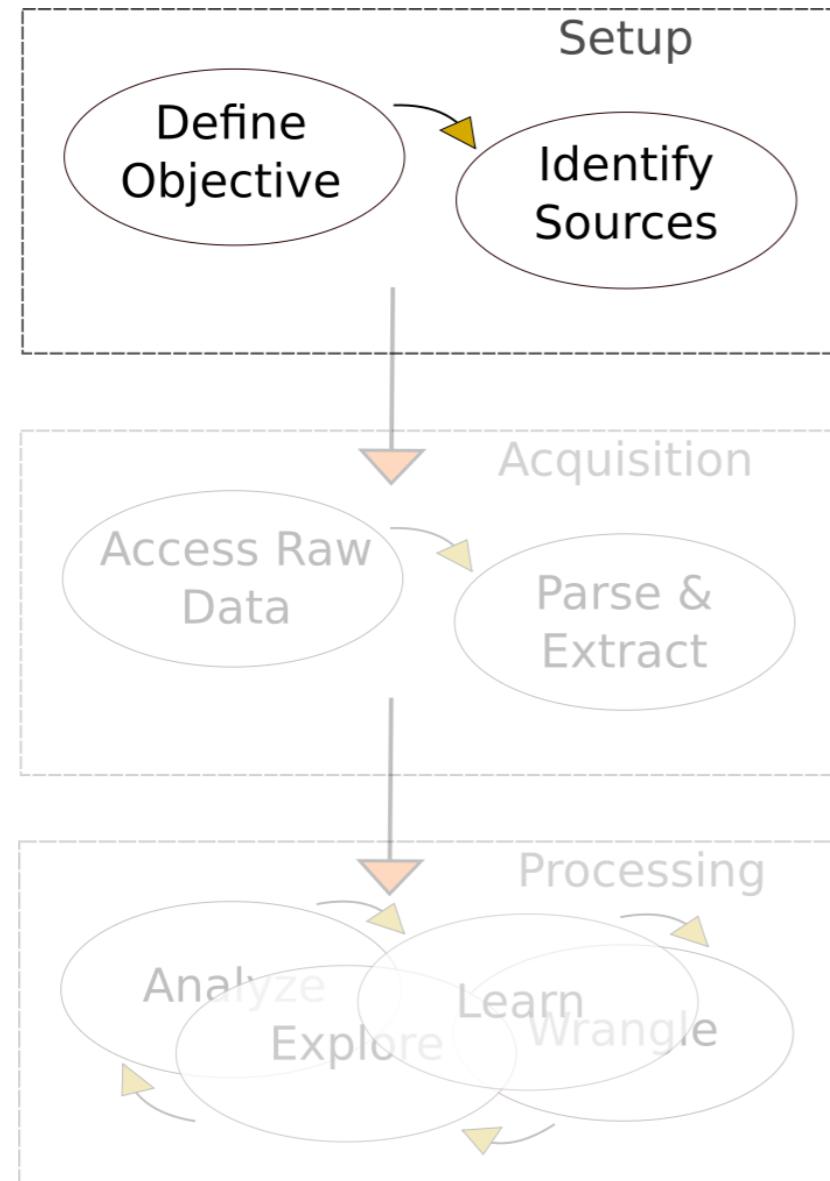
# About My Work



# Pipe Dream



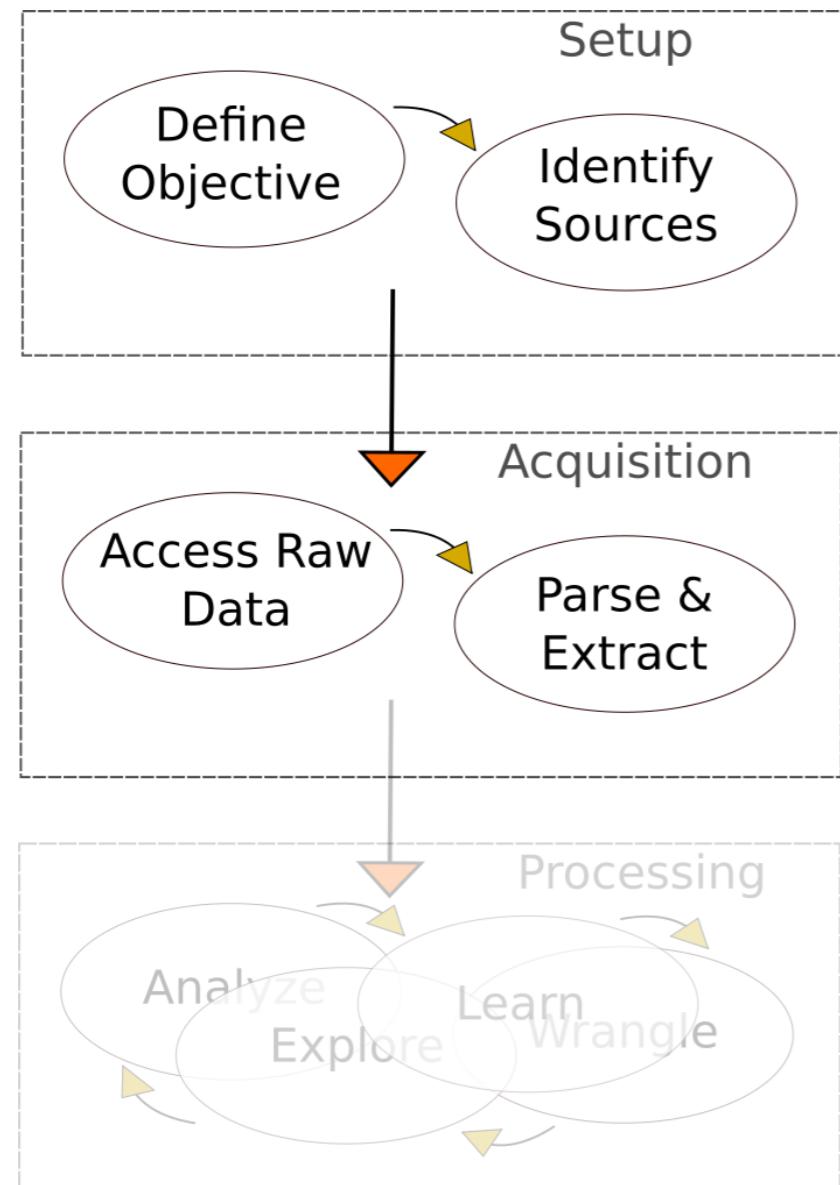
# Pipe Dream: Setup



## Setup

- Understand what we want to do.
- Find sources to help us do it.

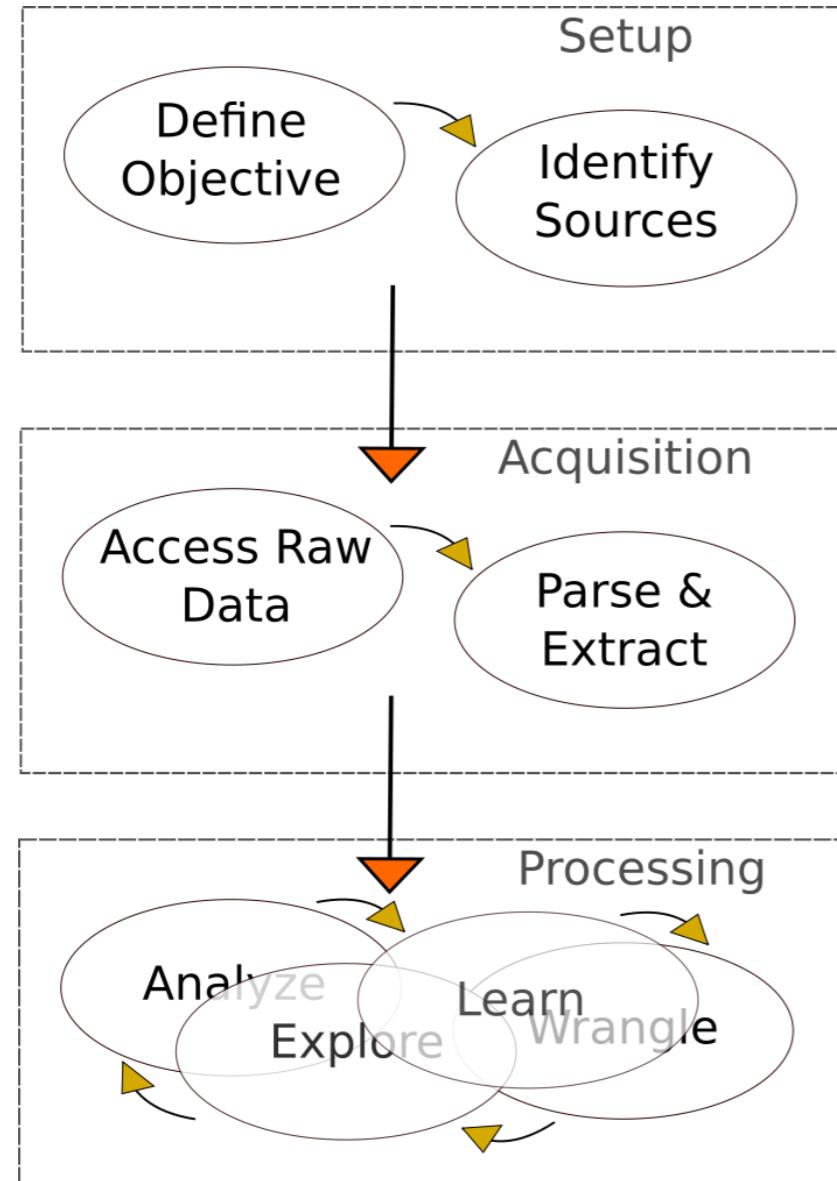
# Pipe Dream: Acquisition



## Acquisition

- Read in the raw data from online.
- Format these data to be usable.

# Pipe Dream: Processing



## Processing

- Many options!

# How do you do?

## Our Focus

- Acquisition!
- (Using `scrapy` via `python`)

# **Are you in?**

**WEB SCRAPING IN PYTHON**

# HyperText Markup Language

WEB SCRAPING IN PYTHON

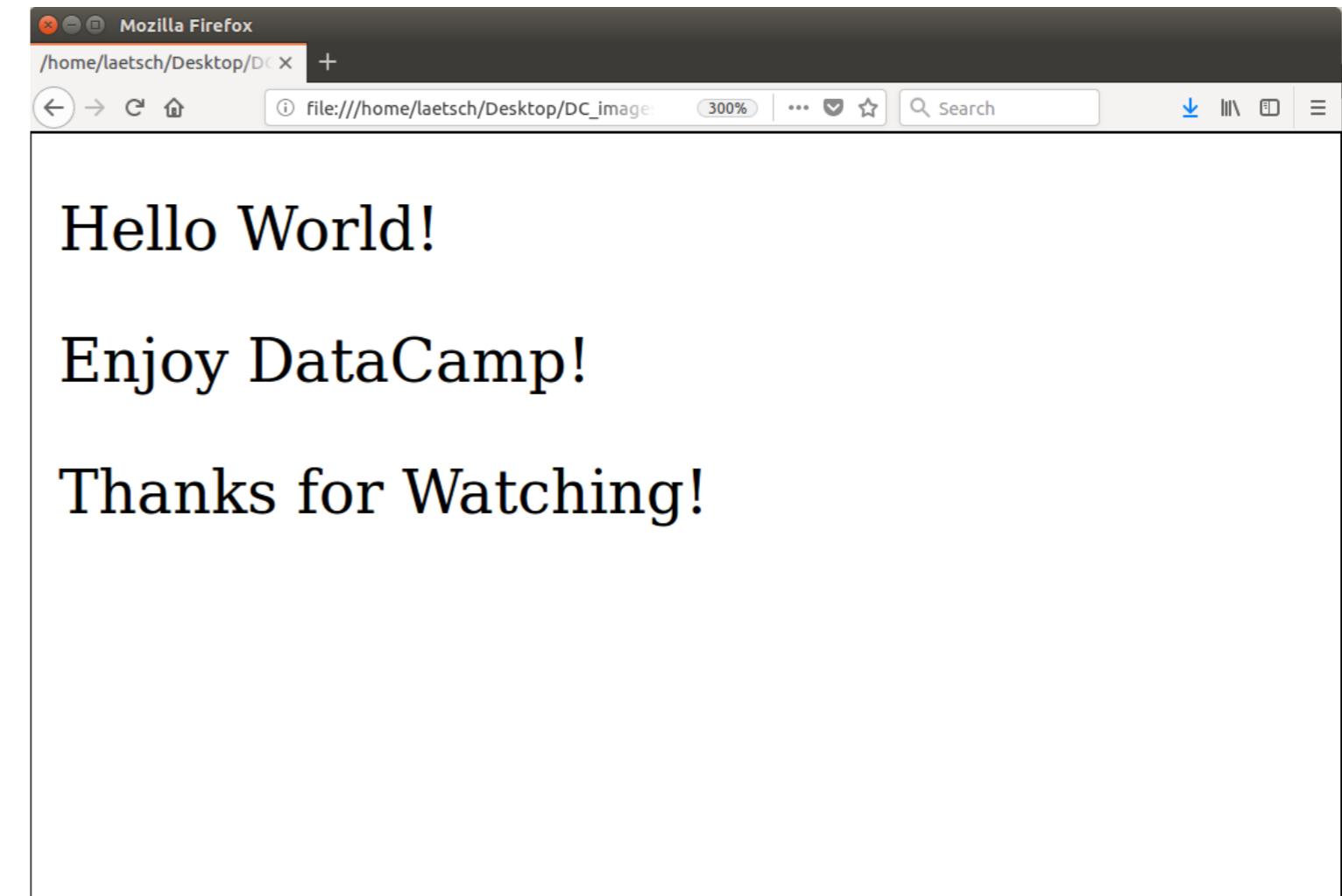


Thomas Laetsch

Data Scientist, NYU

# The main example

```
<html>
  <body>
    <div>
      <p>Hello World!</p>
      <p>Enjoy DataCamp!</p>
    </div>
    <p>Thanks for Watching!</p>
  </body>
</html>
```



