Extracting Data From the Web Using Python

Web Data Extraction Workshop
D-Lab
UC Berkeley

Berkeley
UNIVERSITY OF CALIFORNIA

Learning Objectives

- The purpose of web scraping and APIs when to use either
- HTML & CSS and their use in web scraping.
- Use the requests and BeautifulSoup libraries to acquire and parse data from websites.
- Use a 3rd-party Python library to make API calls.
- Working with data pulled from APIs.

Why Webscrape

- Tons of web data useful for social scientists and humanists
  - social media
  - news media
  - government publications
  - organizational records

- Two kinds of ways to get data off the web
  - Webscraping – i.e. user-facing websites for humans
  - APIs – i.e. application-facing, for computers

Webscraping v. APIs

- Webscraping Benefits
  - Any content that can be viewed on a webpage can be scraped. Period
  - No API needed
  - No rate-limiting or authentication (usually). Your IP can be blocked (403)
- Webscraping Challenges
  - Rarely tailored for researchers
  - Messy, unstructured, inconsistent
  - Entirely site-dependent
- Rule of thumb:
  - Check for API first. If not available, scrape.

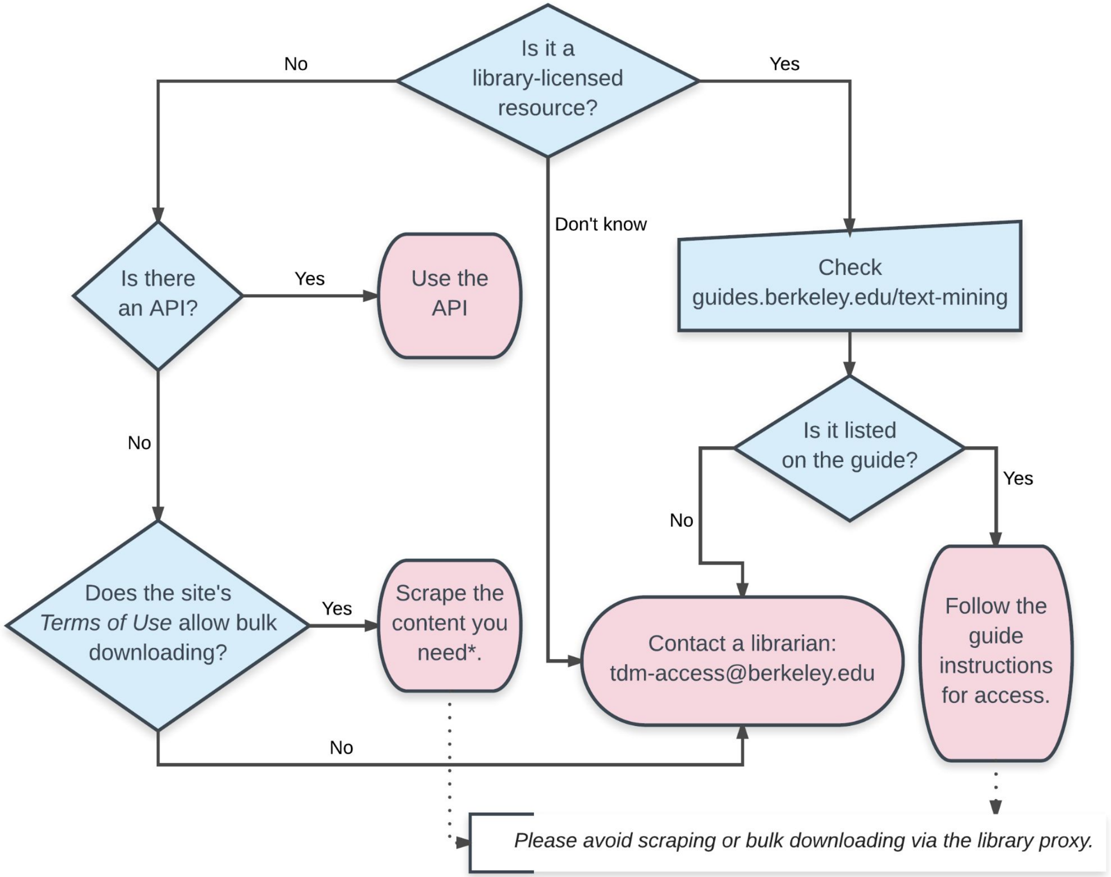
Some Disclaimers!!

