WEB SCRAPING WITH PYTHON

# Web Scraping With Python

Thomas Laetsch
Data Scientist, NYU

# Business Savvy

**What are businesses looking for?**

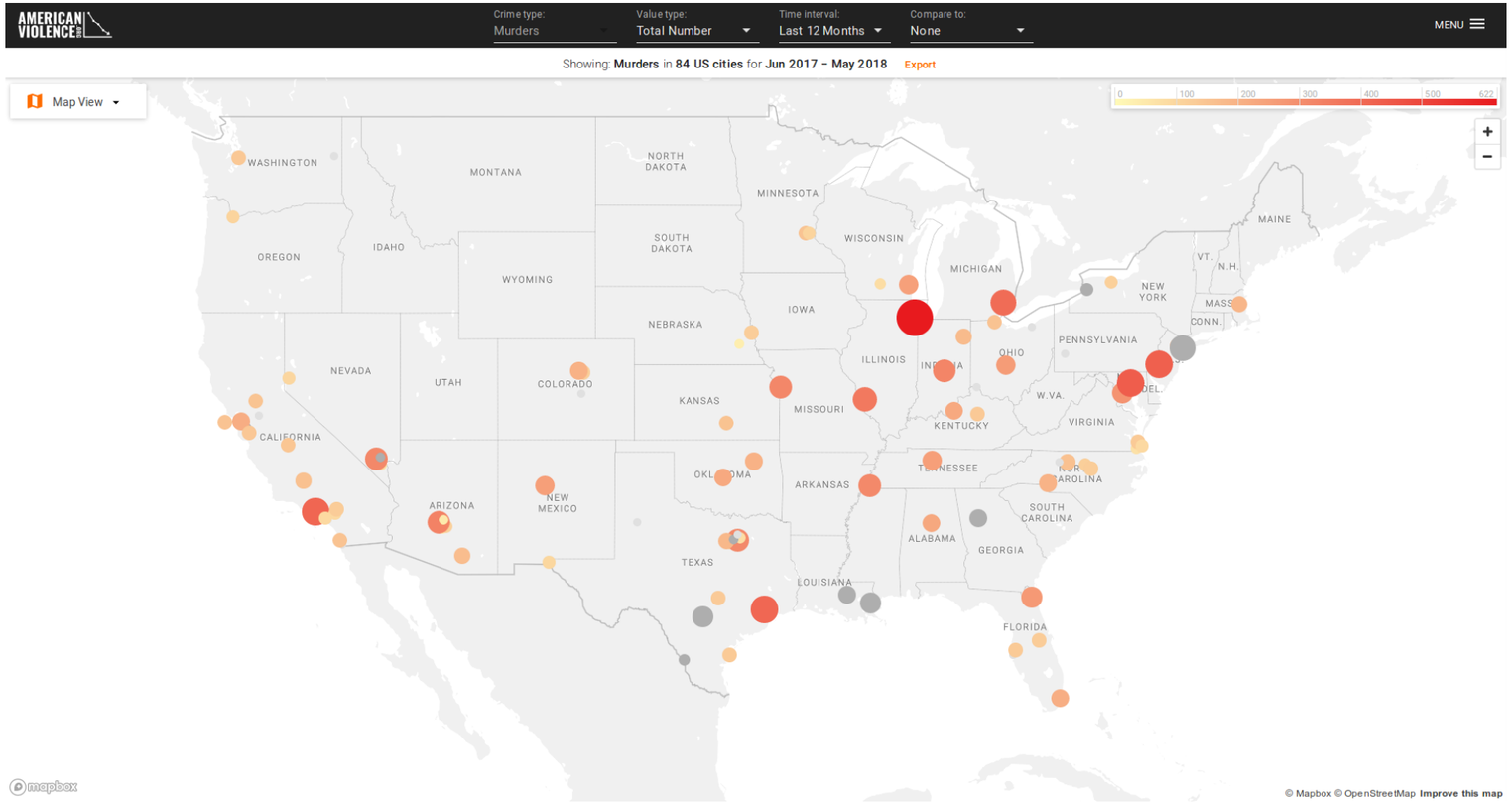
- Comparing prices
- Satisfaction of customers
- Generating potential leads
- ...and much more!

# It's Personal

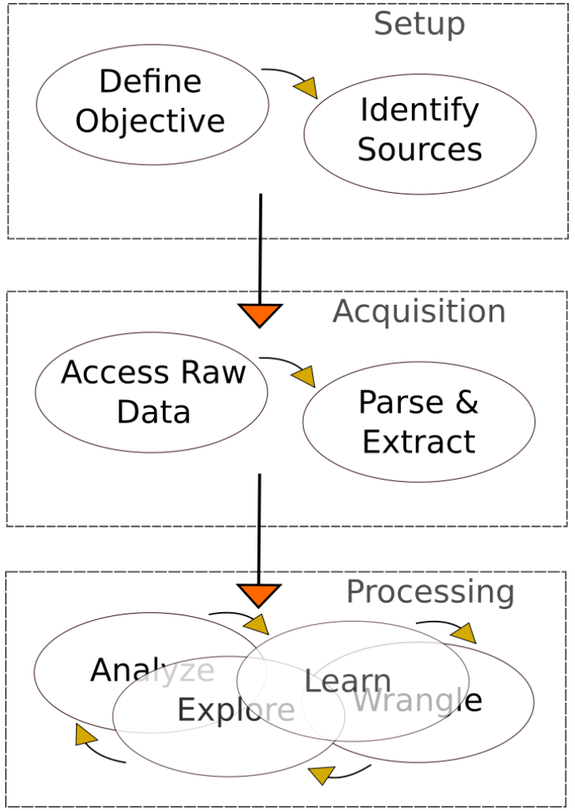
**What could you do?**

- 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!

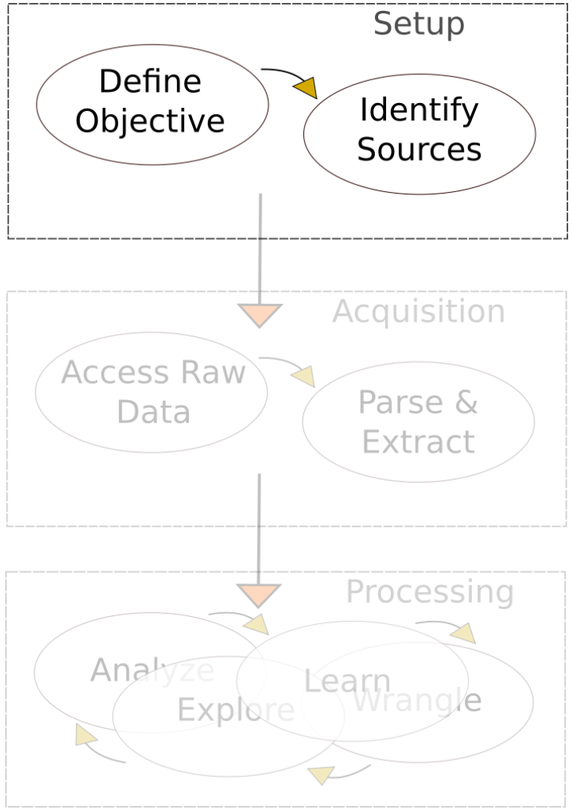
# About My Work



# Pipe Dream


Setup
Define Objective
Identify Sources
Acquisition
Access Raw Data
Parse & Extract
Processing
Analyze
Explore
Learn
Wrangle