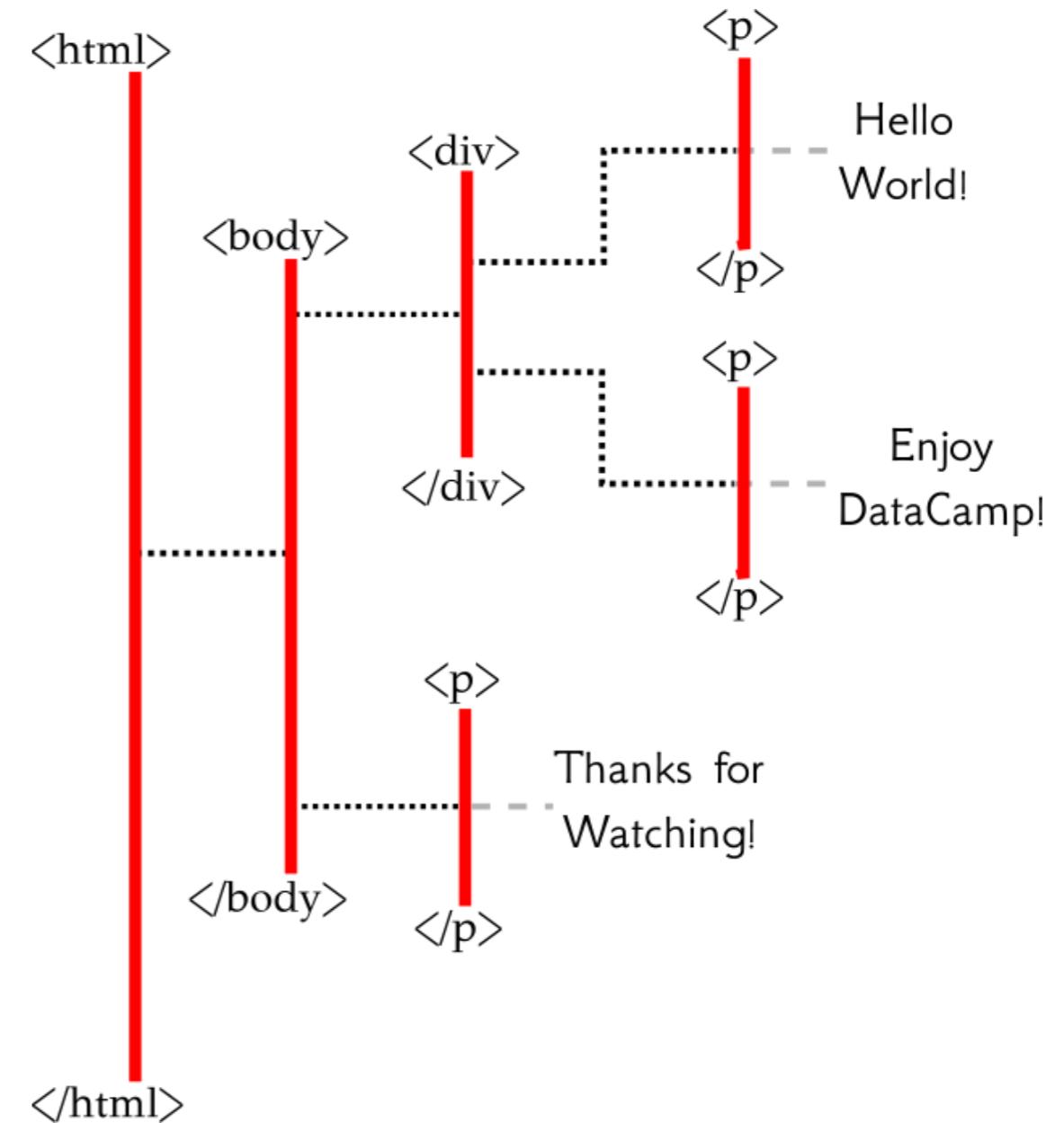
# HTML tags

```
<html>
  <body>
    <div>
      <p>Hello World!</p>
      <p>Enjoy DataCamp!</p>
    </div>
    <p>Thanks for Watching!</p>
  </body>
</html>
```

- <html> ... </html>
- <body> ... </body>
- <div> ... </div>
- <p> ... </p>

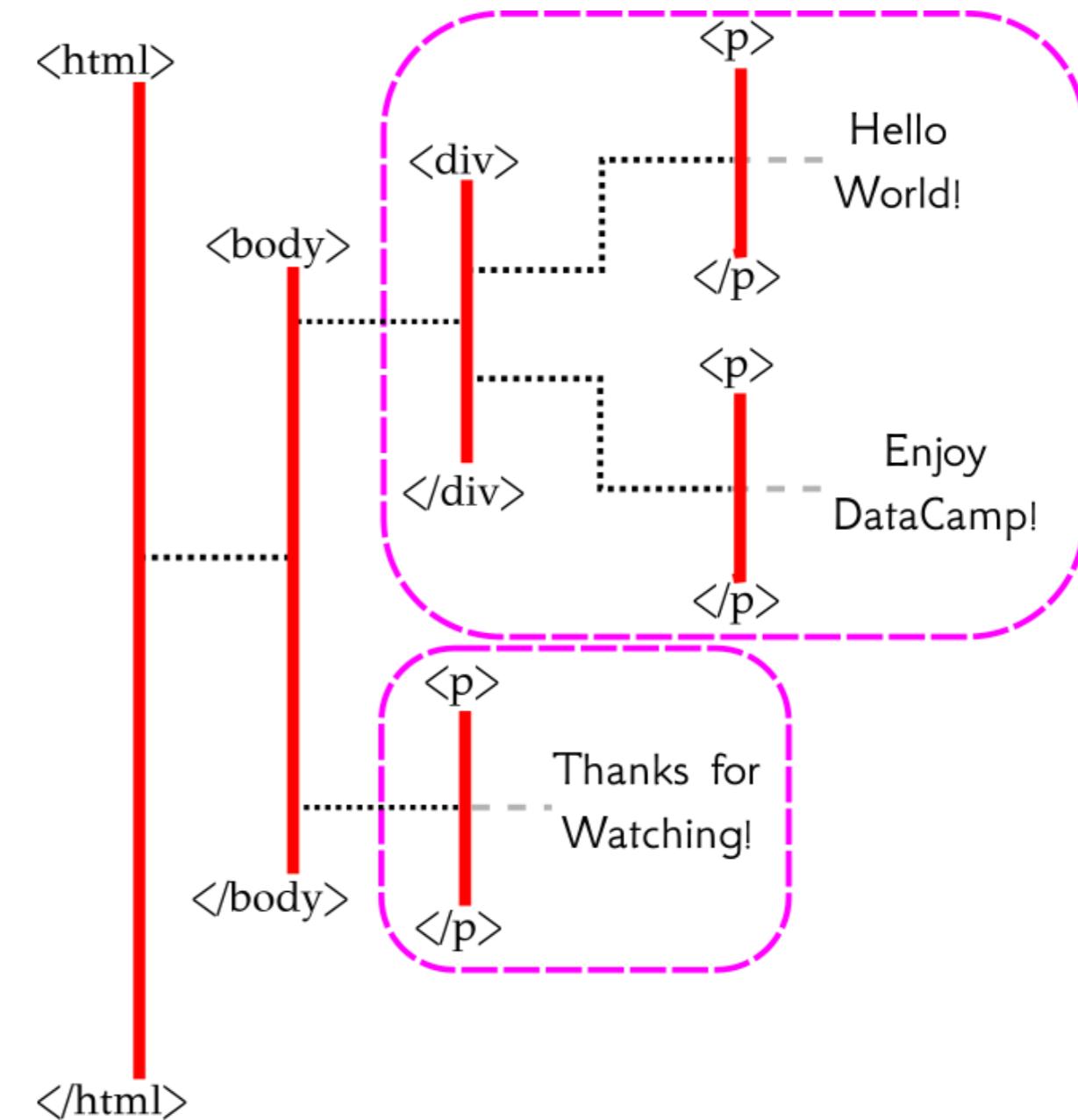
# The HTML tree

```
<html>
  <body>
    <div>
      <p>Hello World!</p>
      <p>Enjoy DataCamp!</p>
    </div>
    <p>Thanks for Watching!</p>
  </body>
</html>
```



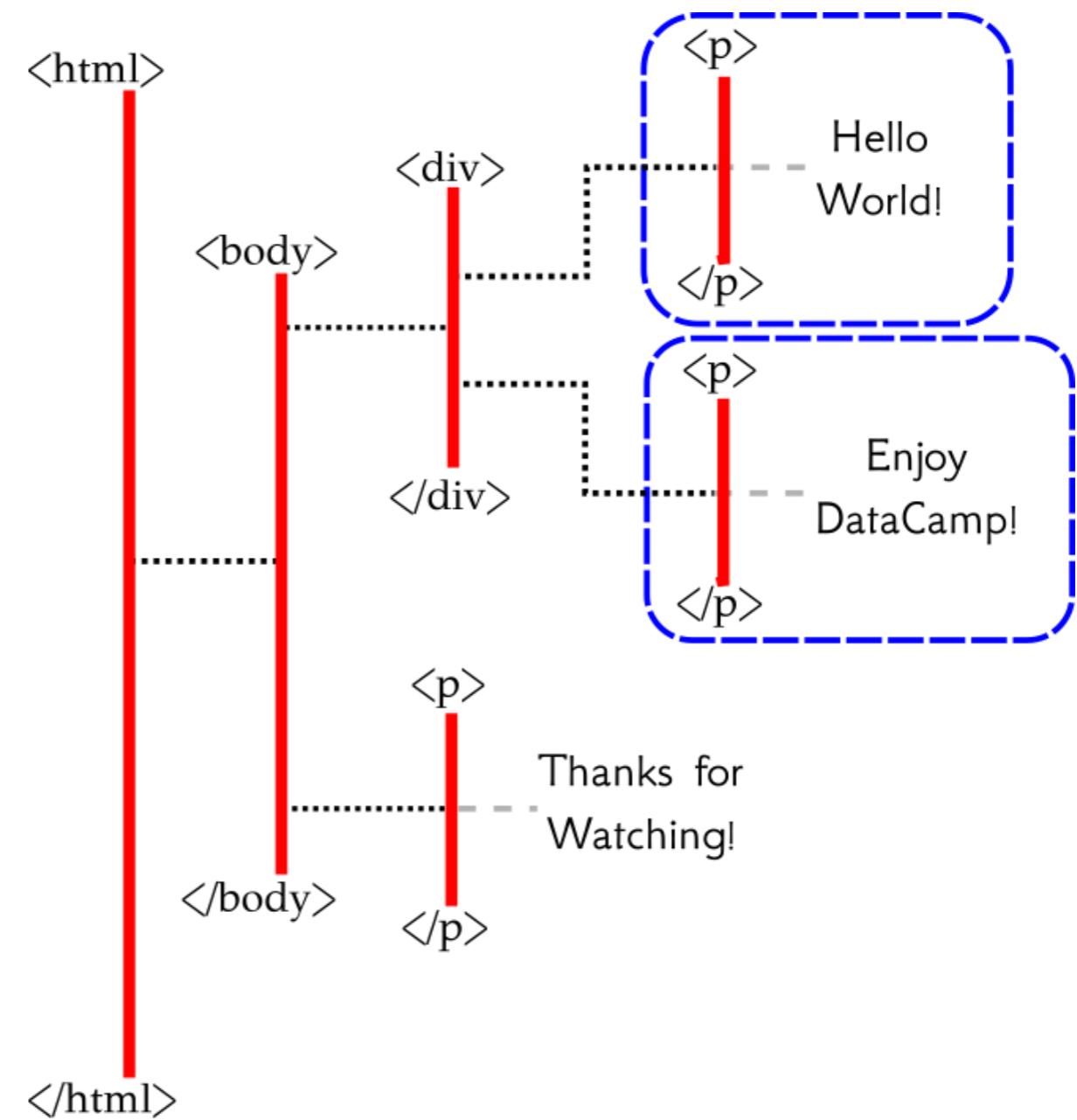
# The HTML tree: Example 1

```
<html>
  <body>
    <div>
      <p>Hello World!</p>
      <p>Enjoy DataCamp!</p>
    </div>
    <p>Thanks for Watching!</p>
  </body>
</html>
```



# The HTML tree: Example 2

```
<html>
  <body>
    <div>
      <p>Hello World!</p>
      <p>Enjoy DataCamp!</p>
    </div>
    <p>Thanks for Watching!</p>
  </body>
</html>
```

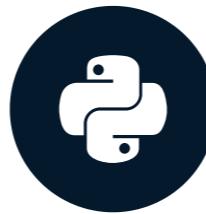


# **Introduction to HTML Outro**

**WEB SCRAPING IN PYTHON**

# HTML Tags and Attributes

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Do we have to?

