Web Scraping

Steve V. Iannaccone
Taryn Heilman

(modified by Jon Courtney)

galvanize

- Motivation
- The Internet and the WWW
- HTML and CSS
- Beautiful Soup
- Web APIs

Learning Objectives

galvanize

- Differentiate Internet vs. World Wide Web (www)
- Understand the basics of the client-server relationship in a web framework
- Explain the basic concepts of HTML and CSS
- Know how to use Python to pull content from a web page
- Know how to use Python to pull content using an API

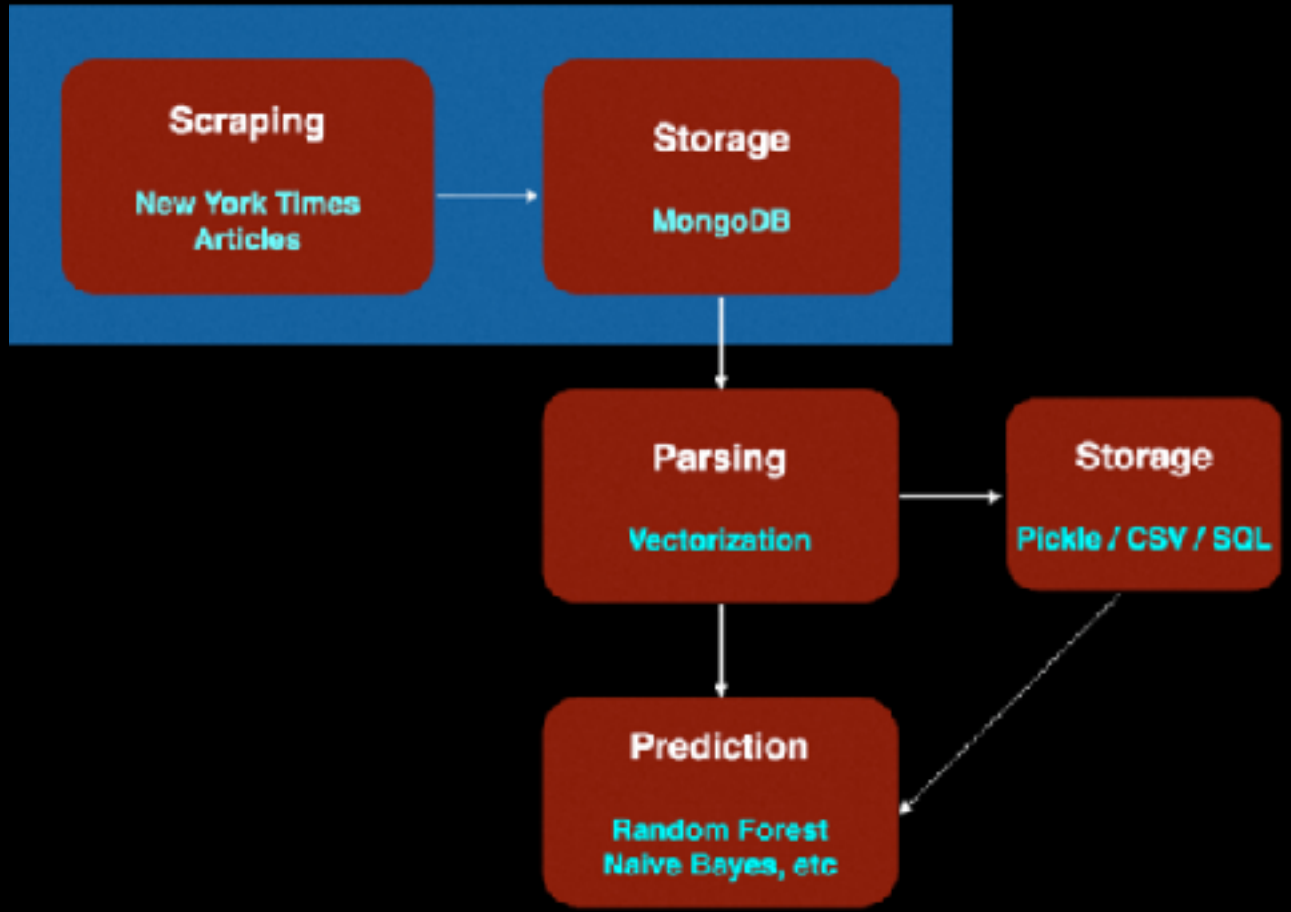
Motivation: Why do we want to do web scraping?

- Any time that you want data from the web and it doesn't have a clickable link, you will have to pull down that data via the command line (e.g., using curl) or via a program (e.g, using Python)
- The web is an enormous database of text/image/video training data*

* Be aware of any Terms of Service when using a website.
Some websites really don't like scrapers… https://techcrunch.com/2016/08/15/linkedin-sues-scrapers/

Typical Web Scraping + Machine Learning Pipeline

galvanize



WWW vs Internet

galvanize

- The ***world wide web*** is a global collection of interconnected hypermedia documents hosted on **web servers**
- The ***Internet*** is the global network that connects them (using TCP/IP)
- Think of web servers as islands existing all over the globe, while the Internet provides bridges connecting those islands

Less helpful:

> *"…the Internet is not something you just dump something on. It's not a big truck. It's a series of tubes."*
>
> — Senator Ted Stevens, opposing net neutrality June 28, 2006

URLs

galvanize

- **URL - U**niform **R**esource **L**ocator. Used to specify the location of a document within the world-wide web.
- Each url has a few different parts…