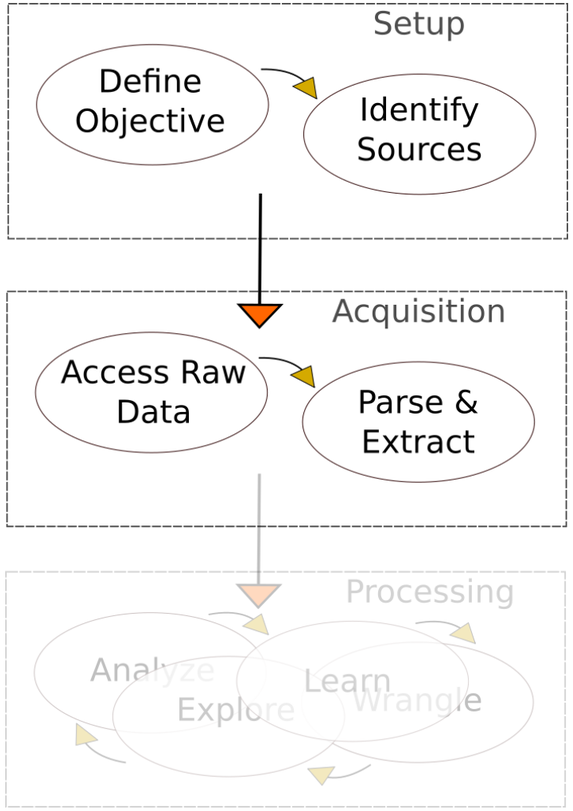
# Pipe Dream: Setup




**Setup**

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

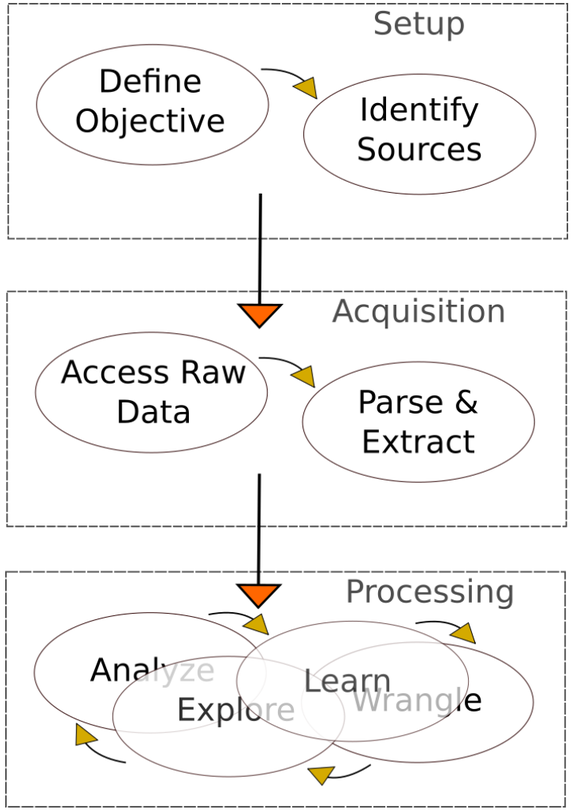
# Pipe Dream: Acquisition




**Acquisition**

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

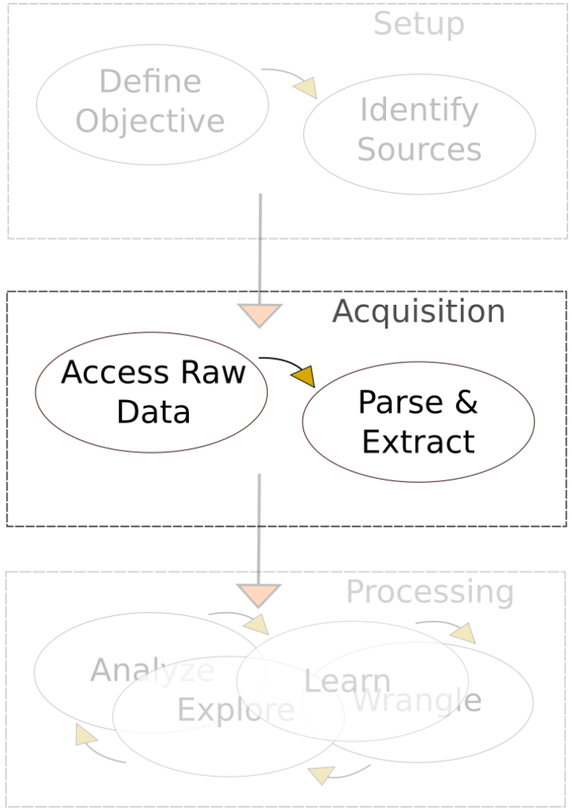
# Pipe Dream: Processing




**Processing**

- Many options!

# How do you do?




**Our Focus**

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

WEB SCRAPING WITH PYTHON

# Are you in?

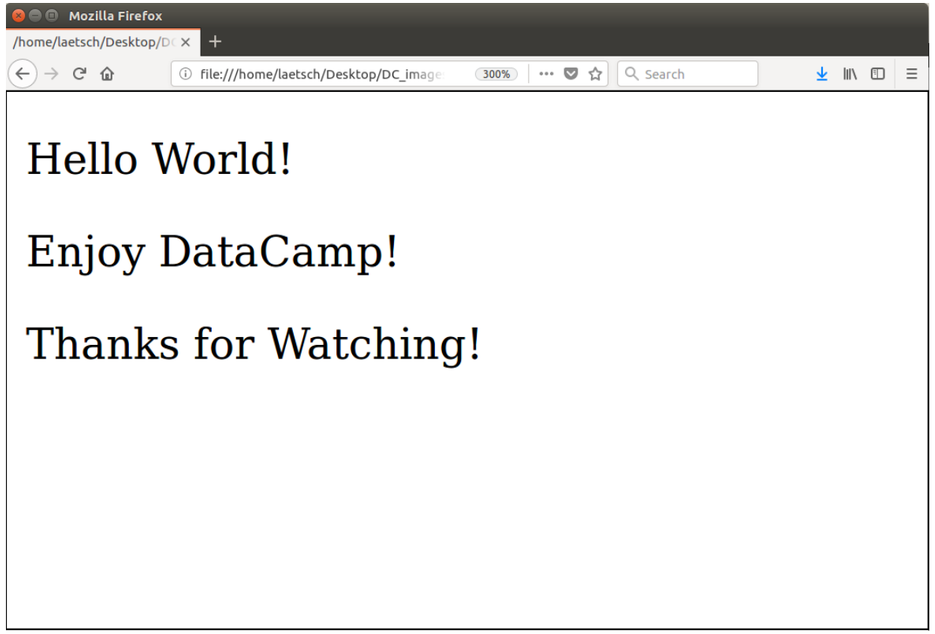
WEB SCRAPING WITH PYTHON

# HyperText Markup Language

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>
```



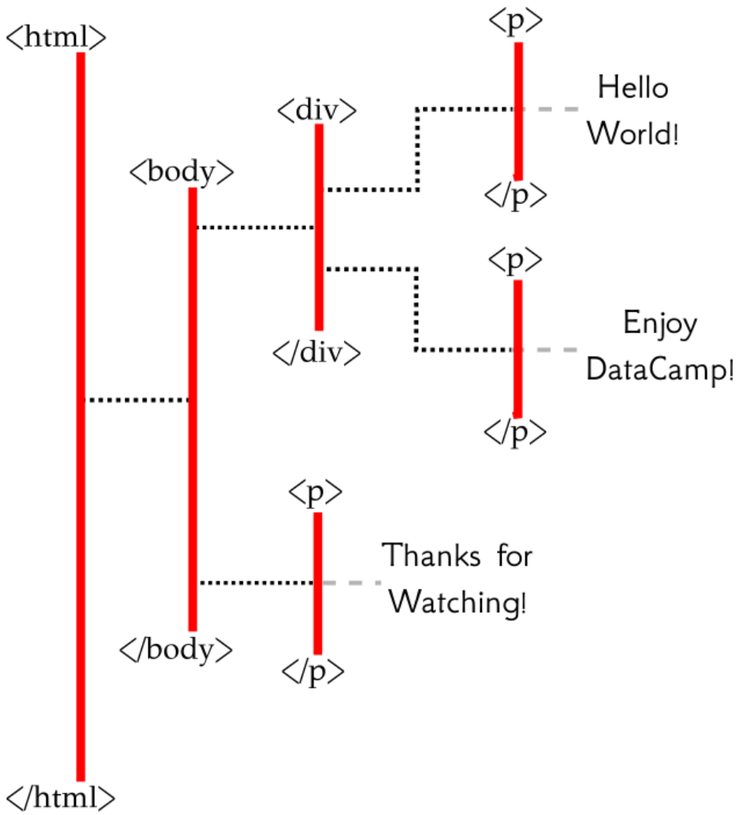
# HTML tags

```
<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>`

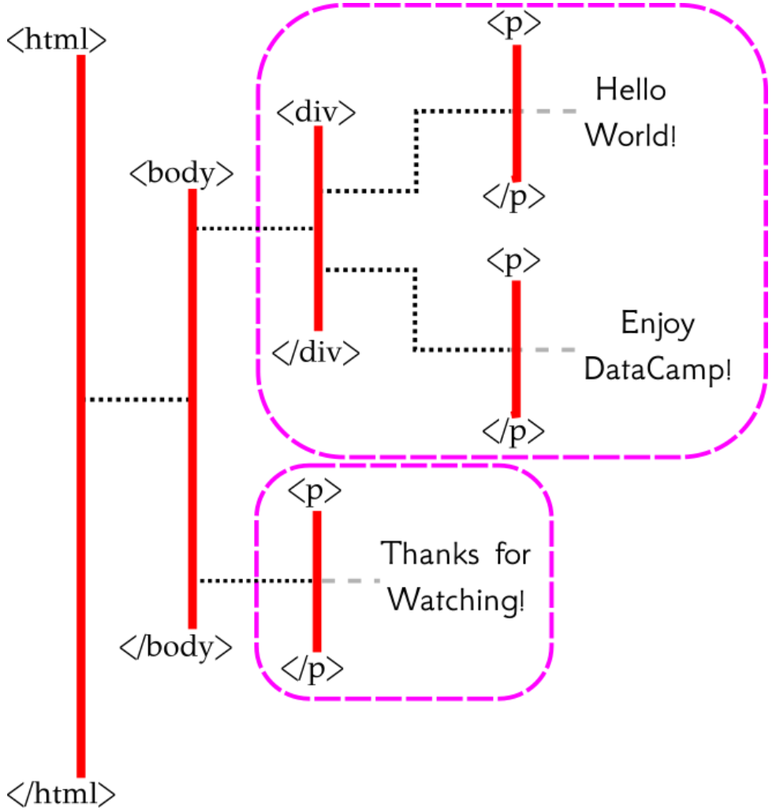
# The HTML tree

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



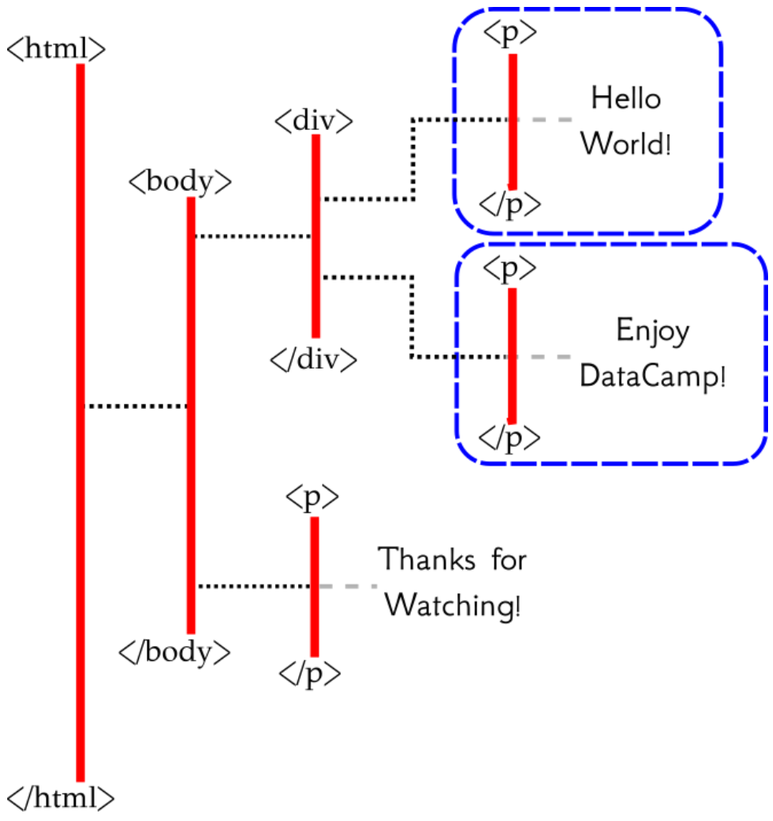
# The HTML tree: Example 1

```
<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>
```