- Information within HTML tags can be valuable
- Extract link URLs
- Easier way to select elements

# Tag, you're it!

```
<tag-name attrib-name="attrib info">
```

..element contents..

```
</tag-name>
```

- We've seen **tag names** such as **html**, **div**, and **p**.
- The **attribute name** is followed by = followed by information assigned to that attribute, usually quoted text.

# Let's "div"vy up the tag

```
<div id="unique-id" class="some class">
```

..div element contents..

```
</div>
```

- **id** attribute should be unique
- **class** attribute doesn't need to be unique

# "a" be linkin'

```
<a href="https://www.datacamp.com">
```

This text links to DataCamp!

```
</a>
```

- **a** tags are for **hyperlinks**
- **href** attribute tells what link to go to

# Tag Traction

The screenshot shows a web browser displaying the [w3schools.com/tags/default.asp](https://www.w3schools.com/tags/default.asp) page. The page title is "HTML Tags Ordered Alphabetically". On the left, there's a sidebar with "HTML Reference" and a list of HTML topics. The "HTML by Alphabet" link is highlighted. The main content area contains a search bar and a table listing HTML tags with their descriptions. A note indicates that the `<article>` tag is new in HTML5.

Tag	Description
<code>&lt;!--...--&gt;</code>	Defines a comment
<code>&lt;!DOCTYPE&gt;</code>	Defines the document type
<code>&lt;a&gt;</code>	Defines a hyperlink
<code>&lt;abbr&gt;</code>	Defines an abbreviation or an acronym
<code>&lt;acronym&gt;</code>	Not supported in HTML5. Use <code>&lt;abbr&gt;</code> instead. Defines an acronym
<code>&lt;address&gt;</code>	Defines contact information for the author/owner of a document
<code>&lt;applet&gt;</code>	Not supported in HTML5. Use <code>&lt;embed&gt;</code> or <code>&lt;object&gt;</code> instead. Defines an embedded applet
<code>&lt;area&gt;</code>	Defines an area inside an image-map
<code>&lt;article&gt;</code>	Defines an article

# **Et Tu, Attributes?**

**WEB SCRAPING IN PYTHON**

# Crash Course X

## WEB SCRAPING IN PYTHON



**Thomas Laetsch**  
Data Scientist, NYU

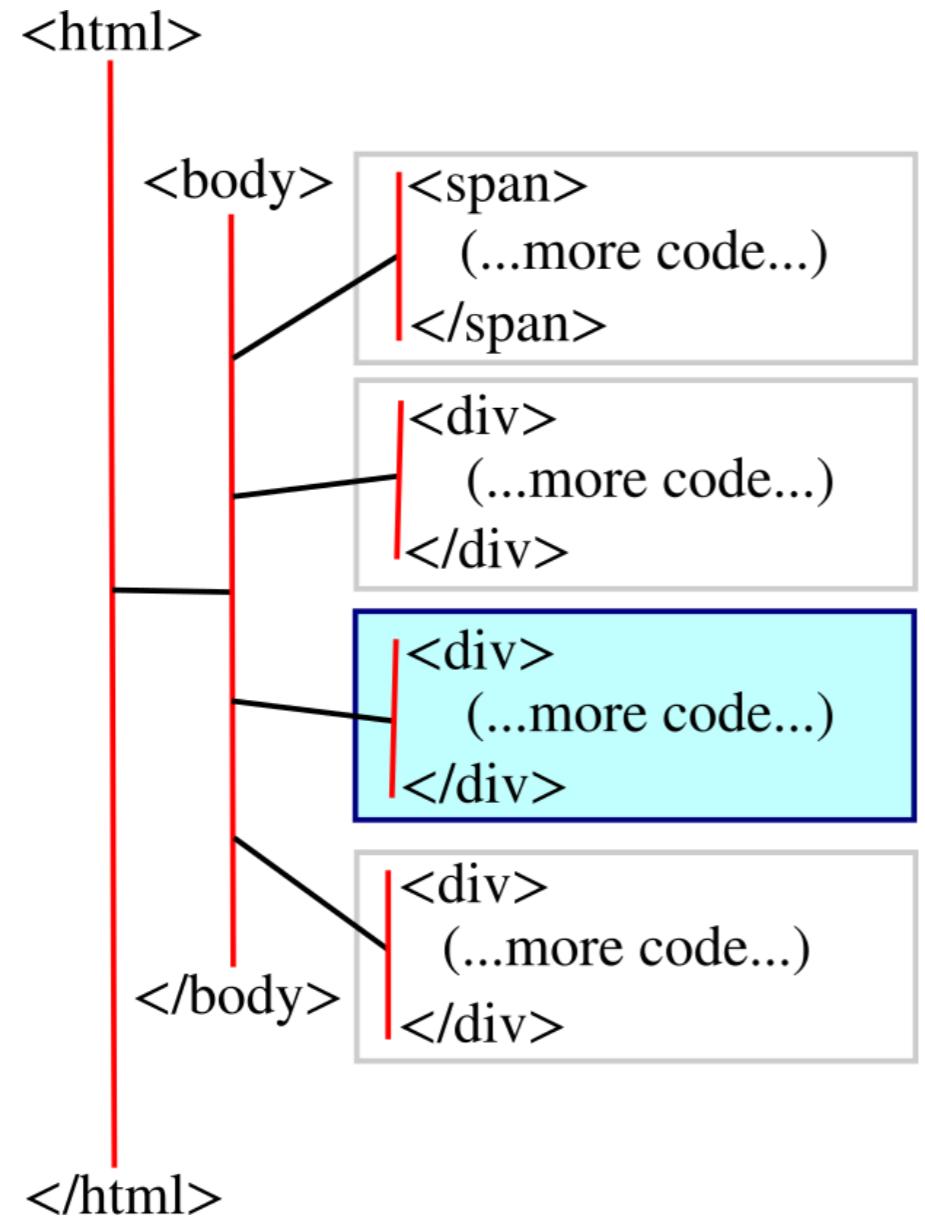
# Another Slasher Video?

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

## Simple XPath:

- Single forward-slash / used to move forward one generation.
- tag-names between slashes give direction to which element(s).
- Brackets [] after a tag name tell us which of the selected siblings to choose.

# Another Slasher Video?



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

# Slasher Double Feature?

- Direct to all `table` elements within the entire HTML code:

```
xpath = '//table'
```

- Direct to all `table` elements which are descendants of the 2nd `div` child of the `body` element:

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

# **Ex(path)celent**

**WEB SCRAPING IN PYTHON**

# XPath Navigation

## WEB SCRAPING IN PYTHON



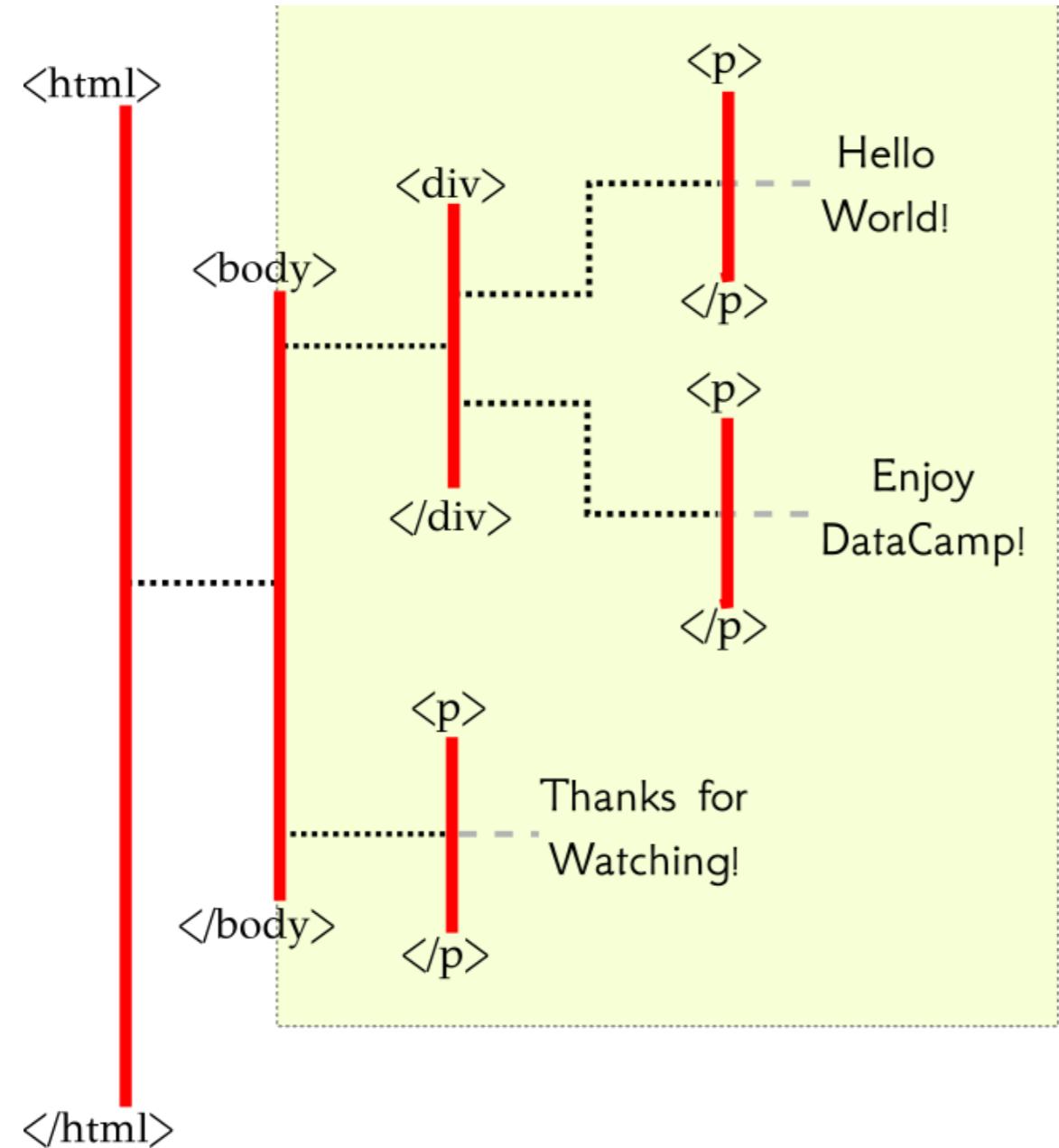
Thomas Laetsch

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



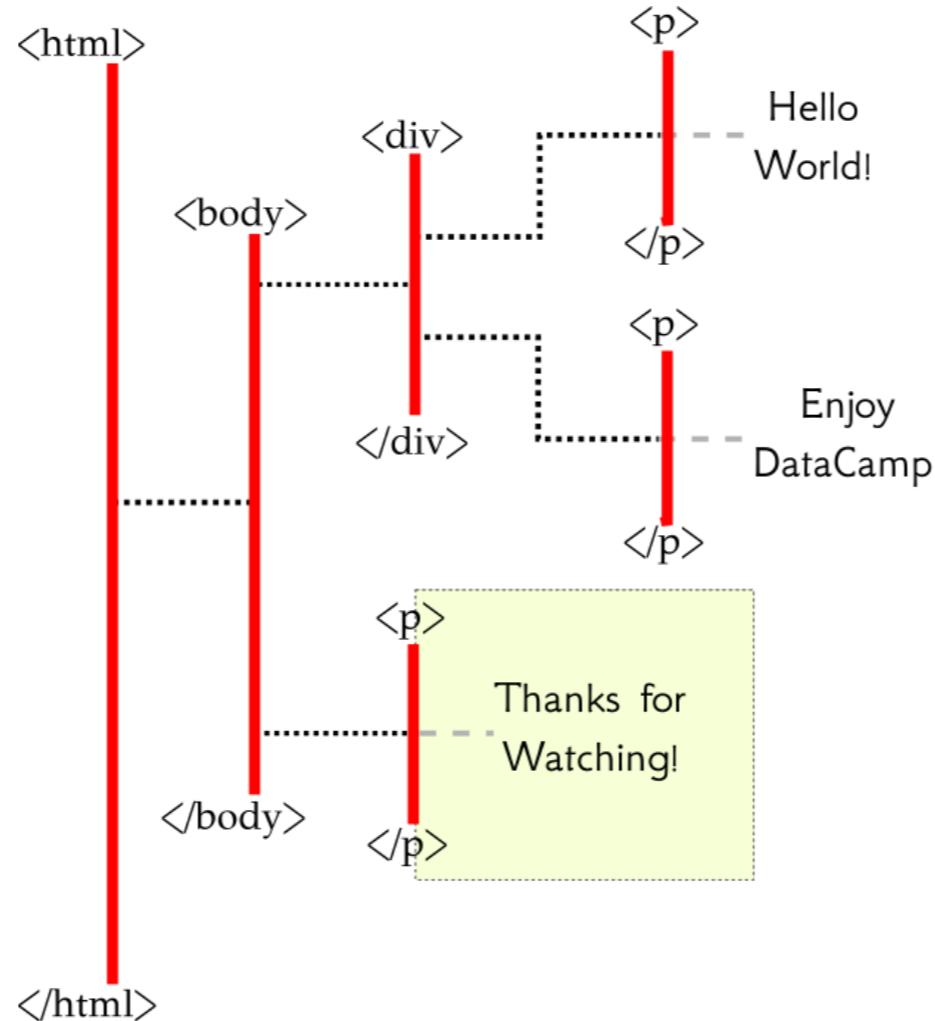
```
xpath = '/html/body'
```

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

- Give the same selection

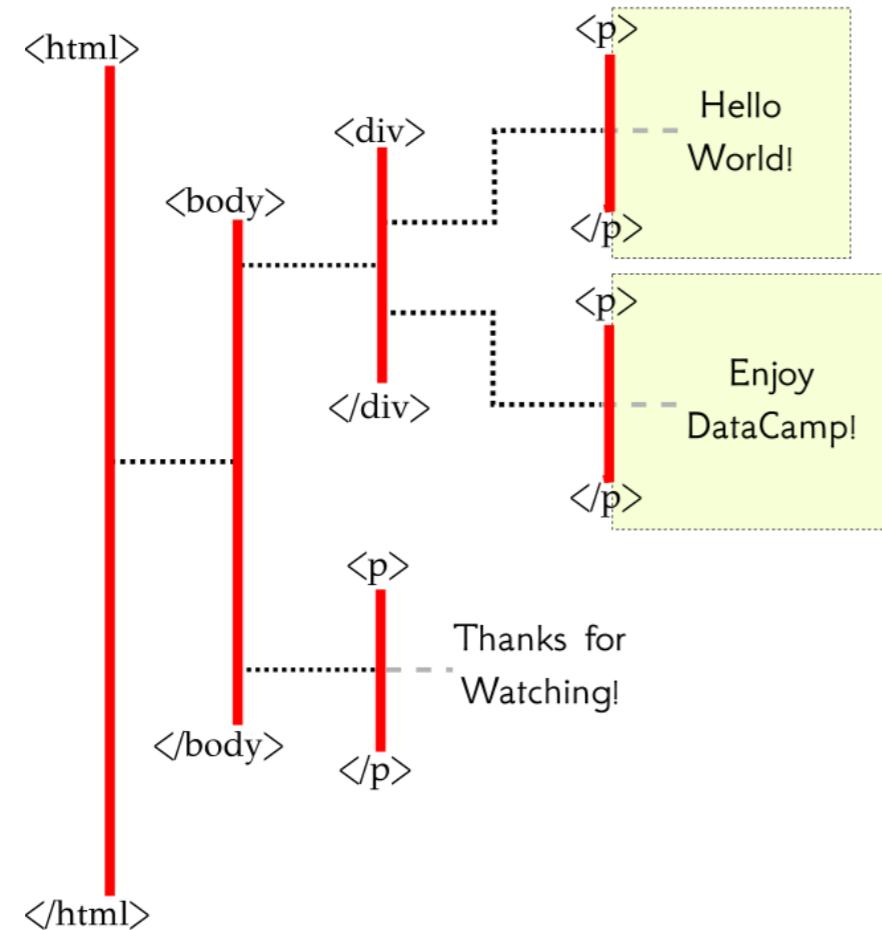
# A Body of P

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