- Check a site's terms and conditions before scraping.

- Be nice – don't hammer the site's server.
  - Add a time delay in between batches of scrapping.

- Sites change their layout all the time. Your scraper will break.

Workflow

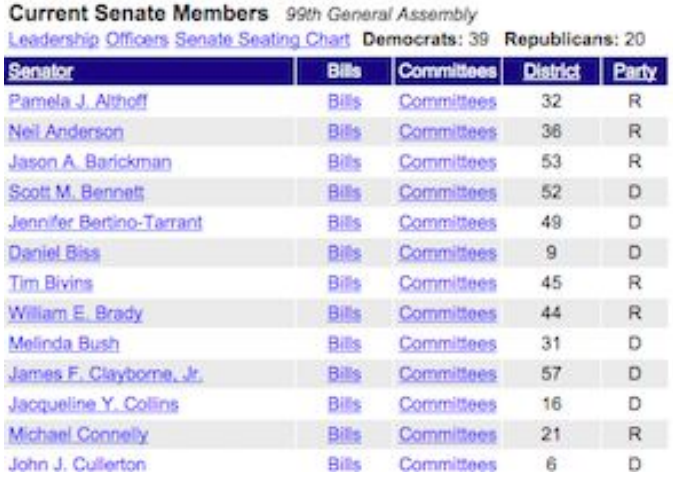

Is it a library-licensed resource?
No
Yes
Don't know
Is there an API?
Yes
Use the API
No
Check guides.berkeley.edu/text-mining
Is it listed on the guide?
No
Yes
Does the site's *Terms of Use* allow bulk downloading?
Yes
Scrape the content you need*.
No
Contact a librarian: tdm-access@berkeley.edu
Follow the guide instructions for access.
Please avoid scraping or bulk downloading via the library proxy.

Berkeley
UNIVERSITY OF CALIFORNIA

What's a website

- Some combination of codebase, database.
- The "front end" product is HTML + CSS stylesheets + javascript.
- Browser turns the left image into the right.

[illegible]



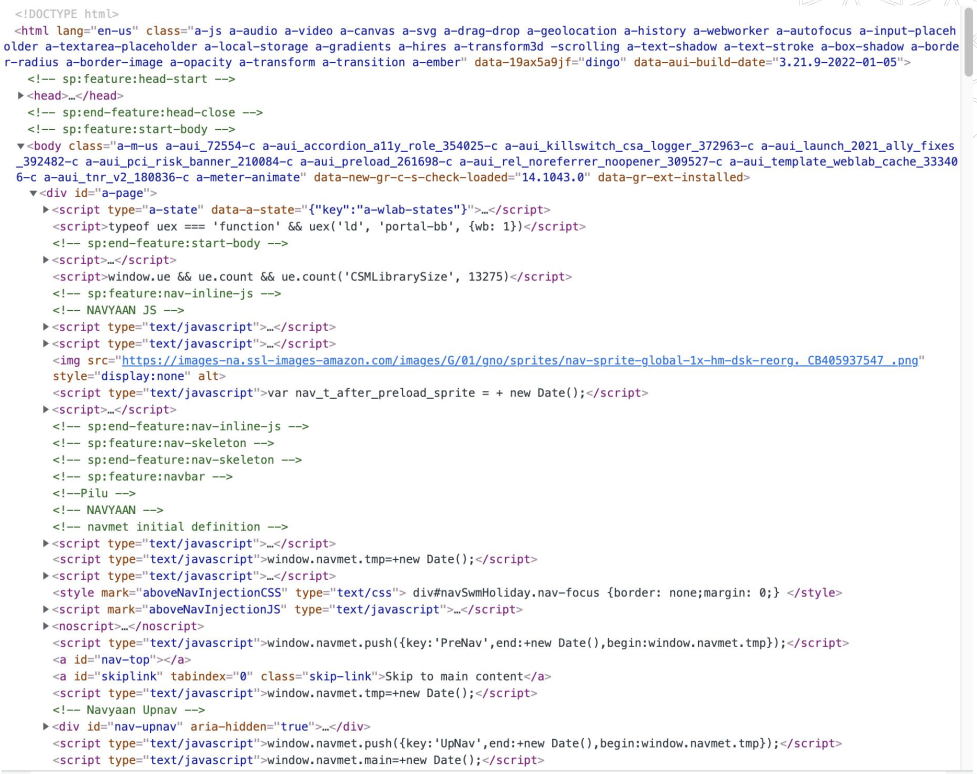
Current Senate Members *99th General Assembly*