WEB SCRAPING WITH PYTHON

# Introduction to HTML Outro

WEB SCRAPING WITH PYTHON

# HTML Tags and Attributes

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>
```

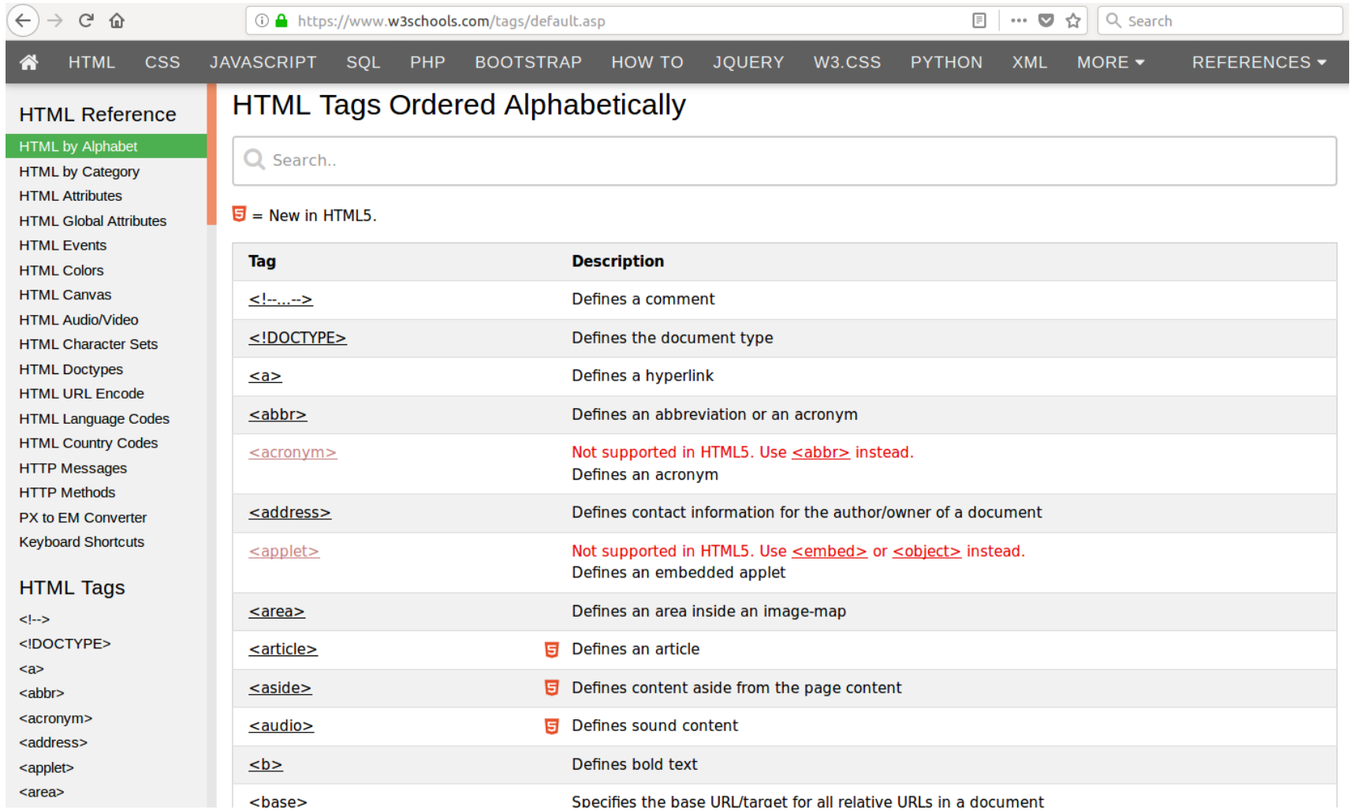
- **id** attribute should be unique
- **class** attribute doesn't need to be unique

# "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



WEB SCRAPING WITH PYTHON

# Et Tu, Attributes?

WEB SCRAPING WITH PYTHON

# Crash Course X

Thomas Laetsch
Data Scientist, NYU

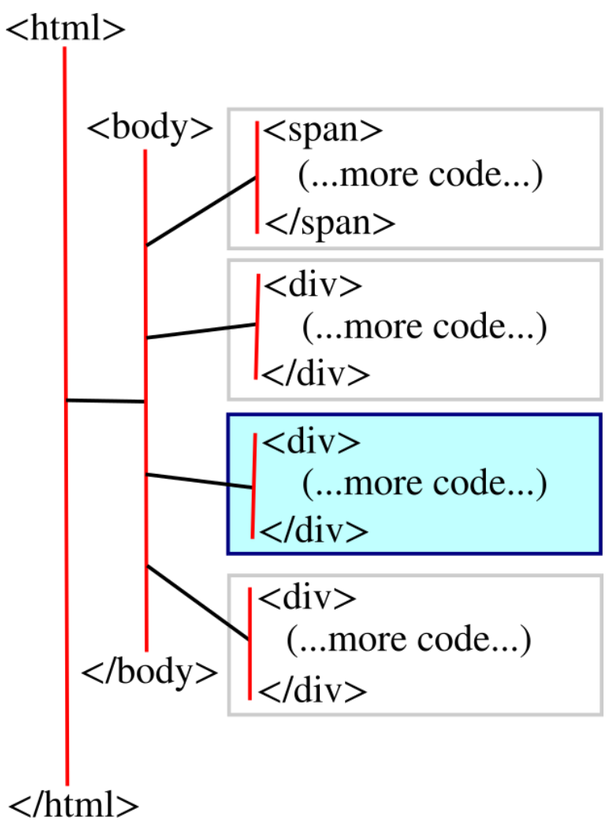
# Another Slasher Video?

```
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`
```

WEB SCRAPING WITH PYTHON

# Ex(path)celent

WEB SCRAPING WITH PYTHON

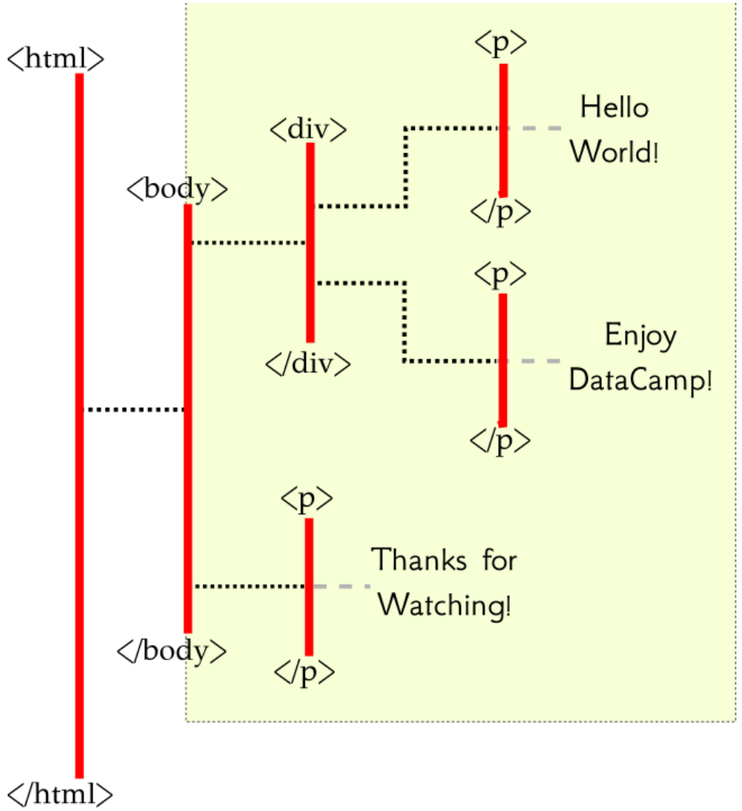
# XPath Navigation

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



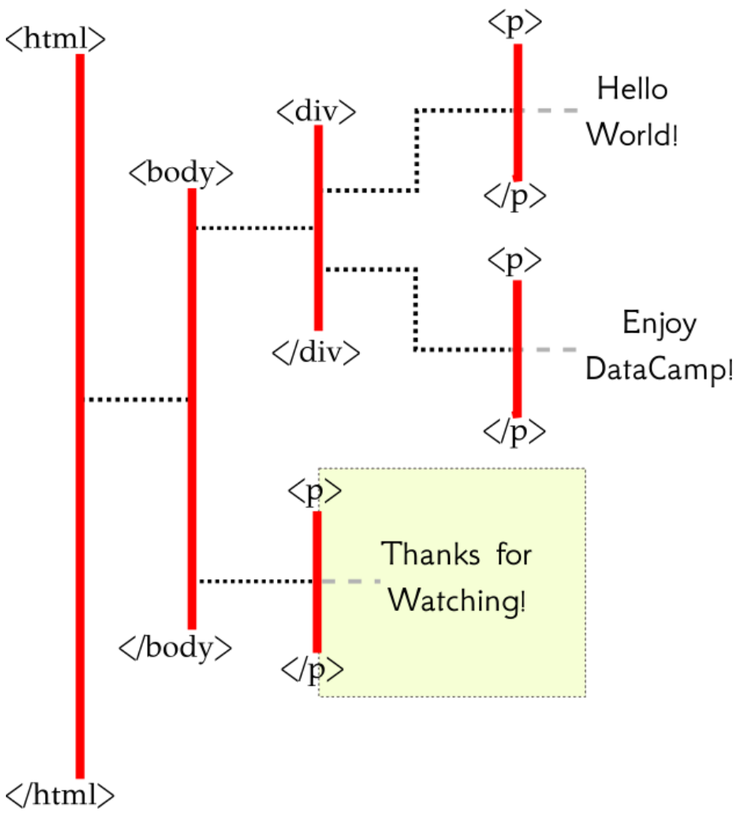