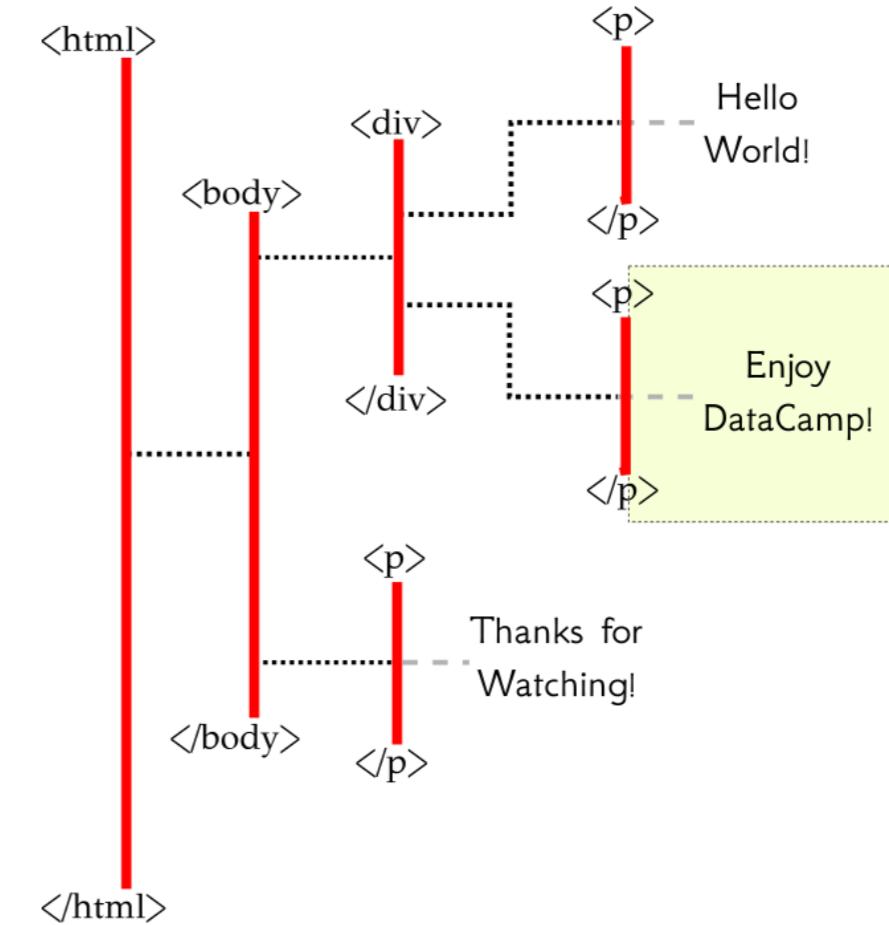


# The Birds and the Ps

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

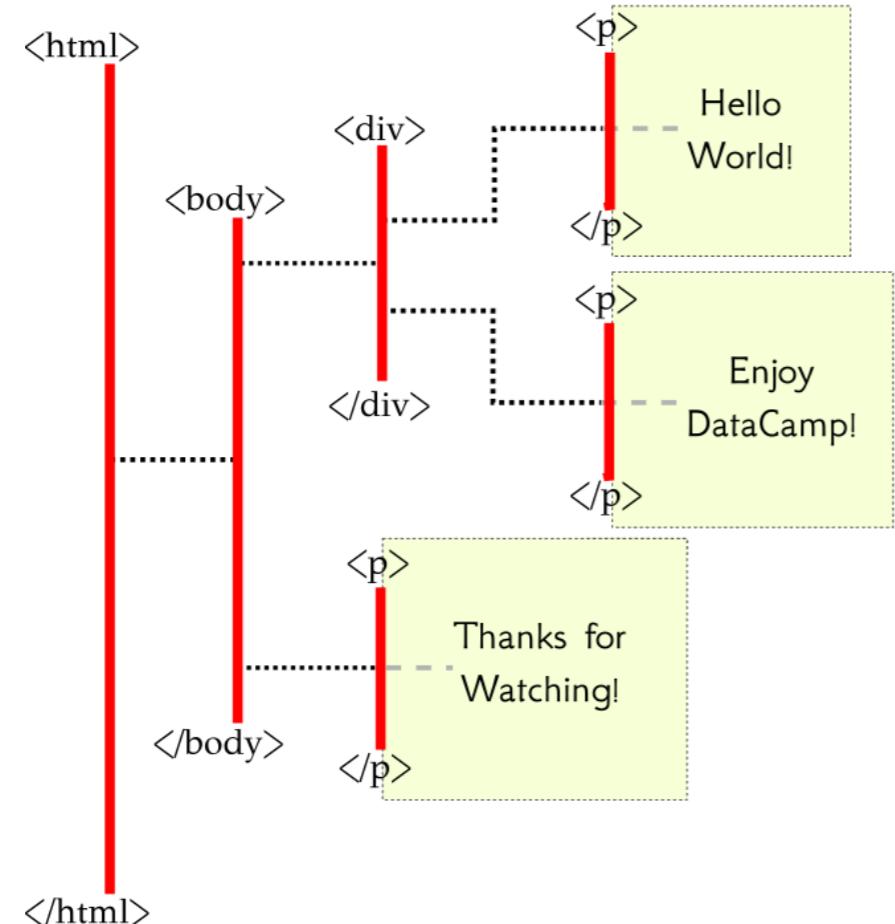


```
xpath = '/html/body/div/p[2]'
```

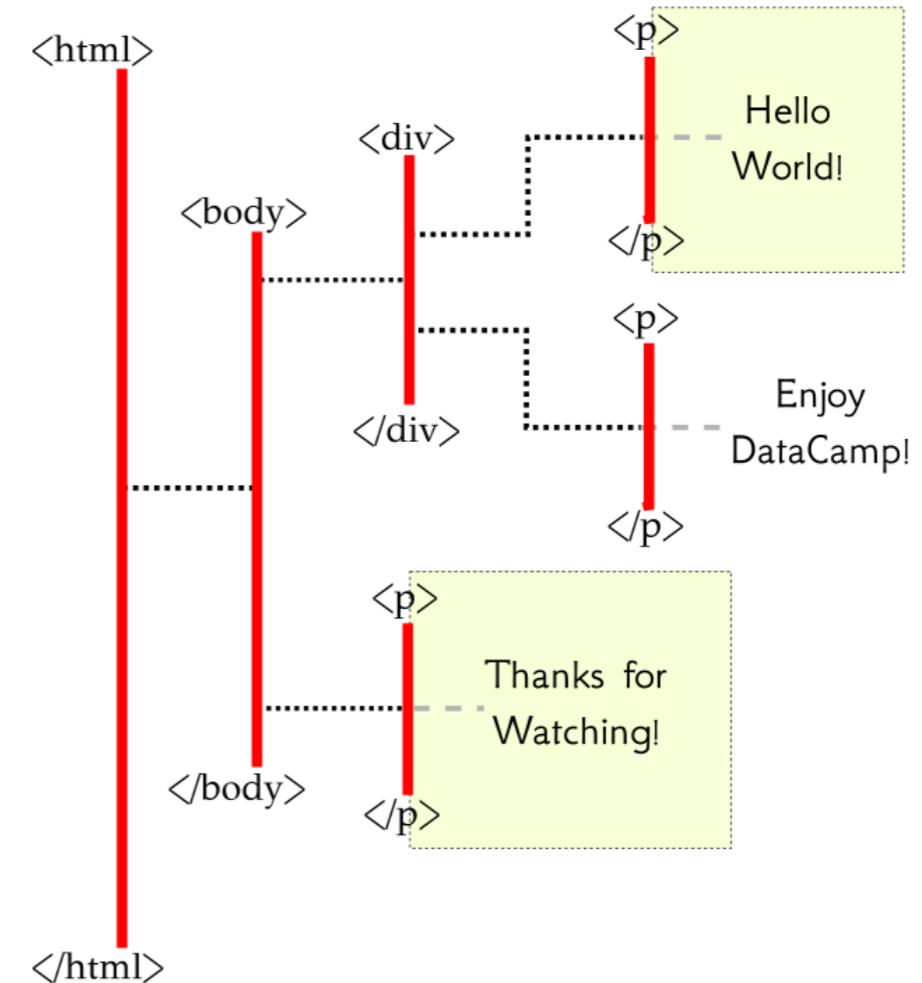


# Double Slashing the Brackets

`xpath = '//p'`



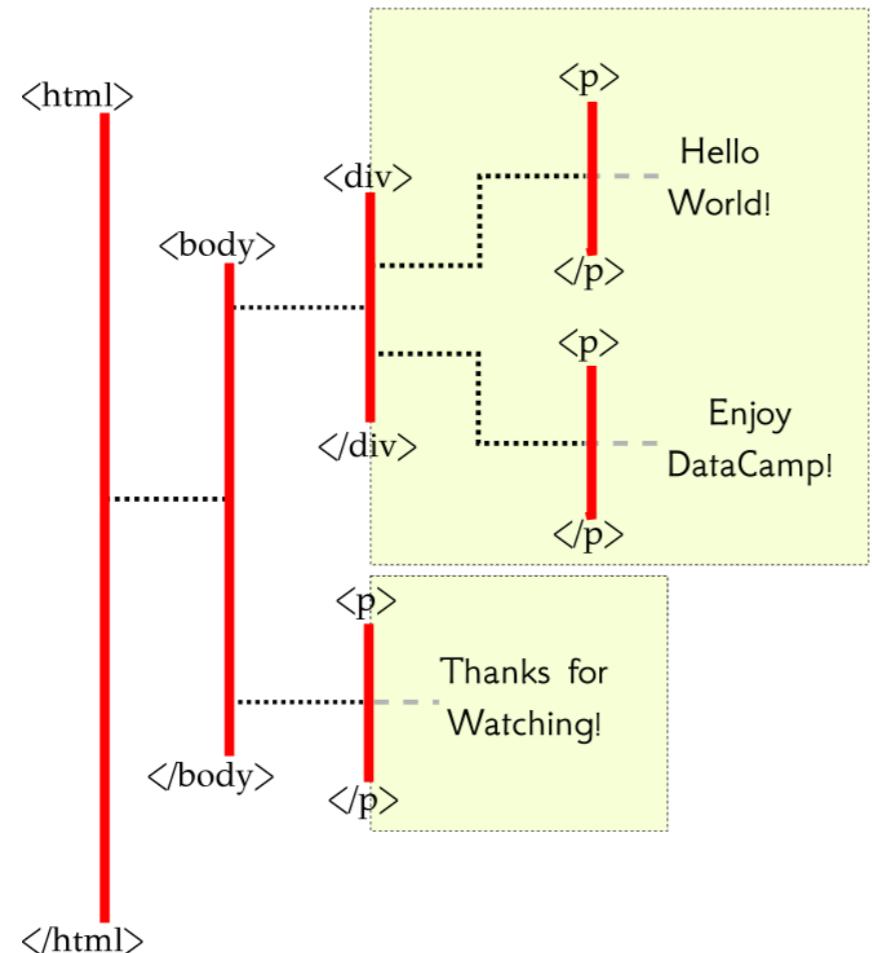
`xpath = '//p[1]'`



# The Wildcard

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

- The asterisks \* is the "wildcard"