Leadership Officers Senate Seating Chart **Democrats:** 39 **Republicans:** 20

Senator	Bills	Committees	District	Party
Pamela J. Althoff	Bills	Committees	32	R
Neil Anderson	Bills	Committees	36	R
Jason A. Barickman	Bills	Committees	53	R
Scott M. Bennett	Bills	Committees	52	D
Jennifer Bertino-Tarrant	Bills	Committees	49	D
Daniel Biss	Bills	Committees	9	D
Tim Bivins	Bills	Committees	45	R
William E. Brady	Bills	Committees	44	R
Melinda Bush	Bills	Committees	31	D
James F. Clayborne, Jr.	Bills	Committees	57	D
Jacqueline Y. Collins	Bills	Committees	16	D
Michael Connelly	Bills	Committees	21	R
John J. Cullerton	Bills	Committees	6	D

Webscraping returns HTML

- It's easy to pull HTML from a website
- It's much more difficult to find the information you want from that HTML
- So we have to learn how to **parse** HTML to find the data we want



```
<!DOCTYPE html>
<html lang="en-us" class="a-js a-audio a-video a-canvas a-svg a-drag-drop a-geolocation a-history a-webworker a-autofocus a-input-placeh
older a-textarea-placeholder a-local-storage a-gradients a-hires a-transform3d -scrolling a-text-shadow a-text-stroke a-box-shadow a-borde
r-radius a-border-image a-opacity a-transform a-transition a-ember" data-19ax5a9jf="dingo" data-aui-build-date="3.21.9-2022-01-05">
   <!-- sp:feature:head-start -->
 ▶<head>…</head>
   <!-- sp:end-feature:head-close -->
   <!-- sp:feature:start-body -->
 ▼<body class="a-m-us a-aui_72554-c a-aui_accordion_a11y_role_354025-c a-aui_killswitch_csa_logger_372963-c a-aui_launch_2021_ally_fixes
 _392482-c a-aui_pci_risk_banner_210084-c a-aui_preload_261698-c a-aui_rel_noreferrer_noopener_309527-c a-aui_template_weblab_cache_33340
 6-c a-aui_tnr_v2_180836-c a-meter-animate" data-new-gr-c-s-check-loaded="14.1043.0" data-gr-ext-installed>
   ▼<div id="a-page">
     ▶<script type="a-state" data-a-state="{"key":"a-wlab-states"}">…</script>
      <script>typeof uex === 'function' && uex('ld', 'portal-bb', {wb: 1})</script>
      <!-- sp:end-feature:start-body -->
     ▶<script>…</script>
      <script>window.ue && ue.count && ue.count('CSMLibrarySize', 13275)</script>
      <!-- sp:feature:nav-inline-js -->
      <!-- NAVYAAN JS -->
     ▶<script type="text/javascript">…</script>
     ▶<script type="text/javascript">…</script>
      <img src="https://images-na.ssl-images-amazon.com/images/G/01/gno/sprites/nav-sprite-global-1x-hm-dsk-reorg._CB405937547_.png"
      style="display:none" alt>
      <script type="text/javascript">var nav_t_after_preload_sprite = + new Date();</script>
     ▶<script>…</script>
      <!-- sp:end-feature:nav-inline-js -->
      <!-- sp:feature:nav-skeleton -->
      <!-- sp:end-feature:nav-skeleton -->
      <!-- sp:feature:navbar -->
      <!--Pilu -->
      <!-- NAVYAAN -->
      <!-- navmet initial definition -->
     ▶<script type="text/javascript">…</script>
      <script type="text/javascript">window.navmet.tmp=+new Date();</script>
     ▶<script type="text/javascript">…</script>
      <style mark="aboveNavInjectionCSS" type="text/css"> div#navSwmHoliday.nav-focus {border: none;margin: 0;} </style>
     ▶<script mark="aboveNavInjectionJS" type="text/javascript">…</script>
     ▶<noscript>…</noscript>
      <script type="text/javascript">window.navmet.push({key:'PreNav',end:+new Date(),begin:window.navmet.tmp});</script>
      <a id="nav-top"></a>
      <a id="skiplink" tabindex="0" class="skip-link">Skip to main content</a>
      <script type="text/javascript">window.navmet.tmp=+new Date();</script>
      <!-- Navyaan Upnav -->
     ▶<div id="nav-upnav" aria-hidden="true">…</div>
      <script type="text/javascript">window.navmet.push({key:'UpNav',end:+new Date(),begin:window.navmet.tmp});</script>
      <script type="text/javascript">window.navmet.main=+new Date();</script>
```