```
xpath = '/html/body'
```

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

- Give the same selection

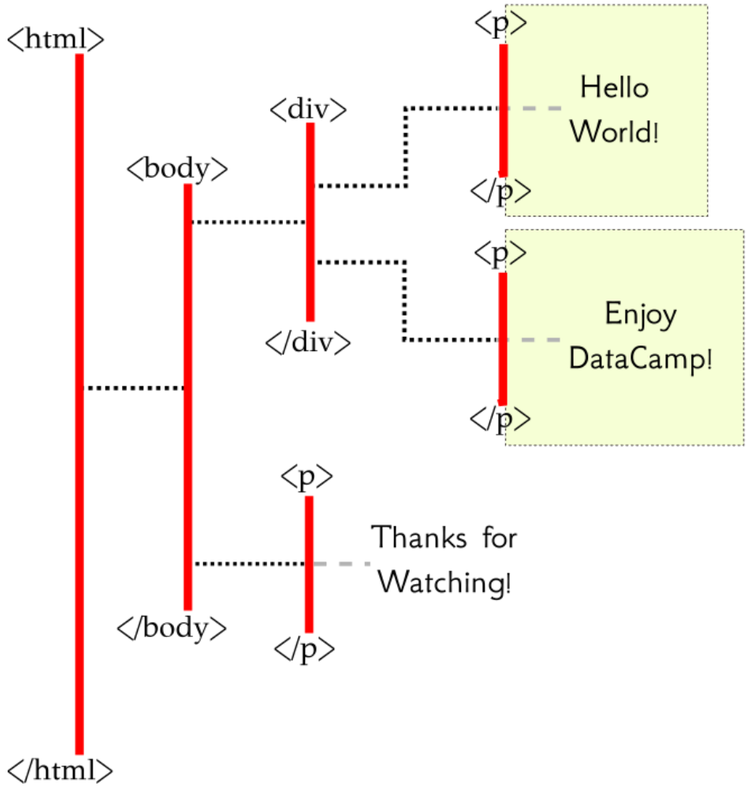
# A Body of P

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

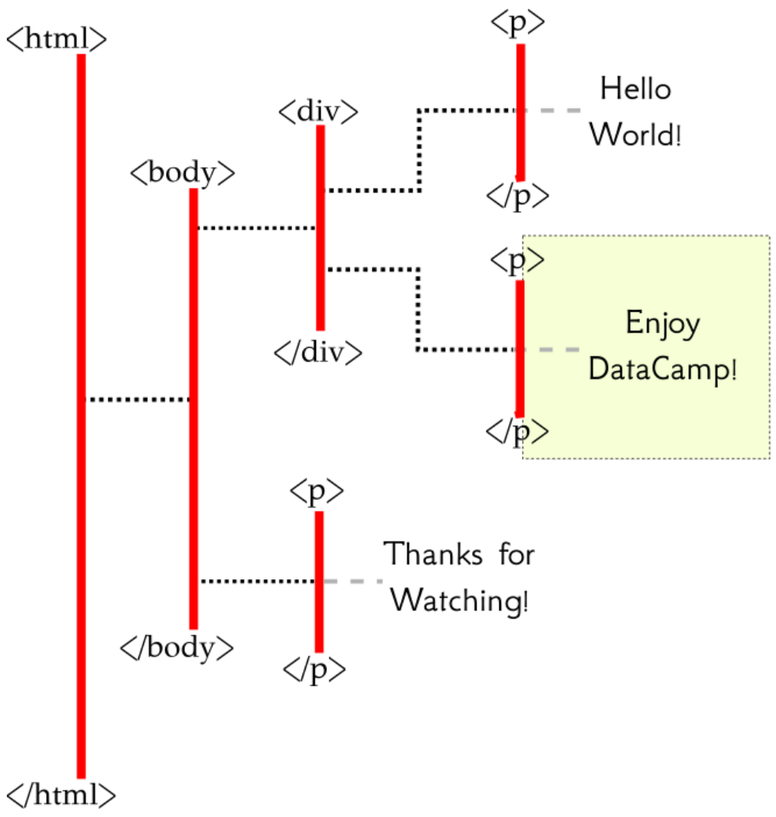


# The Birds and the Ps

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

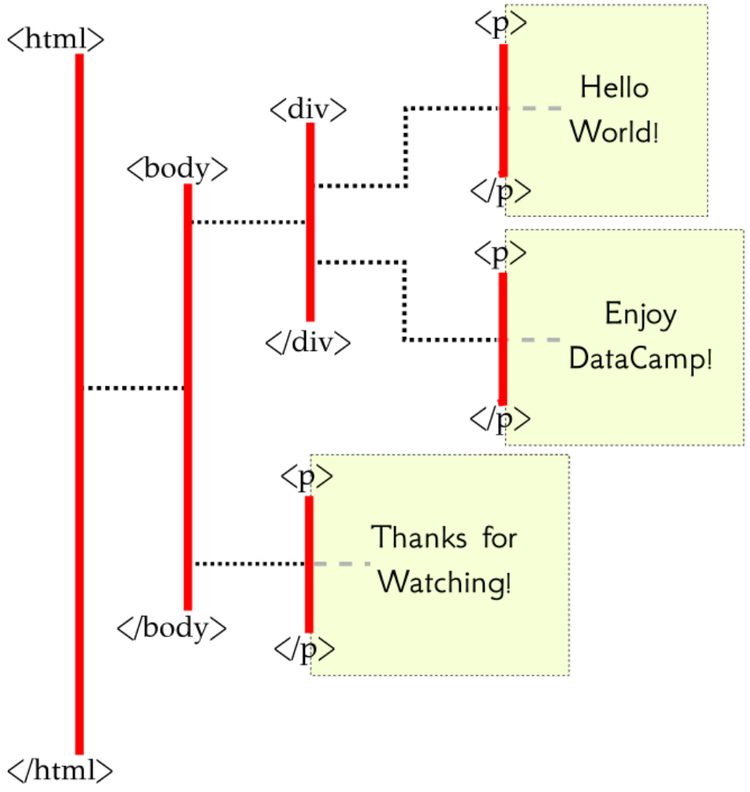



```
xpath = '/html/body/div/p[2]'
```

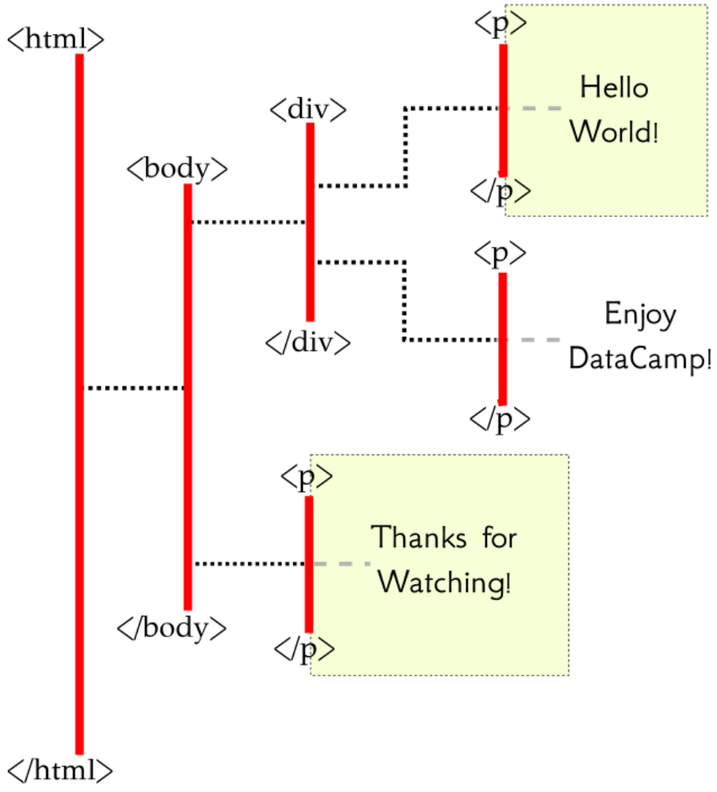


# Double Slashing the Brackets

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



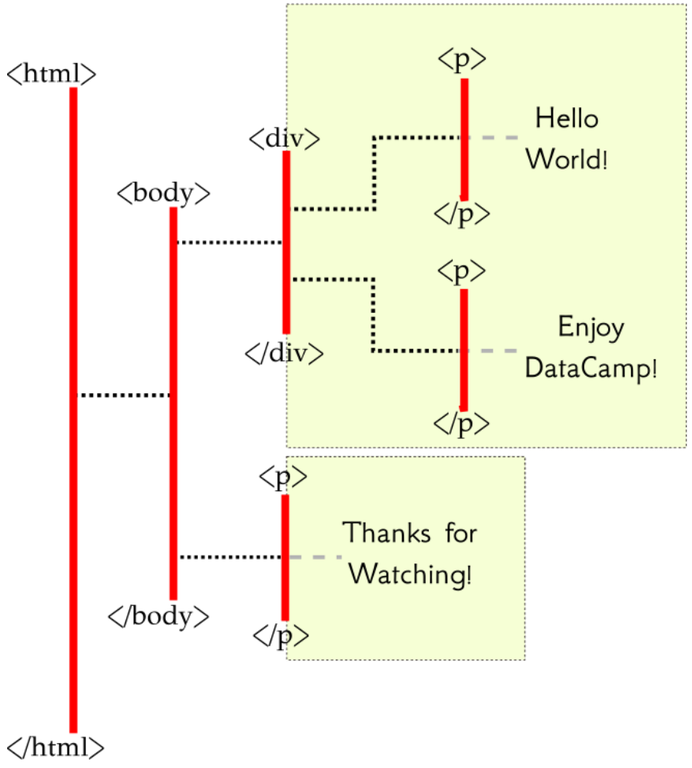