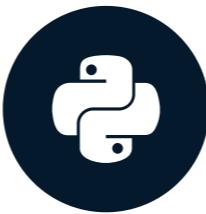


# Xposé

WEB SCRAPING IN PYTHON

# Off the Beaten XPath

WEB SCRAPING IN PYTHON



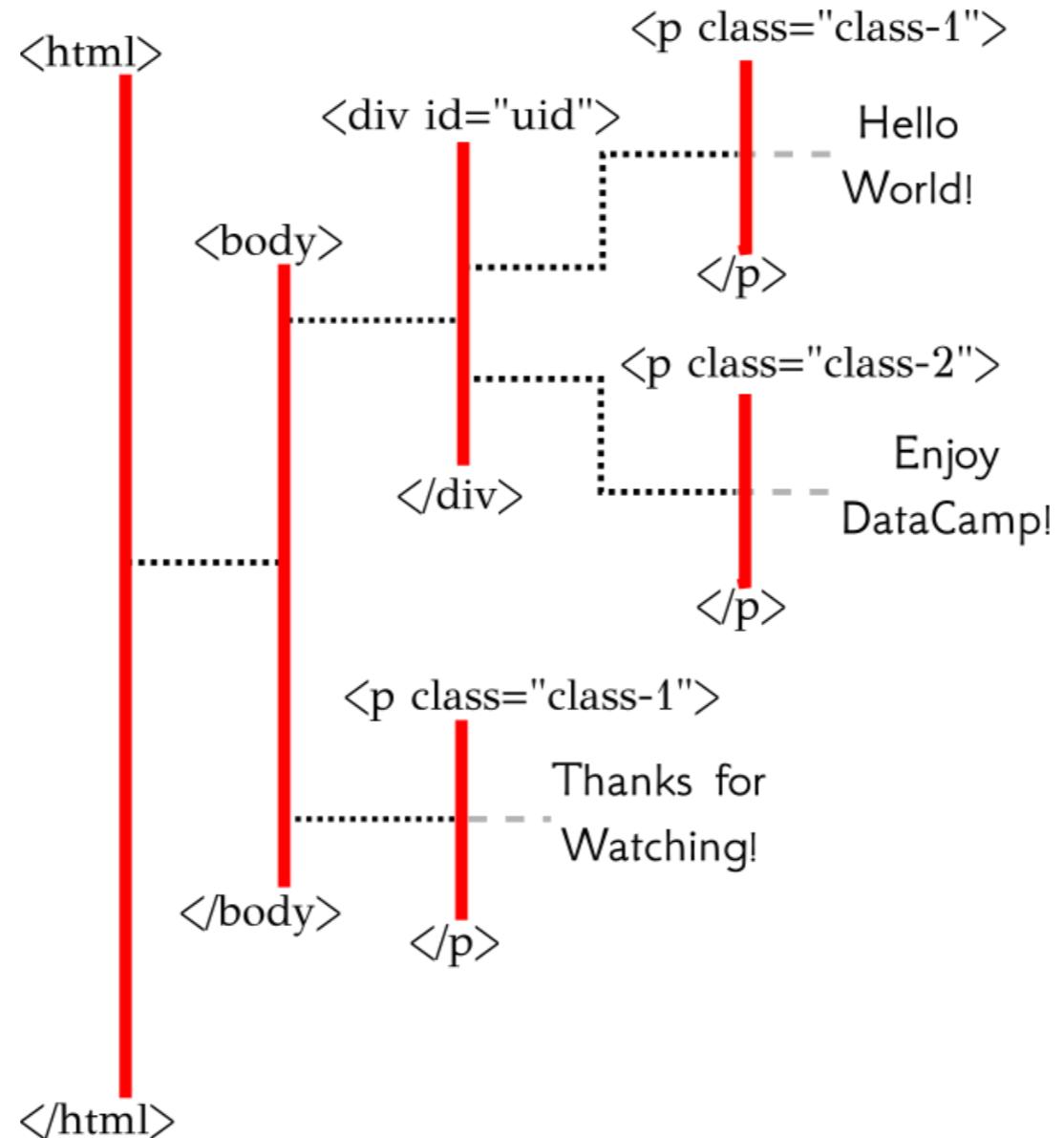
Thomas Laetsch

Data Scientist, NYU

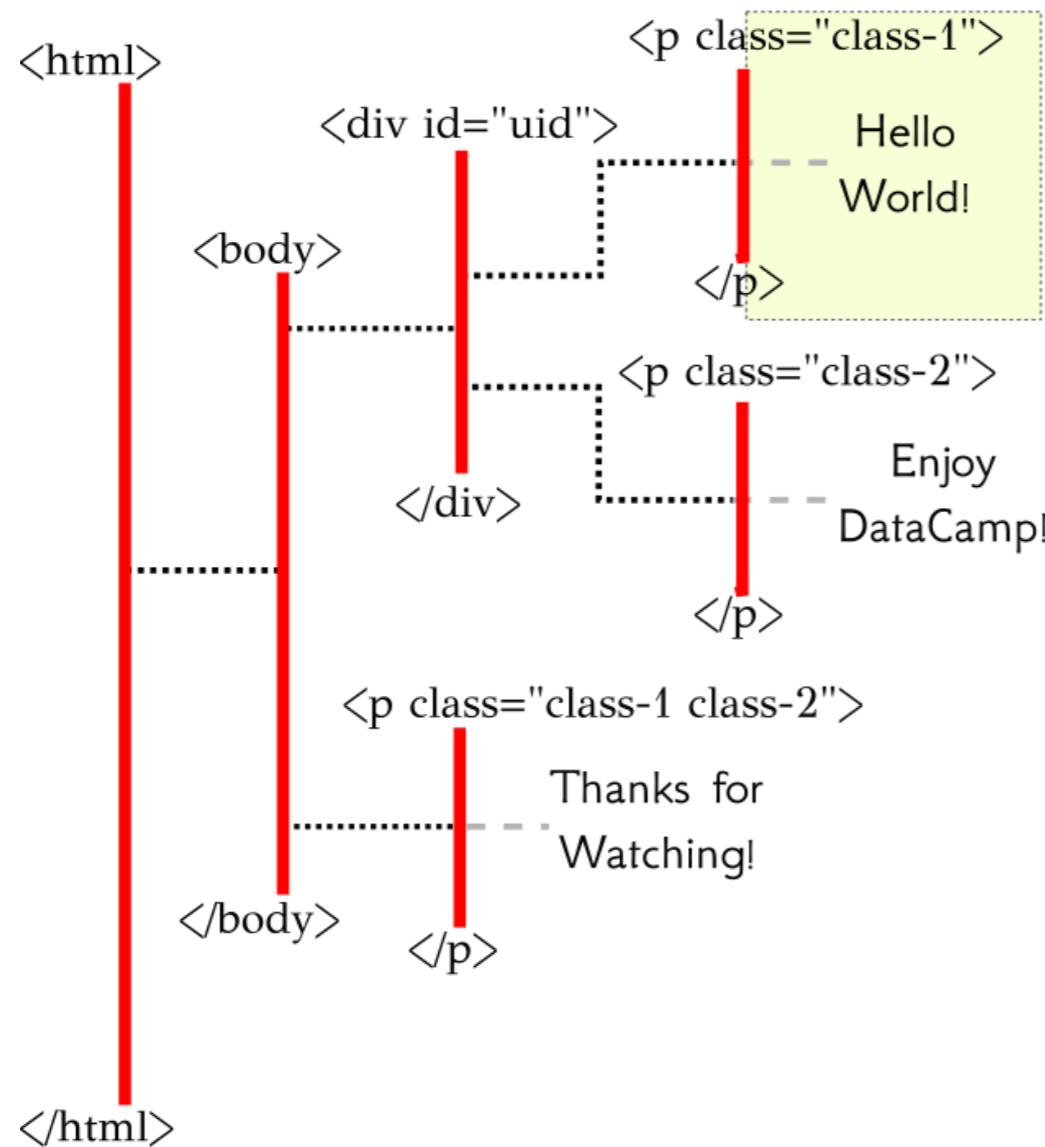
# (At)tribute

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

# Brackets and Attributes

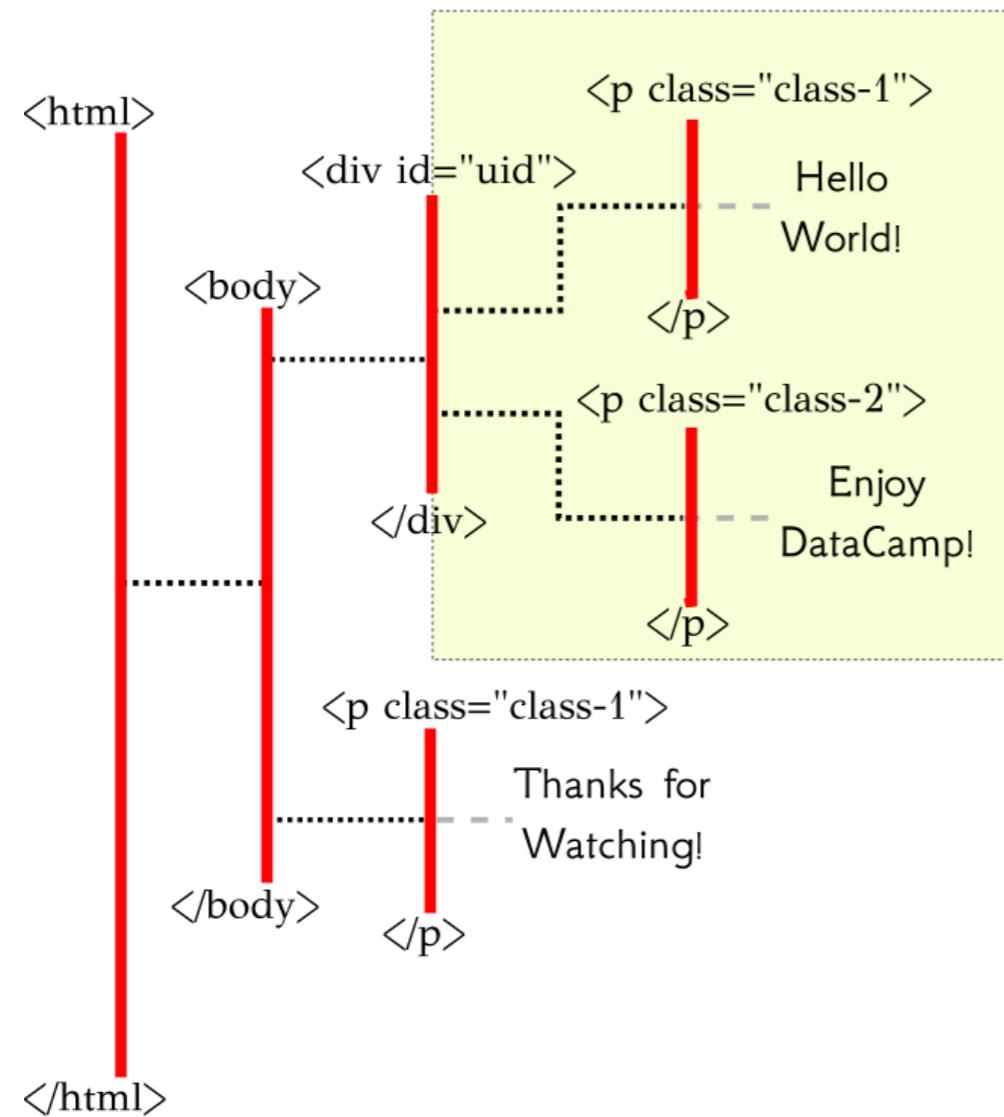


# Brackets and Attributes



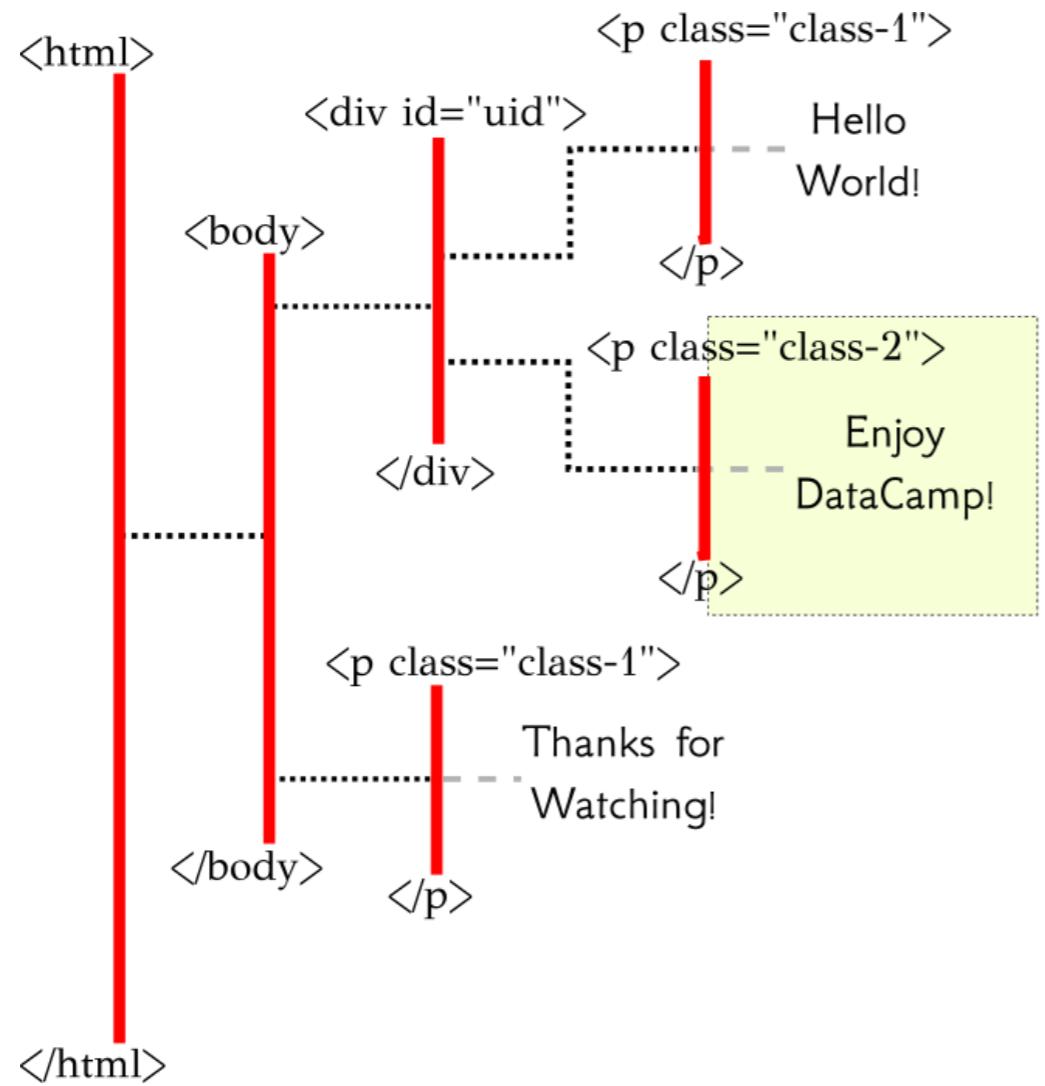
```
xpath = '//p[@class="class-1"]'
```

# Brackets and Attributes



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

# Brackets and Attributes



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

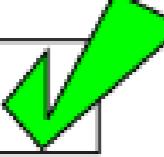
# Content with Contains

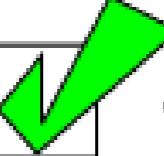
Xpath Contains Notation:

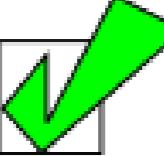
**contains( @attri-name, "string-expr" )**