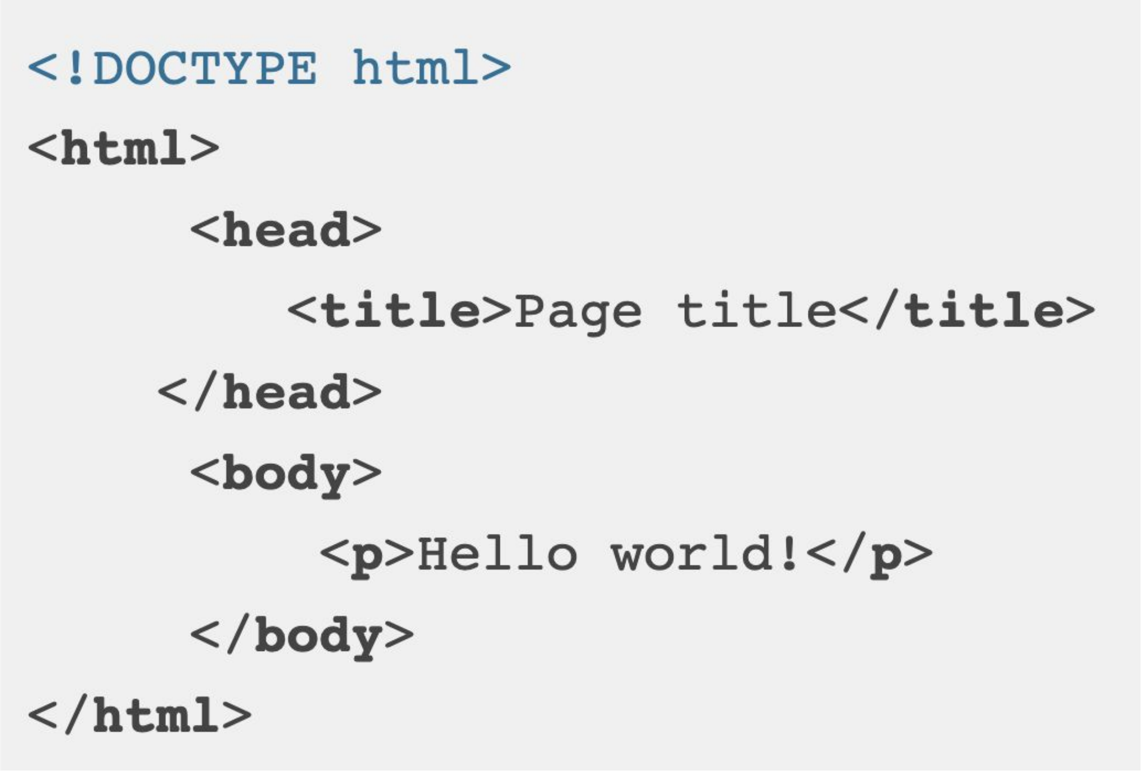
Basic strategy of webscraping:

1. Find out what kind of HTML element your data is in. (Use your browser‘s “inspector”)

2. Think about how you can differentiate those elements from other, similar elements in the webpage using HTML/CSS anatomy.