```
xpath = '//p[1]'
```



# The Wildcard

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

- The asterisks * is the "wildcard"



WEB SCRAPING WITH PYTHON

# Xposé

WEB SCRAPING WITH PYTHON

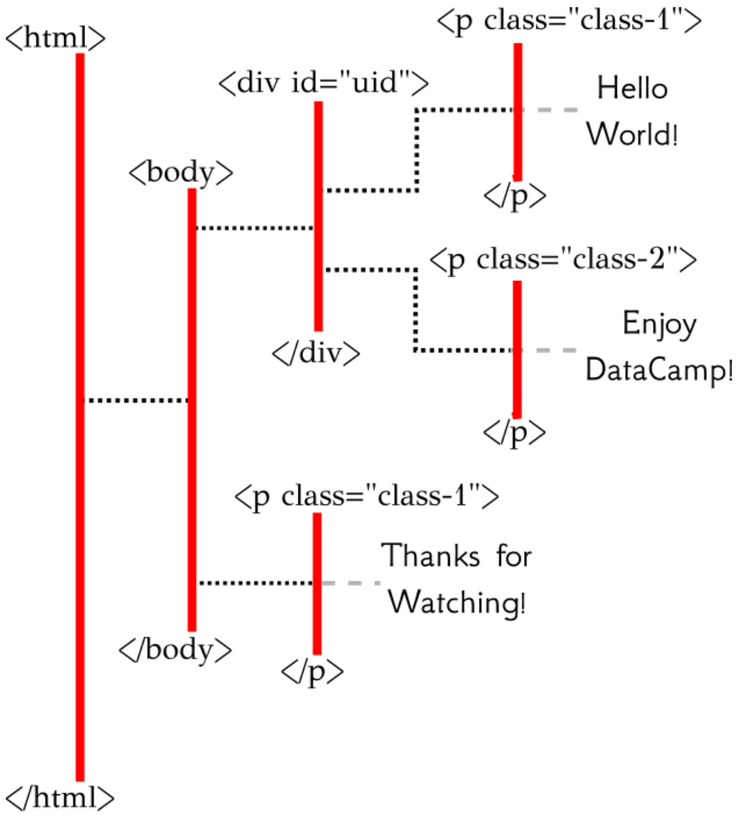
# Off the Beaten XPath

Thomas Laetsch
Data Scientist, NYU

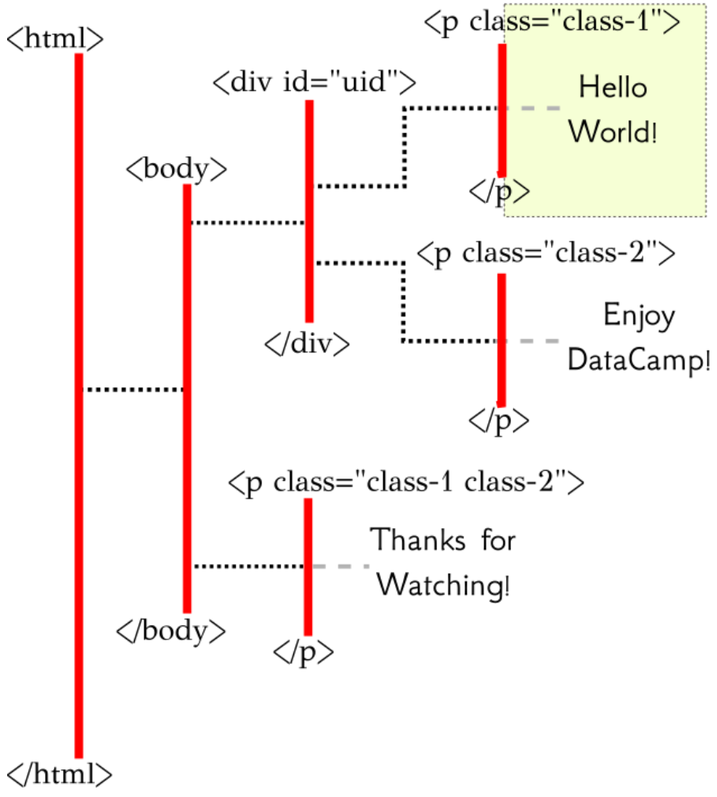
# (At)tribute

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

# Brackets and Attributes



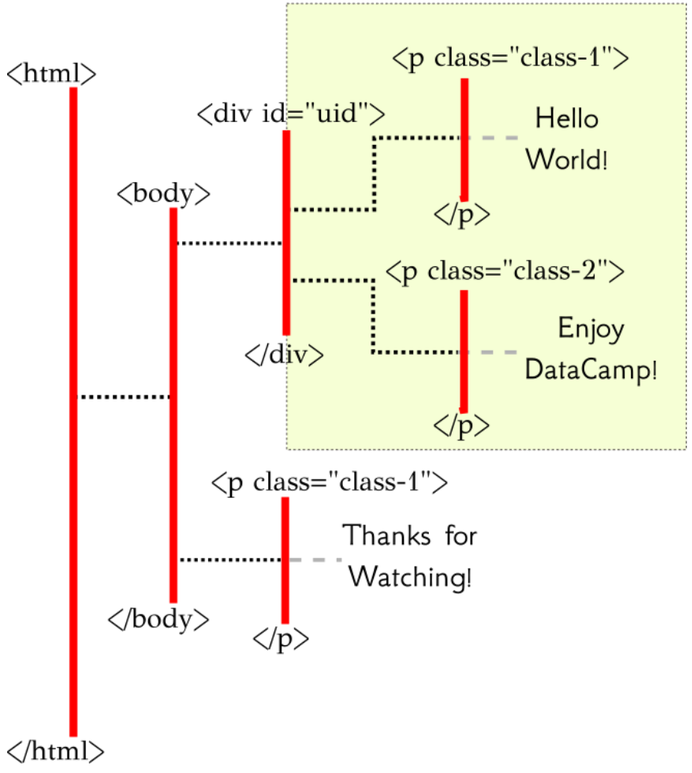
# Brackets and Attributes



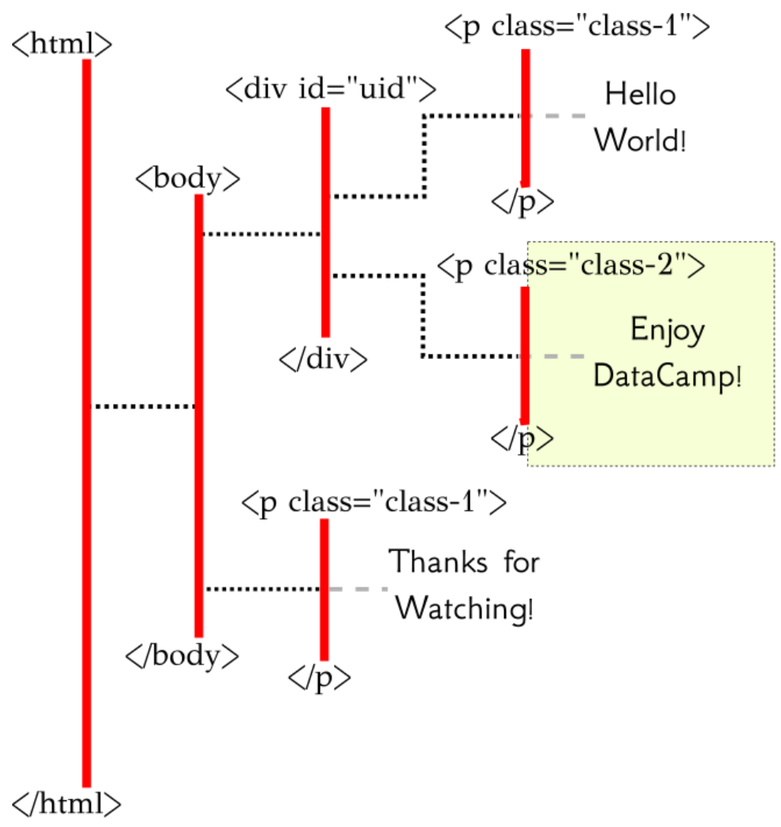

```
xpath = '//p[@class="class-1"]'
```

# 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")]'
```

 ... 

 ... 

 ... 