# 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

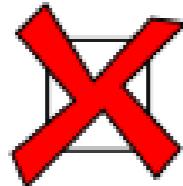
```
xpath = '//*[@class="class-1"]'
```



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

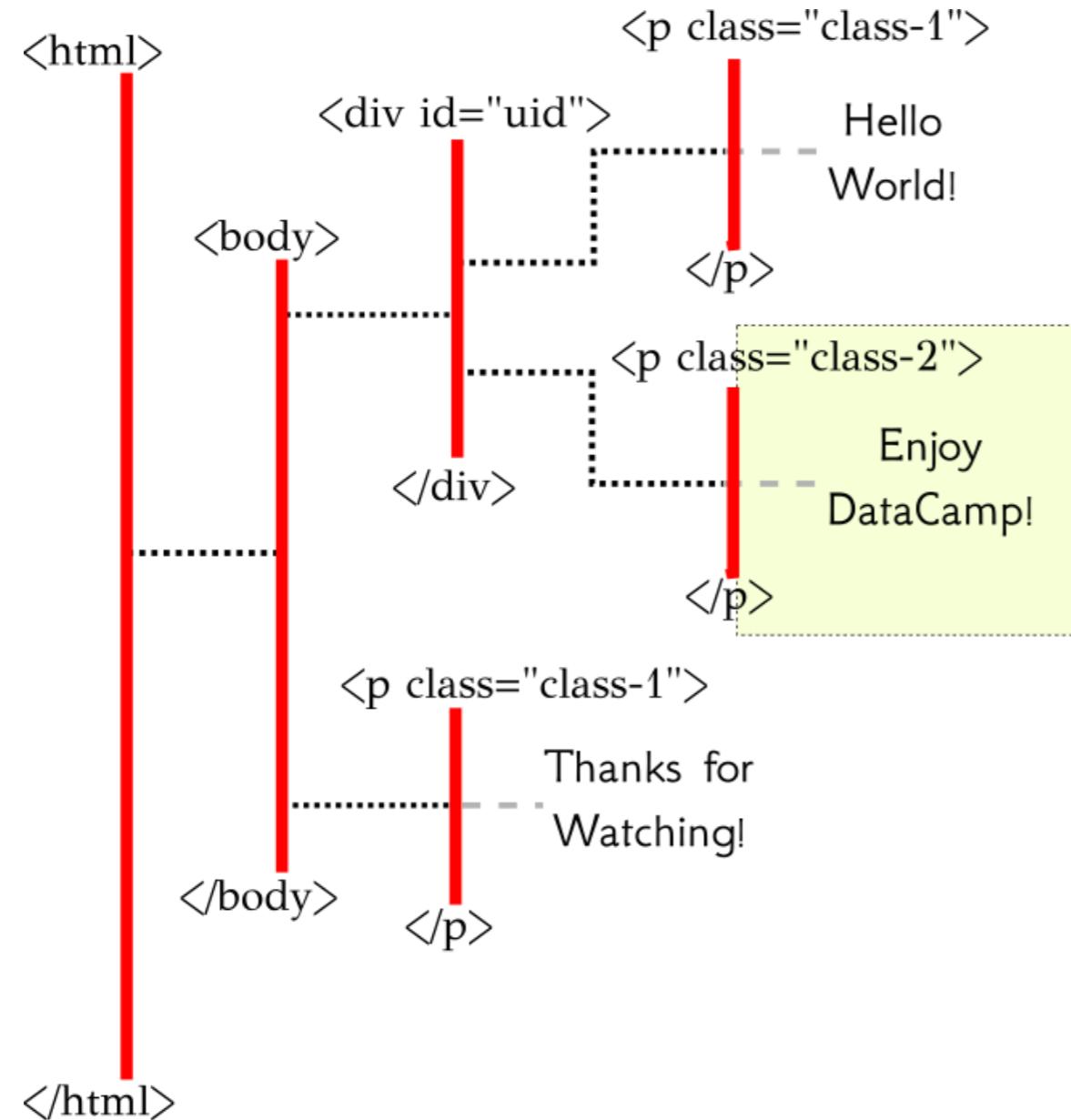


<div class="class-1 class-2"> ... </div>



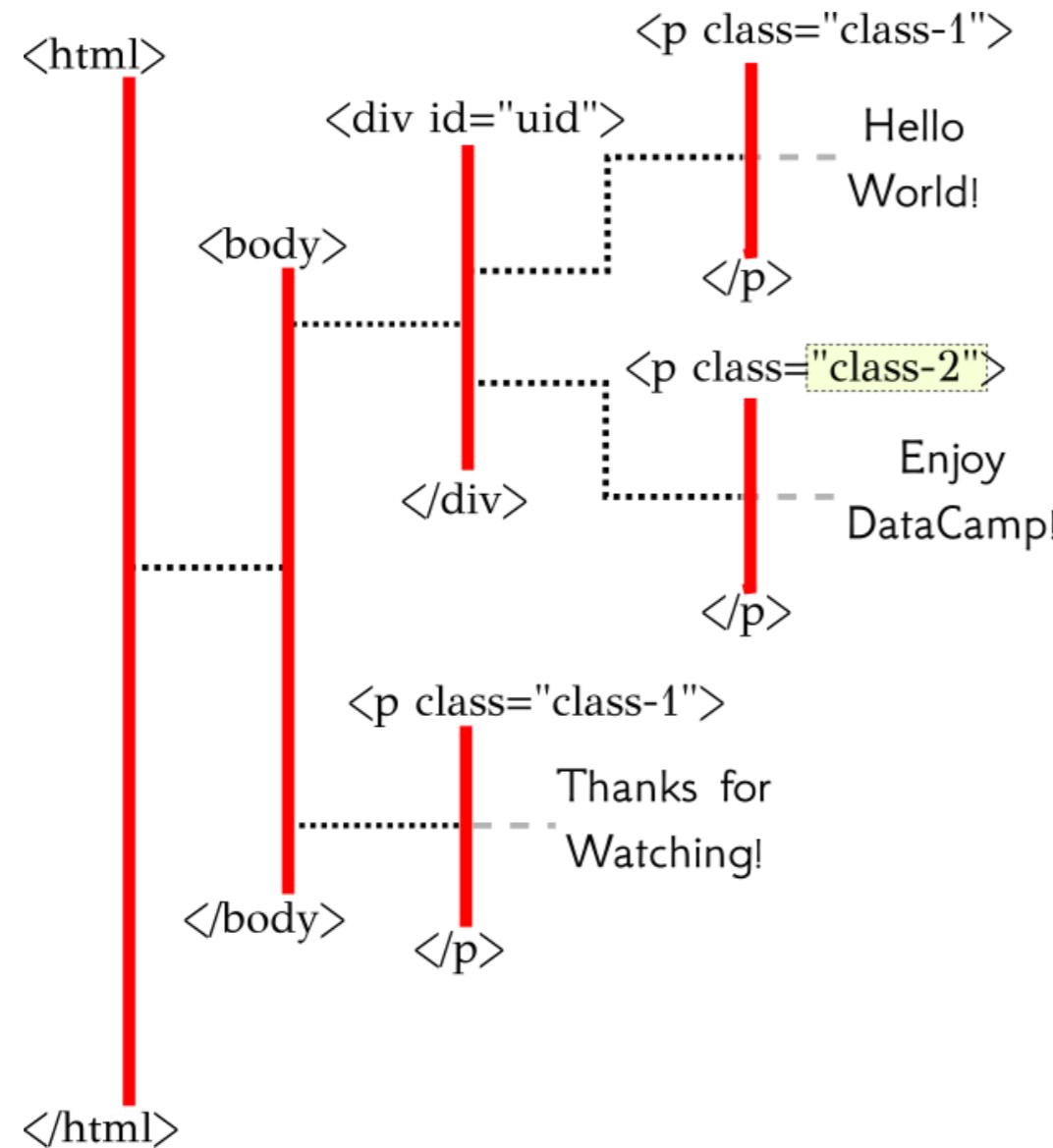
<p class="class-1 2"> ... </p>

# Get Classy



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

# Get Classy



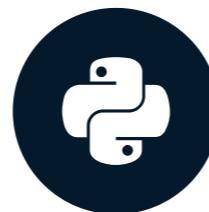
```
xpath = '/html/body/div/p[2]/@class'
```

# **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**

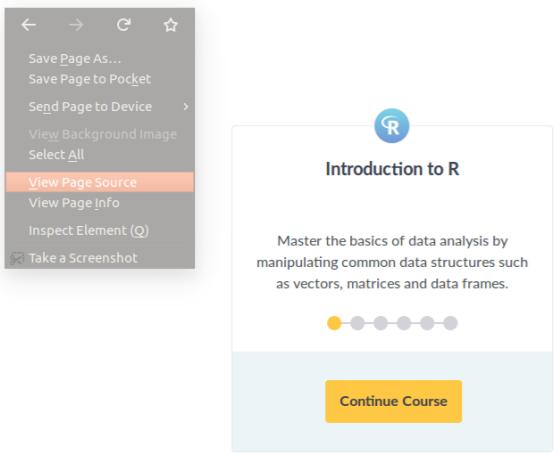


**Thomas Laetsch, PhD**

Data Scientist, NYU

# "Source" = HTML Code

The screenshot shows the DataCamp website's homepage. At the top, there's a banner with the word "Learn" and a subtitle: "Acquire new skills fast in courses that combine short expert videos with immediate hands-on-keyboard exercises." Below this are five course cards: "DATA VISUALIZATION WITH PYTHON", "WORKING WITH DPLYR", "INTRODUCTION TO SQL FOR DATA SCIENCE", "INTRODUCTION TO ML FOR DATA SCIENCE", and "WORKING WITH POSTGRESQL". A search bar at the bottom asks "What do you want to learn?".



The screenshot shows the browser's developer tools with the "View Source" tab selected. The page source code is displayed, showing various HTML, CSS, and JavaScript snippets. The code includes meta tags for description and canonical URLs, script tags for Google Tag Manager, and a large block of CSS and JavaScript at the bottom.

```
<!DOCTYPE html>
<html class="no-js">
  <head>
    <script>
      (function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
        new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName('script')[0],
        j=d.createElement(s),dl=l['dataLayer']?l['dataLayer']:[];j.async=true;j.src=
        'https://www.googletagmanager.com/gtm.js?id='+i;l.appendChild(j);f.parentNode.insertBefore(j,f);
      })(window,document,'script','dataLayer','GTM-TGWB2P');</script>
    ...
    <meta charset="utf-8">
    <script>window.NREUM=[{}];NREUM.info={"beacon":"bam.nr-data.net","errorBeacon":"bam.nr-data.net","licenseKey":"4795905ee2","applicationID":"90826195","transactionName":"JlkNEEQLVA0DE0wFDBBEAFFF1kNCg==","queueTime":0,"appl...
    <script>window.NREUM=[{}],_nr_require=function(e,n,t){function r(t){if(!n[t]){}(o.exports,exports){(var o=e[t][1][n];return r(o)||n)o,o.exports)}return n[t].exports}if("function"==t...
    <title>Data Science Courses: R & Python Analysis Tutorials | DataCamp</title>
    <meta name="description" content="DataCamp offers a variety of online courses & video tutorials to help you learn data science at your own pace. See why over 3,220,000 people use DataCamp now!">
    <link rel="canonical" href="https://www.datacamp.com/courses/all">
    <meta name="twitter:title" content="Data Science Courses: R & Python Analysis Tutorials">
    <meta name="twitter:description" content="DataCamp offers a variety of online courses & video tutorials to help you learn data science at your own pace. See why over 3,220,000 people use DataCamp now!">
    <meta name="twitter:card" content="summary">
    <meta name="twitter:site" content="@DataCamp">
    <meta name="twitter:image" content="https://www.datacamp.com/datacamp-sq.png">
    <meta name="twitter:image:width" content="300">
    <meta name="twitter:image:height" content="300">
    <meta name="twitter:creator" content="@DataCamp">
    <meta name="twitter:domain" content="www.datacamp.com">
    <meta property="og:image" content="https://www.datacamp.com/datacamp.png">
    <meta property="og:image:width" content="1200">
    <meta property="og:image:height" content="630">
    <meta property="og:title" content="Data Science Courses: R & Python Analysis Tutorials">
    <meta name="author" content="https://plus.google.com/u/0/+DataCamp/>
    <link rel="shortcut icon" type="image/x-icon" href="https://cdn.datacamp.com/main-app/assets/favicon-335cd0394b32102a39221d79e5fd7e51078e6d32a0c8aea59676a6869f84e9d8.ico" />
    <meta name="csrf-param" content="authenticity_token" />
    <meta name="csrf-token" content="++Z785sRb47v0ePlWzqoPOnH653qKVUcp9xLxEL3uSqAT8cgUZ1meY9+dsxDabTlHdMZKcv/Pp7S1Mn/RJ3a=>
    <link rel="manifest" href="/manifest.json" />
    ...
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
    <meta name="fragment" content="!"/>
    <meta name="google-site-verification" content="ao3s4PdiisD20sfTbld07YjxVX2QLKEtloPyFTjo8" />
    <meta name="apple-itunes-app" content="app-id=1263413087" />
    <link rel="stylesheet" media="all" href="https://cdn.datacamp.com/main-app/assets/application_v2-566f39ae10a65051b6d46fe6a477656a9871872da2df4cb953391830af5d97dd.css" />
    <script>
      (function(h,o,t,j,a,r){
        h.jid=h.jid||function(){(h.jq=h.jq||[]).push(arguments)};
        h._jsSettings={jid:484663,hjsv:6};
        a=o.getElementsByTagName('head')[0];
        r=o.createElement('script');r.async=1;
        r.src=t+h._jsSettings.jid+j+h._jsSettings.hjsv;
        a.appendChild(r);
      })(window,document,'https://static.hotjar.com/c/hotjar-','.js?sv=');
    </script>
    ...
    </head>
    <body class="js-application-v2" data-env="production"><noscript><iframe src="https://www.googletagmanager.com/ns.html?id=GTM-TGWB2P" height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
    ...
    <div class="site-wrap">
      <div class="dc-flash-wrapper" id="flash_messages">
    </div>
  
```

# Inspecting Elements

The screenshot shows the DataCamp website's main landing page. At the top, there's a banner titled "Learn" with a sub-instruction: "Acquire new skills fast in courses that combine short expert videos with immediate hands-on-keyboard exercises." Below this, there's a row of five course icons: "DATA VISUALIZATION WITH PYTHON", "WORKING WITH DPLYR", "INTRODUCTION TO T-SQL FOR DATA SCIENCE", "INTRODUCTION TO SPSS FOR DATA SCIENCE", and "INTRODUCING DATA IN POSTGRESQL". A search bar at the top right contains the placeholder "What do you want to learn?". Below the search bar, there are dropdown menus for "All Technologies" and "All Topics". The main content area is titled "All Data Science Courses" and features three course cards:

- Introduction to R**: Master the basics of data analysis by manipulating common data structures such as vectors, matrices and data frames. Includes a progress bar and a "Continue Course" button.
- Intro to Statistics with R: Correlation and Linear R...**: If you have ever taken a math or statistics class, you've probably heard the old adage "Correlation does not imply causation". Includes a progress bar and a "Continue Course" button.
- Intro to Statistics with R: Multiple Regression**: Multiple regression is a powerful statistical technique, and here you will discover what it is, how it works, and how to use it. Part of the DataCamp Data Science Specialization. Includes a progress bar and a "Continue Course" button. This card also has a context menu open over the third item, with options like "Open Link in New Tab", "Bookmark This Link", "Save Link As...", "Save Link to Pocket", "Copy Link Location", "Search Google for 'Intro to Statistics with R: Multiple Regression'", and "Send Link to Device". A "Inspect Element" button is highlighted in orange.

This screenshot shows the same DataCamp page with the browser's developer tools (Inspector) open. The "Elements" tab is selected, and the "course-block\_title" element of the third course card is highlighted with a blue border. The browser's address bar shows the URL <https://www.datacamp.com/courses/all>. The developer tools interface includes a "Search HTML" field, a "Filter Styles" dropdown, and a large panel on the right displaying the CSS styles for the selected element. The styles listed include:

```
element { inline; } .a477656a9871872da2df4cb953391830af5d97dd.css:1 .course-block_title { margin-bottom: 0; margin-top: 18px; min-height: 56px; } h4, .h4, .dc-h4 { font-family: "lato", sans-serif; font-size: 20px; font-weight: 700; line-height: 25px; margin-bottom: 0px; color: #3d4251; font-weight: 700; }
```

# HTML text to Selector

```
from scrapy import Selector
```

```
import requests
url = 'https://www.datacamp.com/courses/all'
html = requests.get( url ).content
```

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

# You Know Our Secrets

WEB SCRAPING IN PYTHON

# CSS Locators

## WEB SCRAPING IN PYTHON



**Thomas Laetsch**

Data Scientist, NYU

# Rosetta CSStone

- `/` replace by `>` (except first character)
  - **XPath:** `/html/body/div`
  - **CSS Locator:** `html > body > div`
- `//` replaced by a blank space (except first character)
  - **XPath:** `//div/span//p`
  - **CSS Locator:** `div > span p`
- `[N]` replaced by `:nth-of-type(N)`
  - **XPath:** `//div/p[2]`
  - **CSS Locator:** `div > p:nth-of-type(2)`

# Rosetta CSStone

## XPATH

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

## CSS

```
css = 'html > body div > p:nth-of-type(2)'
```

# Attributes in CSS

- To find an element by class, use a period .
  - Example: `p.class-1` selects all paragraph elements belonging to `class-1`
- To find an element by id, use a pound sign #
  - Example: `div#uid` selects the `div` element with `id` equal to `uid`

# Attributes in CSS

Select paragraph elements within class `class1` :

```
css_locator = 'div#uid > p.class1'
```

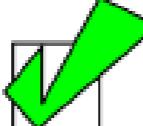
Select all elements whose class attribute belongs to `class1` :

```
css_locator = '.class1'
```