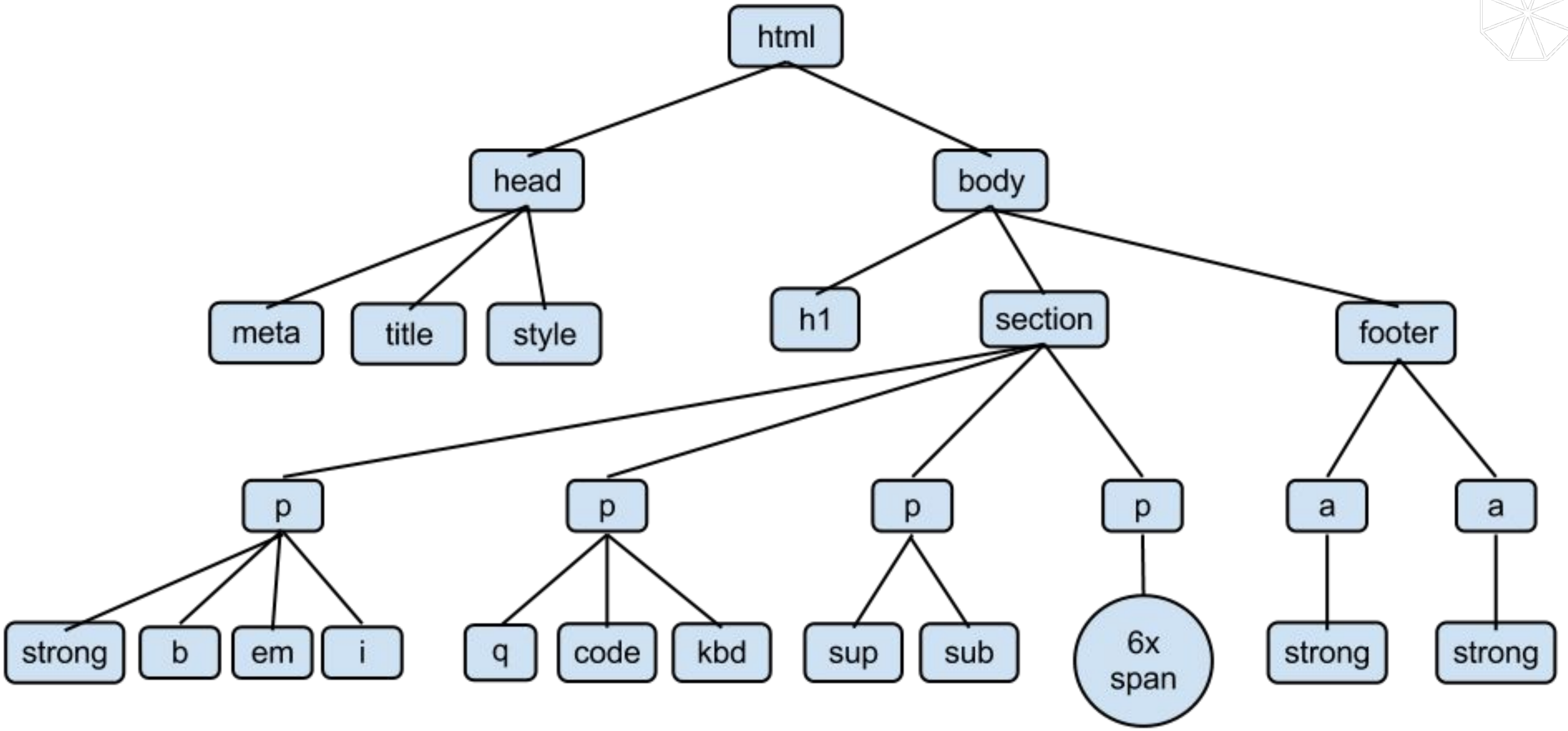
3. Use Python and add-on modules like BeautifulSoup to extract just that data.

HTML: Basic structure



```
<!DOCTYPE html>
<html>
     <head>
        <title>Page title</title>
    </head>
     <body>
         <p>Hello world!</p>
     </body>
</html>
```