# Contain This

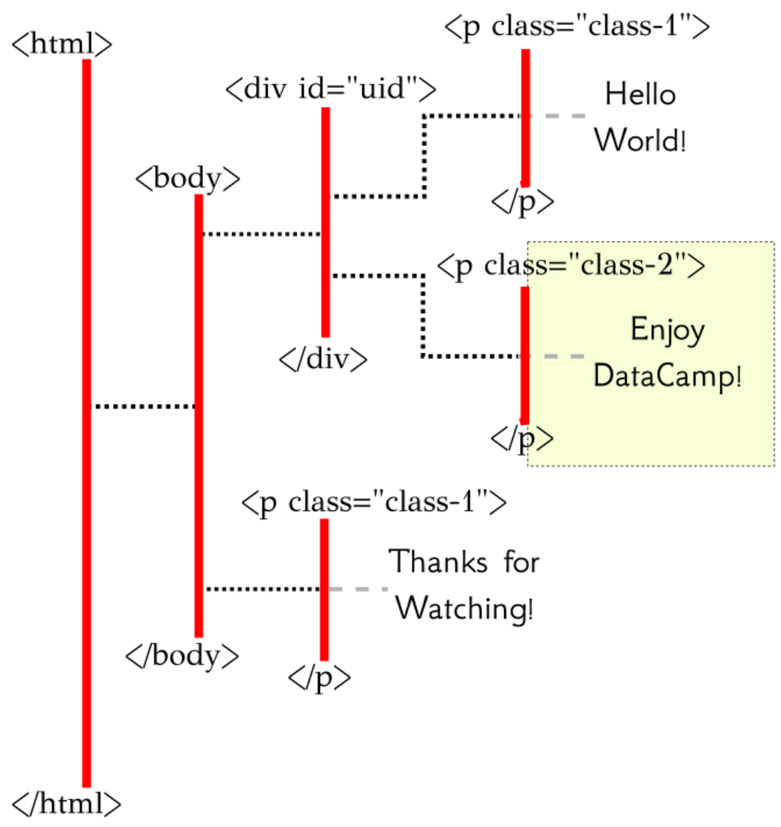
```
xpath = '//*[@class="class-1"]'
```

 ... 

 ... 

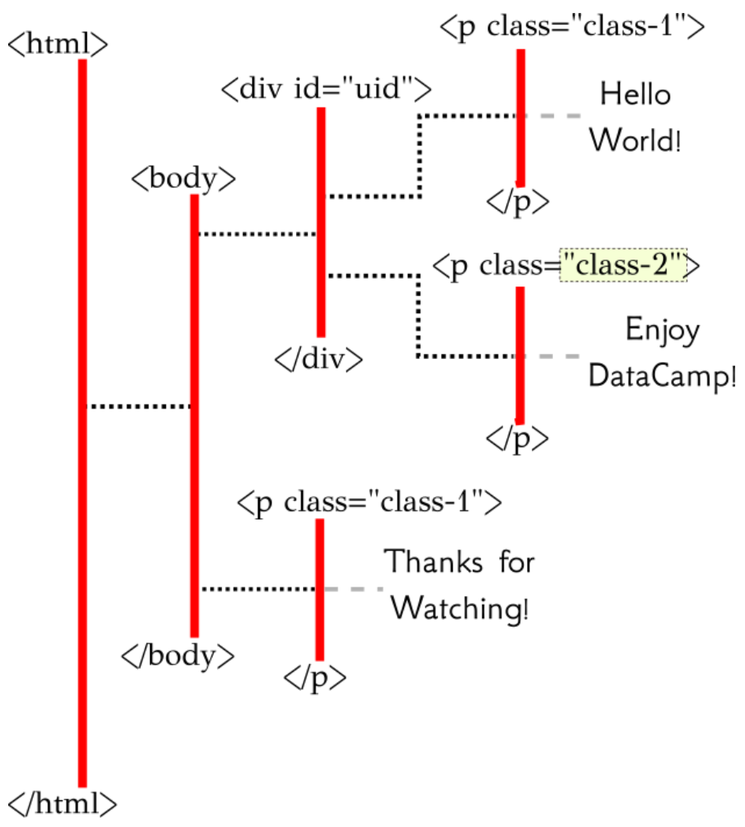
 ... 

# Get Classy




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

# Get Classy




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

WEB SCRAPING WITH PYTHON

# End of the Path

WEB SCRAPING WITH PYTHON

# Introduction to the scrapy Selector

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>'
```

WEB SCRAPING WITH PYTHON

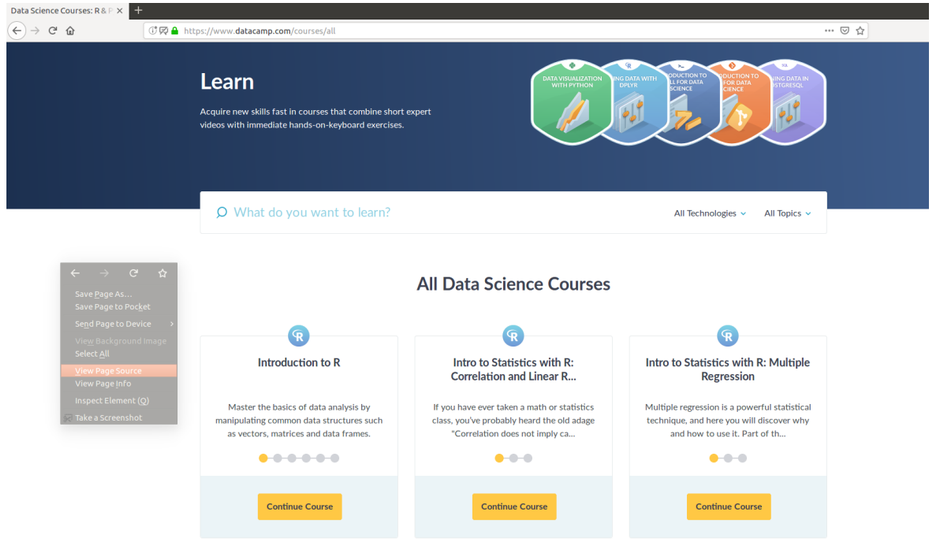
# Select This Course!

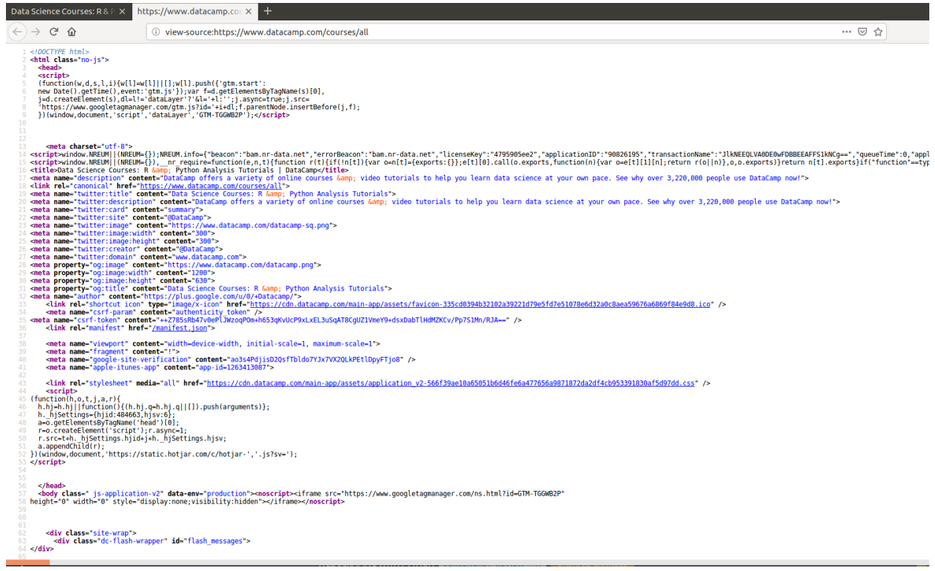
WEB SCRAPING WITH PYTHON

# "Inspecting the HTML"

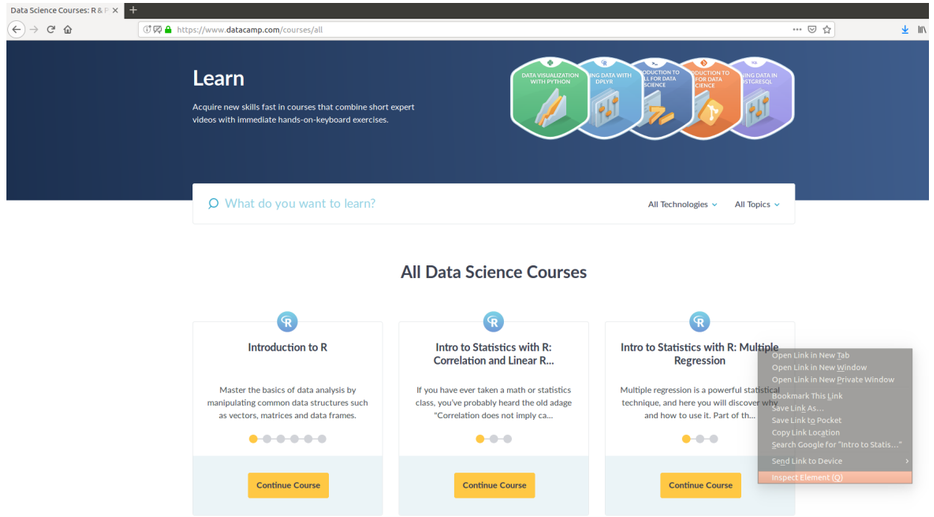
Thomas Laetsch, PhD
Data Scientist, NYU

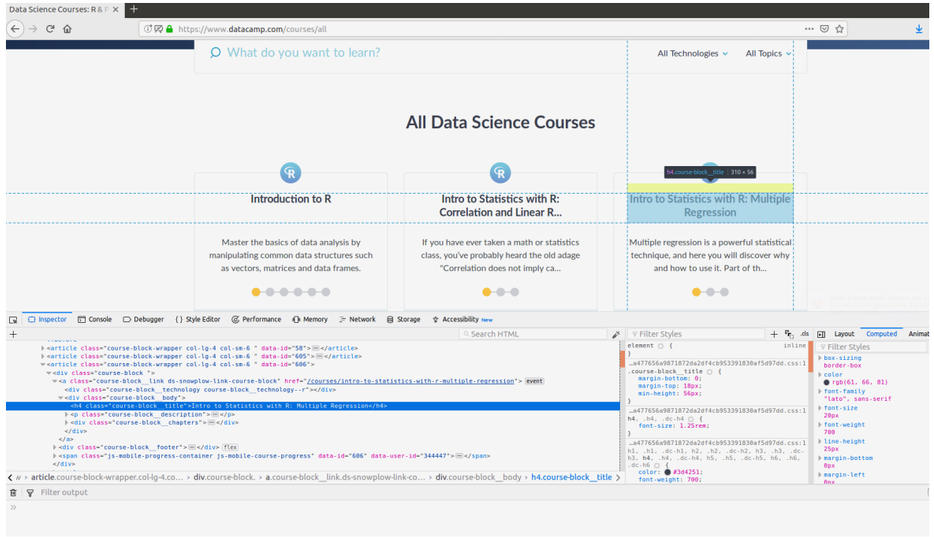
# "Source" = HTML Code



# Inspecting Elements



# 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 )
```