# Class Status

```
css = '.class1'
```

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

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

 <p class="class-12" > ... </p>

# Class Status

```
xpath = '//*[@class="class1"]'
```



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



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

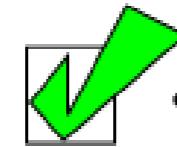


<p class="class-12"> ... </p>

# Class Status

```
xpath = '//*[contains(@class, "class1")]'
```

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

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

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

# Selectors with CSS

```
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 )
```

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

>>> sel.css("div > p").extract()
out: [ '<p>Hello World!</p>' ]
```

# c(ss) You Soon!

WEB SCRAPING IN PYTHON

# Attribute and Text Selection

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# You Must have Guts to use your Colon

- Using XPath: <xpath-to-element>/@attr-name

```
xpath = '//div[@id="uid"]/a/@href'
```

- Using CSS Locator: <css-to-element>::attr(attr-name)

```
css_locator = 'div#uid > a::attr(href)'
```

# Text Extraction

```
<p id="p-example">  
    Hello world!  
    Try <a href="http://www.datacamp.com">DataCamp</a> today!  
</p>
```

- In XPath use `text()`

```
sel.xpath('//p[@id="p-example"]/text()').extract()  
# result: ['\n Hello world!\n Try ', ' today!\n']
```

```
sel.xpath('//p[@id="p-example"]//text()').extract()  
# result: ['\n Hello world!\n Try ', 'DataCamp', ' today!\n']
```

# Text Extraction

```
<p id="p-example">  
    Hello world!  
    Try <a href="http://www.datacamp.com">DataCamp</a> today!  
</p>
```

- For CSS Locator, use `::text`

```
sel.css('p#p-example::text').extract()  
# result: ['\n Hello world!\n Try ', ' today!\n']
```

```
sel.css('p#p-example ::text').extract()  
# result: ['\n Hello world!\n Try ', 'DataCamp', ' today!\n']
```

# **Scoping the Colon**

**WEB SCRAPING IN PYTHON**

# Getting Ready to Crawl

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Let's Respond

## Selector vs Response:

- The Response **has all the tools we learned with Selectors**:
  - `xpath` and `css` methods followed by `extract` and `extract_first` methods.
- The Response also **keeps track of the url** where the HTML code was loaded from.
- The Response **helps us move from one site to another**, so that we can "crawl" the web while scraping.

# What We Know!

- `xpath` method works like a Selector

```
response.xpath('//div/span[@class="bio"]')
```

- `css` method works like a Selector

```
response.css('div > span.bio')
```

- Chaining works like a Selector

```
response.xpath('//div').css('span.bio')
```

- Data extraction works like a Selector

```
response.xpath('//div').css('span.bio').extract()  
response.xpath('//div').css('span.bio').extract_first()
```

# What We Don't Know

- The `response` keeps track of the URL within the `response.url` variable.

```
response.url
```

```
>>> 'http://www.DataCamp.com/courses/all'
```

- The `response` lets us "follow" a new link with the `follow()` method

```
# next_url is the string path of the next url we want to scrape
```

```
response.follow( next_url )
```

- We'll learn more about `follow` later.

# In Response

## WEB SCRAPING IN PYTHON

# Scraping For Reals

## WEB SCRAPING IN PYTHON



**Thomas Laetsch**  
Data Scientist, NYU

# DataCamp Site

<https://www.datacamp.com/courses/all>

# What's the Div, Yo?

```
# response loaded with HTML from https://www.datacamp.com/courses/all
```

```
course_divs = response.css('div.course-block')
```

```
print( len(course_divs) )  
>>> 185
```

# Inspecting course-block

```
first_div = course_divs[0]
children = first_div.xpath('./*')
print( len(children) )
>>> 3
```

# The first child

```
first_div = course_divs[0]  
children = first_div.xpath('./*')
```

```
first_child = children[0]  
print( first_child.extract() )  
>>> <a class=... />
```

# The second child

```
first_div = course_divs[0]  
children = first_div.xpath('./*')
```

```
second_child = children[1]  
print( second_child.extract() )  
>>> <div class=... />
```

# The forgotten child

```
first_div = course_divs[0]  
children = first_div.xpath('./*')
```

```
third_child = children[2]  
print( third_child.extract() )  
>>> <span class=... />
```

# Listful

- In one CSS Locator

```
links = response.css('div.course-block > a::attr(href)').extract()
```

- Stepwise

```
# step 1: course blocks
course_divs = response.css('div.course-block')
# step 2: hyperlink elements
hrefs = course_divs.xpath('./a/@href')
# step 3: extract the links
links = hrefs.extract()
```

# Get Schooled

```
for l in links:  
    print( l )  
  
>>> /courses/free-introduction-to-r  
>>> /courses/data-table-data-manipulation-r-tutorial  
>>> /courses/dplyr-data-manipulation-r-tutorial  
>>> /courses/ggvis-data-visualization-r-tutorial  
>>> /courses/reporting-with-r-markdown  
>>> /courses/intermediate-r  
  
...
```

# **Links Achieved**

**WEB SCRAPING IN PYTHON**

# A Classy Spider

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Your Spider

```
import scrapy
from scrapy.crawler import CrawlerProcess

class SpiderClassName(scrapy.Spider):
    name = "spider_name"
    # the code for your spider
    ...

process = CrawlerProcess()

process.crawl(SpiderClassName)

process.start()
```

# Your Spider

- Required imports

```
import scrapy
from scrapy.crawler import CrawlerProcess
```

- The part we will focus on: the actual spider

```
class SpiderClassName(scrapy.Spider):
    name = "spider_name"
    # the code for your spider
    ...
```

- Running the spider

```
# initiate a CrawlerProcess
process = CrawlerProcess()

# tell the process which spider to use
process.crawl(YourSpider)

# start the crawling process
process.start()
```

# Weaving the Web

```
class DCspider( scrapy.Spider ):  
  
    name = 'dc_spider'  
  
    def start_requests( self ):  
        urls = [ 'https://www.datacamp.com/courses/all' ]  
        for url in urls:  
            yield scrapy.Request( url = url, callback = self.parse )  
  
    def parse( self, response ):  
        # simple example: write out the html  
        html_file = 'DC_courses.html'  
        with open( html_file, 'wb' ) as fout:  
            fout.write( response.body )
```

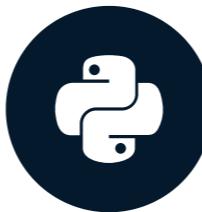
- Need to have a function called `start_requests`
- Need to have at least one parser function to handle the HTML code

# We'll Weave the Web Together

WEB SCRAPING IN PYTHON

# A Request for Service

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Spider Recall

```
import scrapy
from scrapy.crawler import CrawlerProcess

class SpiderClassName(scrapy.Spider):
    name = "spider_name"
    # the code for your spider
    ...

process = CrawlerProcess()

process.crawl(SpiderClassName)

process.start()
```

# Spider Recall

```
class DCspider( scrapy.Spider ):  
    name = "dc_spider"  
  
    def start_requests( self ):  
        urls = [ 'https://www.datacamp.com/courses/all' ]  
        for url in urls:  
            yield scrapy.Request( url = url, callback = self.parse )  
  
    def parse( self, response ):  
        # simple example: write out the html  
        html_file = 'DC_courses.html'  
        with open( html_file, 'wb' ) as fout:  
            fout.write( response.body )
```

# The Skinny on start\_requests

```
def start_requests( self ):  
    urls = ['https://www.datacamp.com/courses/all']  
    for url in urls:  
        yield scrapy.Request( url = url, callback = self.parse )
```

```
def start_requests( self ):  
    url = 'https://www.datacamp.com/courses/all'  
    yield scrapy.Request( url = url, callback = self.parse )
```

- `scrapy.Request` here will fill in a response variable for us
- The `url` argument tells us which site to scrape
- The `callback` argument tells us where to send the response variable for processing

# Zoom Out

```
class DCspider( scrapy.Spider ):  
    name = "dc_spider"  
  
    def start_requests( self ):  
        urls = [ 'https://www.datacamp.com/courses/all' ]  
        for url in urls:  
            yield scrapy.Request( url = url, callback = self.parse )  
  
    def parse( self, response ):  
        # simple example: write out the html  
        html_file = 'DC_courses.html'  
        with open( html_file, 'wb' ) as fout:  
            fout.write( response.body )
```

# **End Request**

**WEB SCRAPING IN PYTHON**

# Move Your Bloomin' Parse

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Once Again

```
class DCspider( scrapy.Spider ):  
    name = "dcspider"  
  
    def start_requests( self ):  
        urls = [ 'https://www.datacamp.com/courses/all' ]  
        for url in urls:  
            yield scrapy.Request( url = url, callback = self.parse )  
  
    def parse( self, response ):  
        # simple example: write out the html  
        html_file = 'DC_courses.html'  
        with open( html_file, 'wb' ) as fout:  
            fout.write( response.body )
```

# You Already Know!

```
def parse( self, response ):  
    # input parsing code with response that you already know!  
    # output to a file, or...  
    # crawl the web!
```

# DataCamp Course Links: Save to File

```
class DCspider( scrapy.Spider ):  
    name = "dcspider"  
  
    def start_requests( self ):  
        urls = [ 'https://www.datacamp.com/courses/all' ]  
        for url in urls:  
            yield scrapy.Request( url = url, callback = self.parse )  
    def parse( self, response ):  
        links = response.css('div.course-block > a::attr(href)').extract()  
        filepath = 'DC_links.csv'  
        with open( filepath, 'w' ) as f:  
            f.writelines( [link + '/n' for link in links] )
```

# DataCamp Course Links: Parse Again

```
class DCspider( scrapy.Spider ):
    name = "dcspider"

    def start_requests( self ):
        urls = [ 'https://www.datacamp.com/courses/all' ]
        for url in urls:
            yield scrapy.Request( url = url, callback = self.parse )
    def parse( self, response ):
        links = response.css('div.course-block > a::attr(href)').extract()
        for link in links:
            yield response.follow( url = link, callback = self.parse2 )
    def parse2( self, response ):
        # parse the course sites here!
```



# **Johnny Parsin'**

**WEB SCRAPING IN PYTHON**

# Capstone

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Inspecting Elements

```
import scrapy
from scrapy.crawler import CrawlerProcess

class DC_Chapter_Spider(scrapy.Spider):

    name = "dc_chapter_spider"

    def start_requests( self ):
        url = 'https://www.datacamp.com/courses/all'
        yield scrapy.Request( url = url,
                             callback = self.parse_front )

    def parse_front( self, response ):
        ## Code to parse the front courses page

    def parse_pages( self, response ):
        ## Code to parse course pages
        ## Fill in dc_dict here

    dc_dict = dict()

    process = CrawlerProcess()
    process.crawl(DC_Chapter_Spider)
    process.start()
```

# Parsing the Front Page

```
def parse_front( self, response ):  
    # Narrow in on the course blocks  
    course_blocks = response.css( 'div.course-block' )  
    # Direct to the course links  
    course_links = course_blocks.xpath( './a/@href' )  
    # Extract the links (as a list of strings)  
    links_to_follow = course_links.extract()  
    # Follow the links to the next parser  
    for url in links_to_follow:  
        yield response.follow( url = url,  
                               callback = self.parse_pages )
```

# Parsing the Course Pages

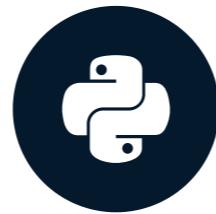
```
def parse_pages( self, response ):  
    # Direct to the course title text  
    crs_title = response.xpath('//h1[contains(@class,"title")]/text()')  
    # Extract and clean the course title text  
    crs_title_ext = crs_title.extract_first().strip()  
    # Direct to the chapter titles text  
    ch_titles = response.css( 'h4.chapter__title::text' )  
    # Extract and clean the chapter titles text  
    ch_titles_ext = [t.strip() for t in ch_titles.extract()]  
    # Store this in our dictionary  
    dc_dict[ crs_title_ext ] = ch_titles_ext
```

# **It's time to Weave**

## **WEB SCRAPING IN PYTHON**

# Stop Scratching and Start Scraping!

WEB SCRAPING IN PYTHON



Thomas Laetsch

Data Scientist, NYU

# Feeding the Machine

# Scraping Skills

- **Objective:** Scrape a website computationally
- **How?** We decide to use `scrapy`
- **How?** We need to work with:
  - `Selector` and `Response` objects
  - Maybe even create a Spider
- **How?** We need to learn XPath or CSS Locator notation
- **How?** Understand the structure of HTML

# What'd'ya Know?

- Structure of HTML
- XPath and CSS Locator notation
- How to use Selector and Response objects in scrapy
- How to set up a spider
- How to scrape the web

# EOT

## WEB SCRAPING IN PYTHON