HTML as a Tree

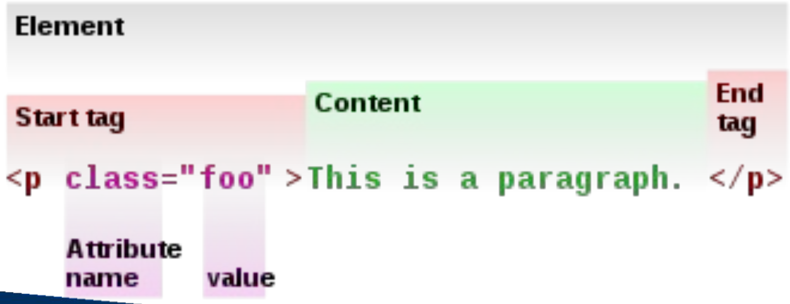


Each branch of the tree is called an **element**

HTML Elements

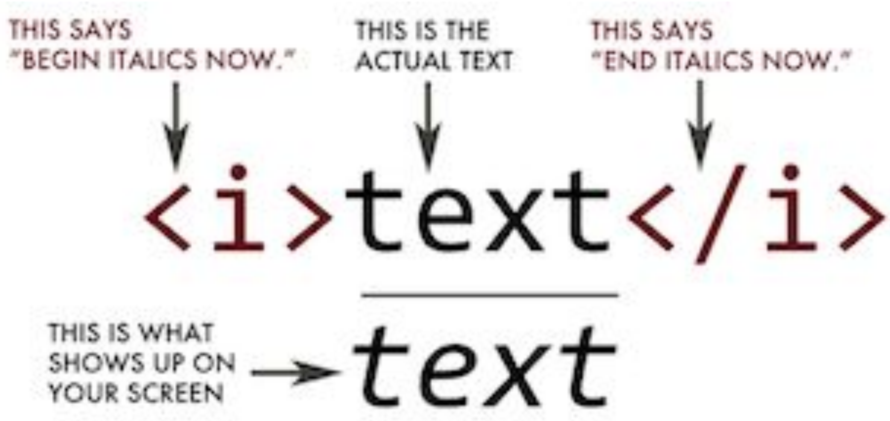
Generally speaking, an HTML element has three components:

1. Tags (starting and ending the element)
2. Attributes (giving information about the element)
3. Text, or Content (the text inside the element)



```
<p class="foo" >This is a paragraph. </p>
```

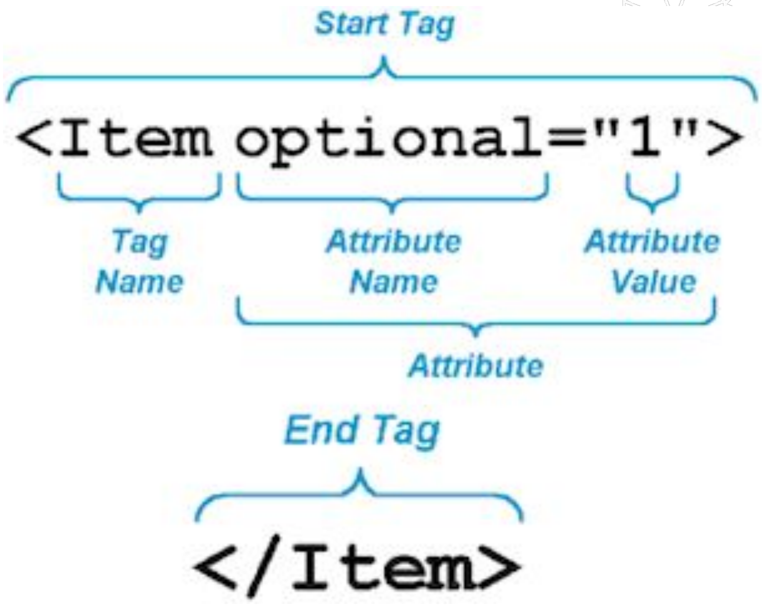
HTML: Tags


THIS SAYS "BEGIN ITALICS NOW."
THIS IS THE ACTUAL TEXT
THIS SAYS "END ITALICS NOW."
<i>text</i>
THIS IS WHAT SHOWS UP ON YOUR SCREEN
text

Tag	Meaning
<head>	page header (metadata, etc
<body>	holds all of the content
<p>	regular text (paragraph)
<h1>,<h2>,<h3>	header text, levels 1, 2, 3
ol,,<ul>,<li>	ordered list, unordered list, list item
<a href="page.html">	link to "page.html"
<table>,<tr>,<td>	table, table row, table item
<div>,<span>	general containers

HTML Attributes

- HTML elements can have attributes.
- Attributes provide additional information about an element.
- Attributes are always specified in the start tag.
- Attributes come in name/value pairs like: name="value"



```
<Item optional="1">
</Item>
```

Finding HTML

- Sometimes we can find the data we want just by using HTML tags or attributes (e.g, all the <a> tags)
- More often, this isn't enough: There might be 1000 <a> tags on a page. But maybe we want only the <a> tags inside of a <p> tag.
- This is where CSS comes in.