WEB SCRAPING WITH PYTHON

# You Know Our Secrets

WEB SCRAPING WITH PYTHON

# CSS Locators

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 belonges to `class1`:

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

# Class Status

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

 ... 

 ... 

 ... 

# Class Status

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

 ... 

 ... 

 ... 

# Class Status

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

 ... 

 ... 

 ... 

# 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>' ]
```

WEB SCRAPING WITH PYTHON

# C(SS) You Soon!

WEB SCRAPING WITH PYTHON

# Attribute and Text Selection

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']
```

WEB SCRAPING WITH PYTHON

# Scoping the Colon

WEB SCRAPING WITH PYTHON

# Getting Ready to Crawl

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.

WEB SCRAPING WITH PYTHON

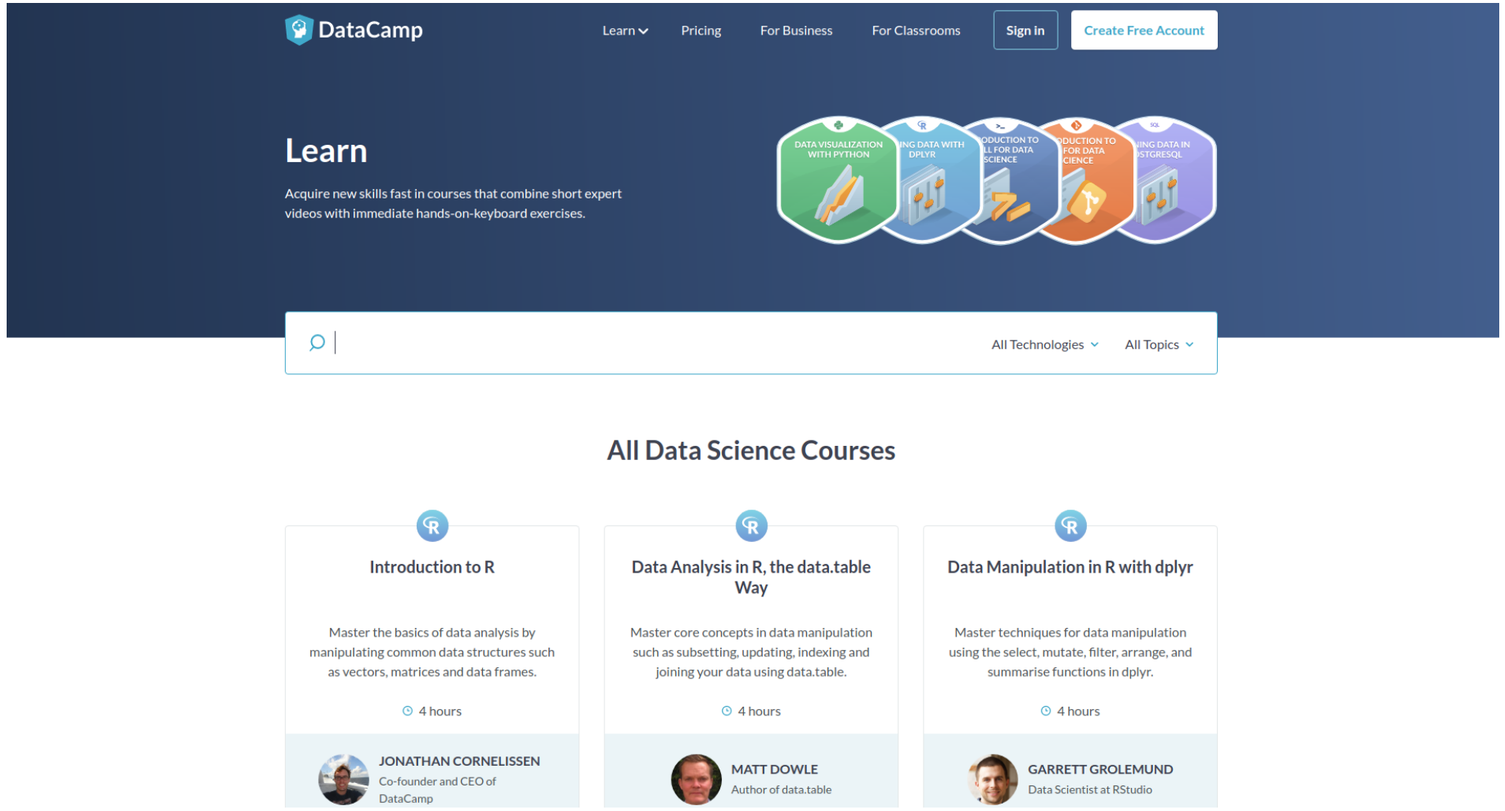
# In Response

WEB SCRAPING WITH PYTHON

# Scraping For Reals

Thomas Laetsch
Data Scientist, NYU

# DataCamp Site




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

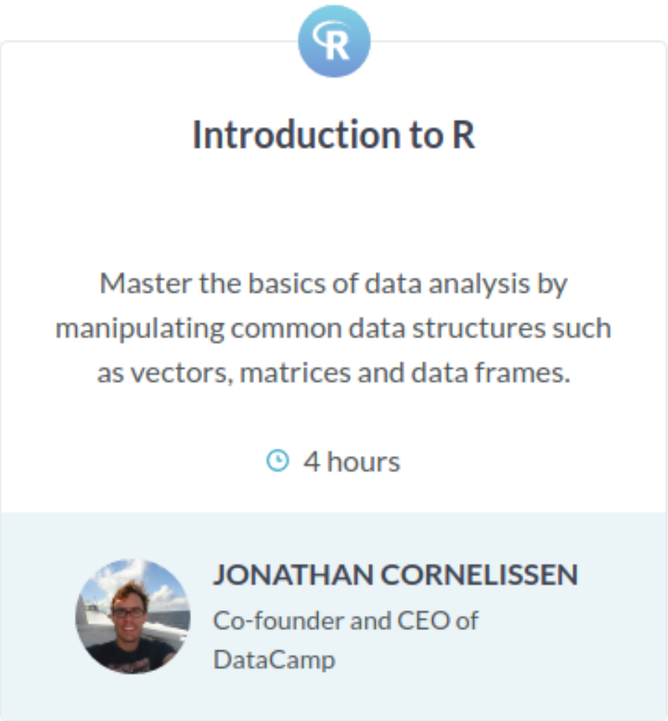
# What's the Div, Yo?

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

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

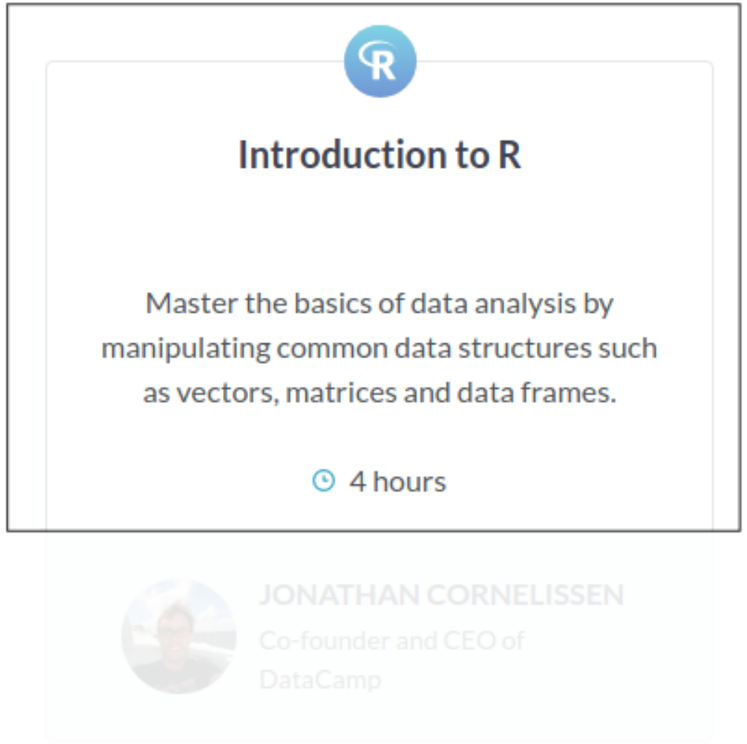
```
print( len(course_divs) )
>>> 185
```

# Inspecting course-block




```
first_div = course_divs[0]

children = first_div.xpath('./*')

print( len(children) )
>>> 3
```

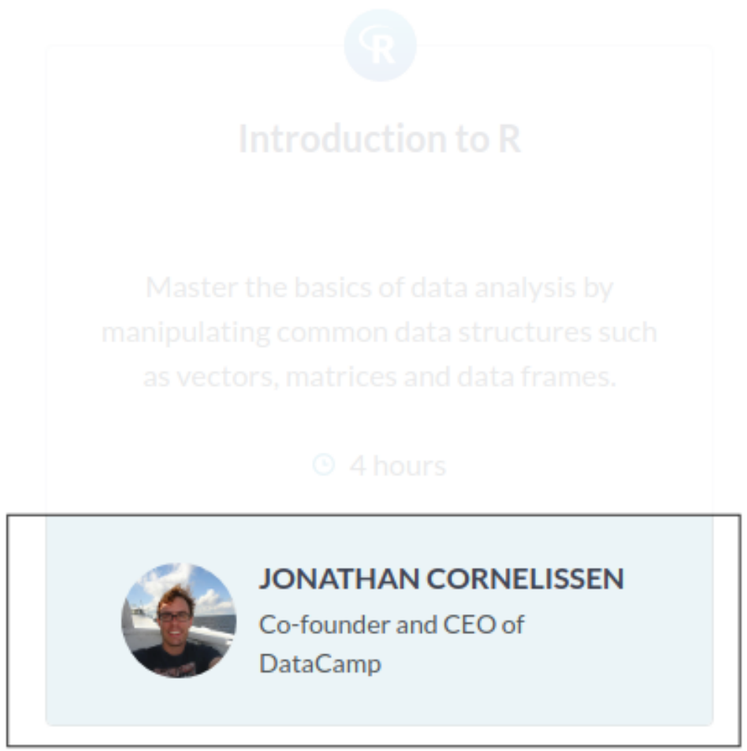
# The first child




```
first_div = course_divs[0]

children = first_div.xpath('./*')
```

```
first_child = children[0]

print( first_child.extract() )
>>> <a class=... />
```

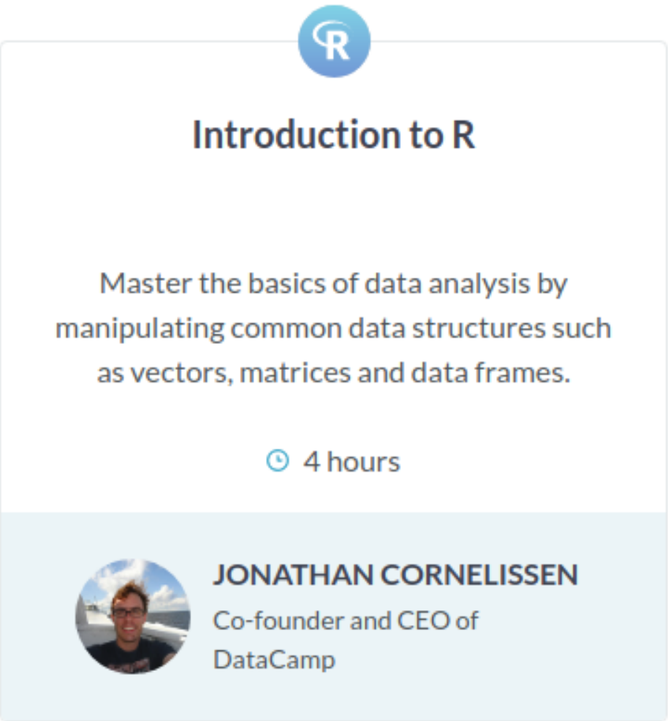
# The second child




```
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
...
```

WEB SCRAPING WITH PYTHON

# Links Achieved

WEB SCRAPING WITH PYTHON

# A Classy Spider

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

WEB SCRAPING WITH PYTHON

# We'll Weave the Web Together

WEB SCRAPING WITH PYTHON

# A Request for Service

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 )
```

WEB SCRAPING WITH PYTHON

# End Request

WEB SCRAPING WITH PYTHON

# Move Your Bloomin' Parse

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!

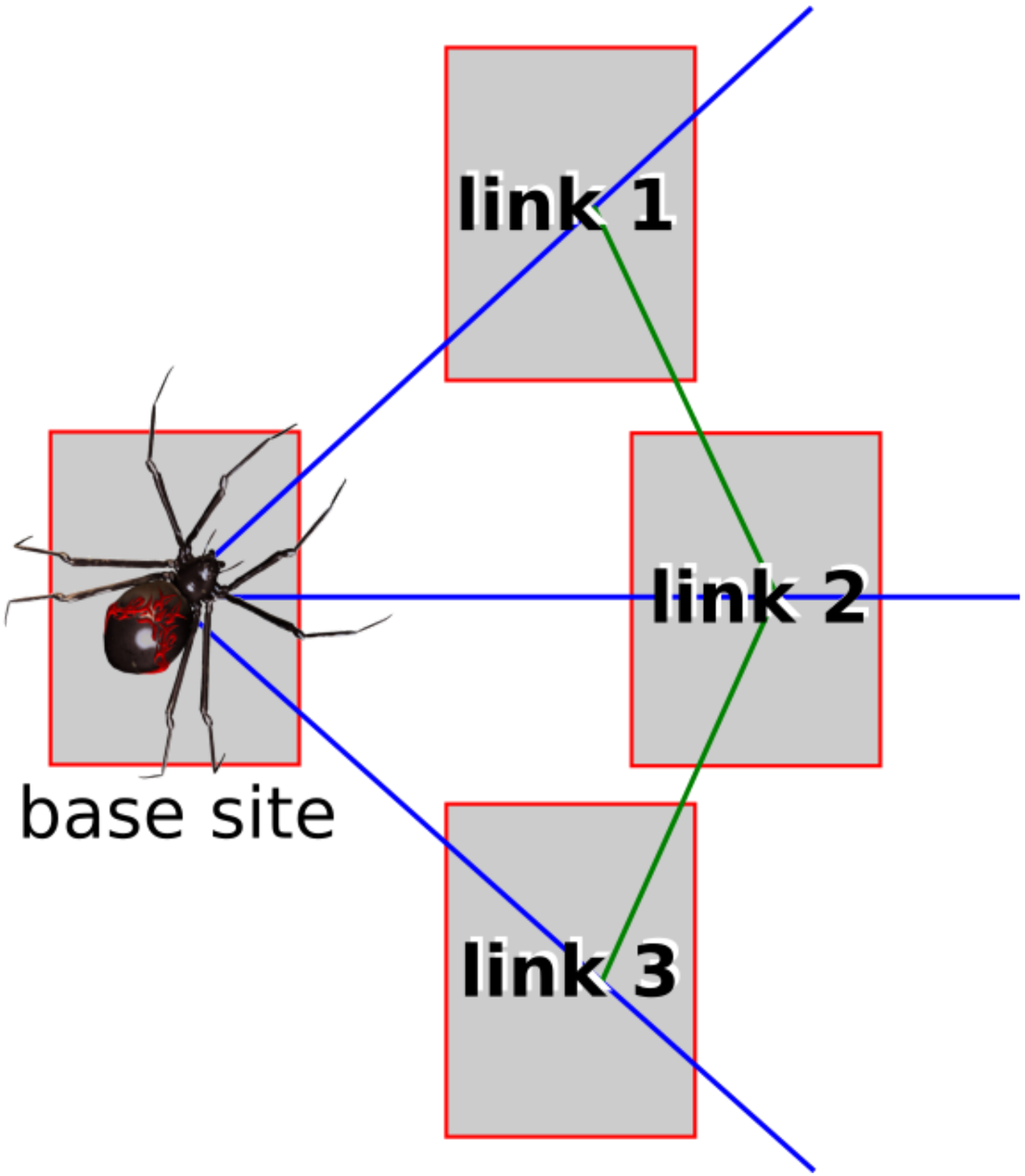
```
def parse( self, response ):

    # 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!
```


link 1
link 2
link 3
base site

WEB SCRAPING WITH PYTHON

# Johnny Parsin'

WEB SCRAPING WITH PYTHON

# Capstone

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
```

WEB SCRAPING WITH PYTHON

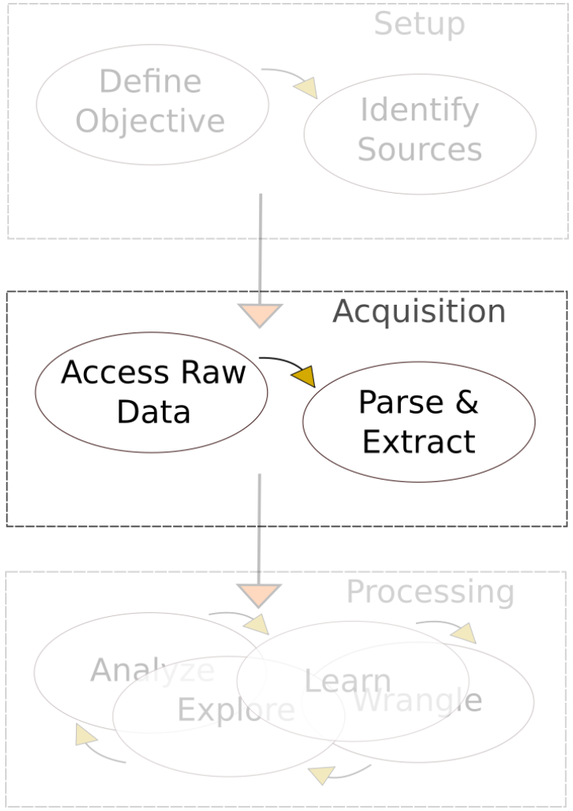
# It's time to Weave

WEB SCRAPING WITH PYTHON

# Stop Scratching and Start Scraping!

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

WEB SCRAPING WITH PYTHON

# EOT