CSS (Cascading Style Sheet)

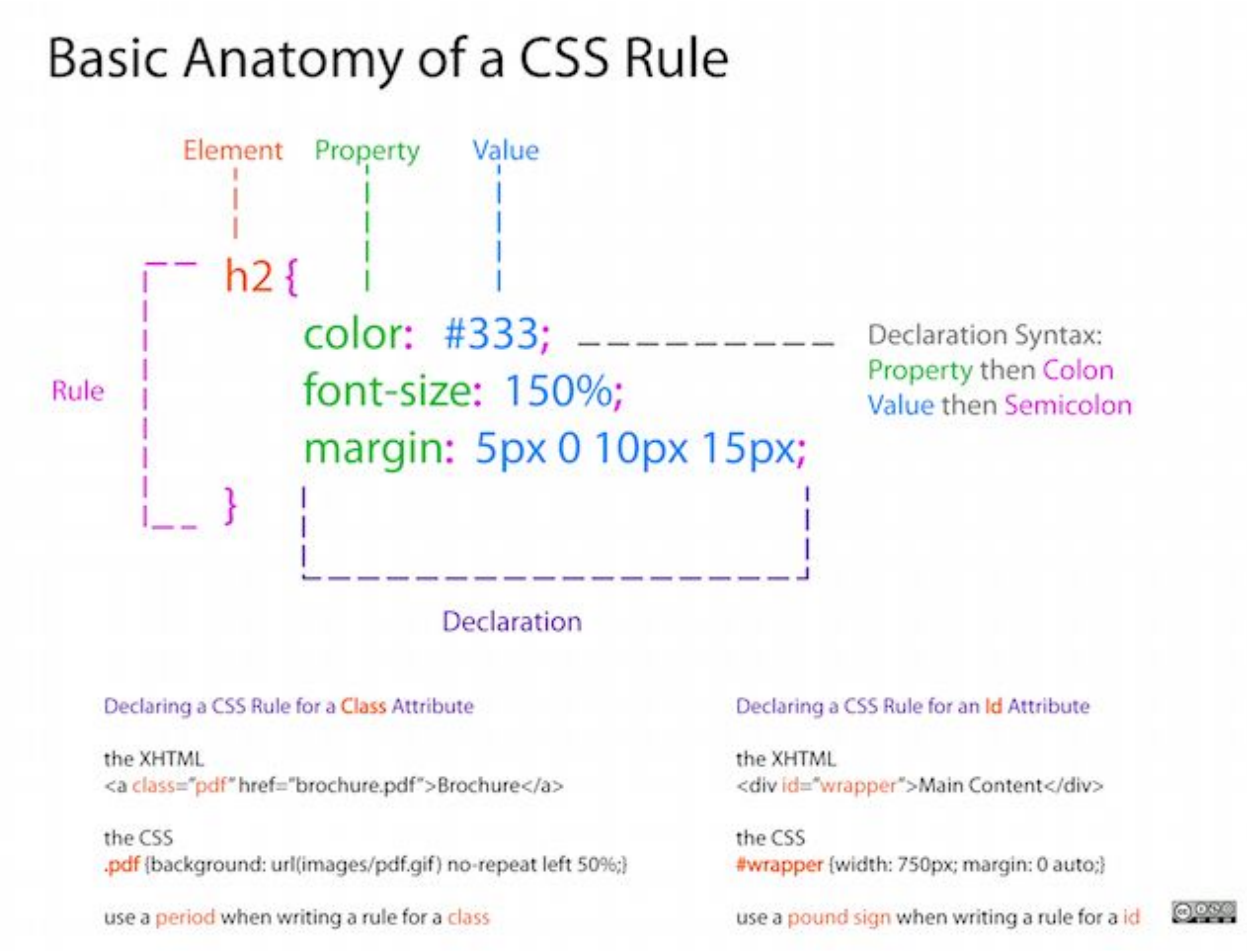
- CSS defines how HTML elements are to be displayed
- HTML came first. But it was only meant to define content, not format it. While HTML contains tags like <font> and <color>, this is a very inefficient way to develop a website.
- To solve this problem, CSS was created specifically to display content on a webpage. Now, one can change the look of an entire website just by changing one file.
- Most web designers litter the HTML markup with tons of classes and ids to provide "hooks" for their CSS.
- You can piggyback on these "hooks" to jump to the parts of the markup that contain the data you need.

CSS Selectors & Declarations

Type	**HTML**	**CSS Selector**
Element	<a>,	a p a
Class	<a class="blue">	.blue a.blue
ID	<a id="blue">	#blue a#blue

- Selector: *a*
- Property: *background-color*
- Value: *yellow*

CSS Hooks



Basic Anatomy of a CSS Rule

```
h2 {
        color:  #333;
        font-size:  150%;
        margin:  5px 0 10px 15px;
}
```

Declaration Syntax:
Property then Colon
Value then Semicolon

Declaring a CSS Rule for a Class Attribute

the XHTML
`<a class="pdf" href="brochure.pdf">Brochure</a>`

the CSS
`.pdf {background: url(images/pdf.gif) no-repeat left 50%;}`

use a period when writing a rule for a class

Declaring a CSS Rule for an Id Attribute

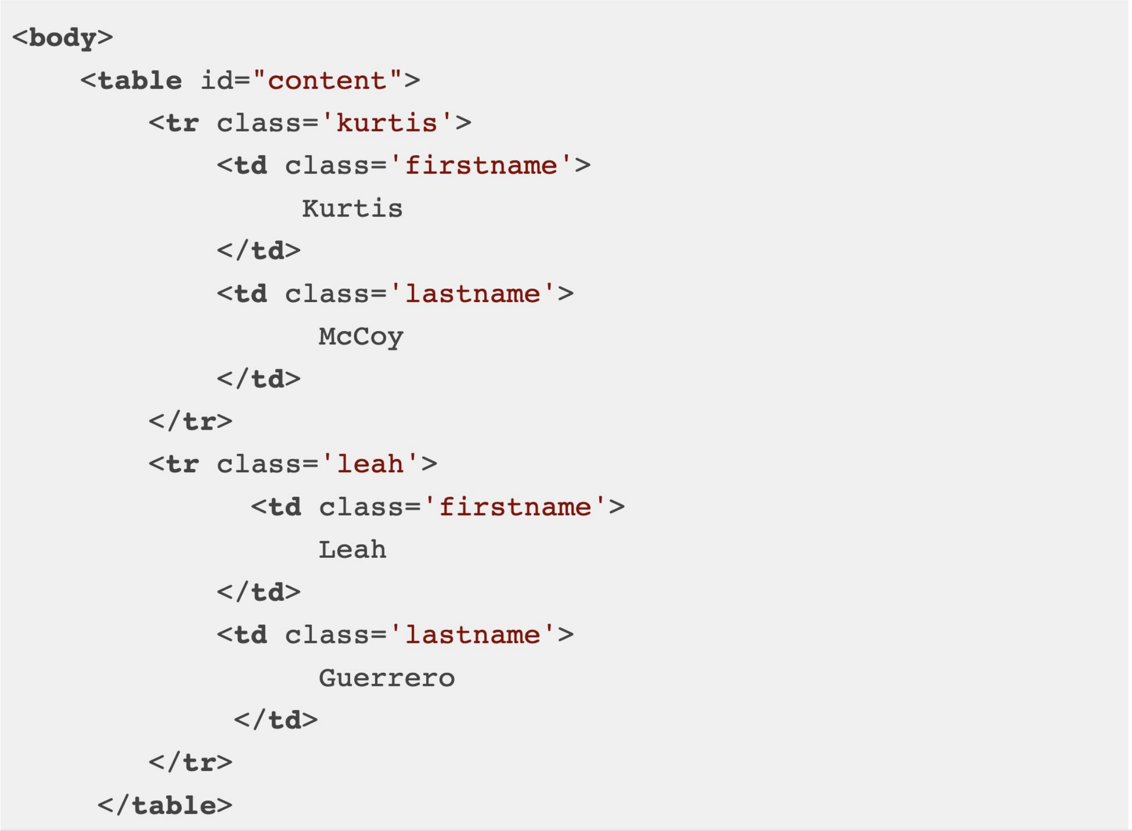
the XHTML
`<div id="wrapper">Main Content</div>`

the CSS
`#wrapper {width: 750px; margin: 0 auto;}`

use a pound sign when writing a rule for a id

CSS meets HTML

What does the following HTML render to?



```html
<body>
    <table id="content">
        <tr class='kurtis'>
            <td class='firstname'>
                 Kurtis
            </td>
            <td class='lastname'>
                  McCoy
            </td>
        </tr>
        <tr class='leah'>
              <td class='firstname'>
                  Leah
            </td>
            <td class='lastname'>
                  Guerrero
             </td>
        </tr>
     </table>
```

Relevance to Webscrapping

- It's necessary to cover HTML & CSS because we'll be using their tags to parse scrapped content.
- In the next section we'll be using the BeautifulSoup library to scrape web data.
- As previously mentioned, that's the easy part.
- Now that we know how tags work and what they're used for, we'll have an easier time sifting through the pile of HTML data