URLs

galvanize

- **Protocol -** specifies the means for communicating with the web server, typically `http` or `https`. Often gets automatically filled in in your web browser, but this is usually not the case when web scraping.
  See http://www.realifewebdesigns.com/web-resources/web-protocols.html for examples.
- **Host -** points to the name of the web server you want to communicate with. A host name is associated with a specific IP address.
- **Port -** holds additional information used to connect with the host. (Think: host is the city name, port is street address)
- **Path -** Indicates where on the server the file you are requesting lives

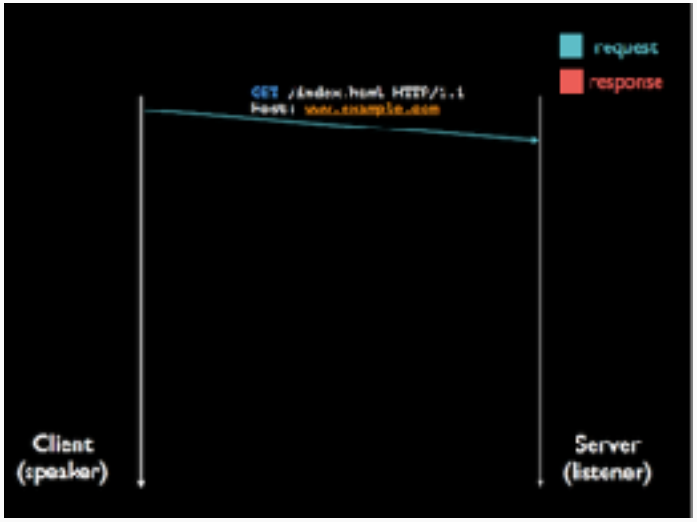
galvanize

Client-Server Relationships

- At any point in time, any person or computer connected to the internet can be either a server or a client
- The **client** is the requesting party (requesting some info, like a webpage)
- The **server** is the party providing that information and responding to requests from a client

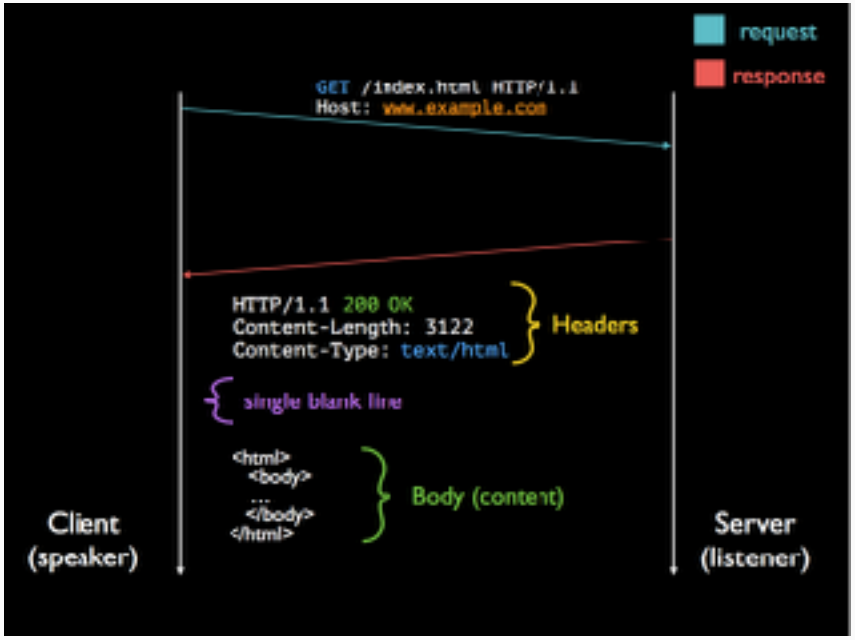
Client-Server Relationships

- If we visit www.example.com in our browser, then we are the client and www.example.com is the server.
- The interaction starts with the client issuing a GET request to the server, indicating that it would like some specific piece of information



Client-Server Relationships

- Once the server gets this request, it will send back a response with the information requested in the body (and a header with a status code in it)



galvanize

Http Status Codes

- In general…
- 2xx successful (usually 200)
- 3xx redirect - ultimately successful
- 4xx client side error aka the user's fault (common: 404 - you are looking for a file that doesn't live where you think it does, or wrong permissions)
- 5xx server side error aka the service's fault

For more info, see http://www.restapitutorial.com/httpstatuscodes.html

galvanize

HTML: Hypertext Markup Language

- The majority of web pages are formatted using Hypertext Markup Language (HTML)
  - Transmitted via Hypertext Transport Protocol (HTTP) over TCP/IP
- HTML uses tags, where each tag describes different document elements

Basic HTML Tags

galvanize

Tag	Description
`<h1> - <h6>`	Heading
`<p>`	Paragraph
`<i>`	Italic
`<b>`	Bold
`<a>`	Anchor (links)
`<ul> & <li>`	Unordered List & List Item
`<blockquote>`	Blockquote
`<img>`	Image
`<div>`	Division

HTML and CSS

- Permits clean separation of content from presentation style
- Radical site redesigns are possible by modifying just the style and not rewriting any of the content

Dave Shea's CSS Zen Garden

galvanize

http://www.csszengarden.com

```
<body id="css-zen-garden">
<div class="page-wrapper">

    <section class="intro" id="zen-intro">
        <header role="banner">
            <h1>CSS Zen Garden</h1>
            <h2>The Beauty of <abbr title="Cascading Style Sheets">CSS</abbr> Design</h2>
        </header>

        <div class="summary" id="zen-summary" role="article">
            <p>A demonstration of what can be accomplished through <abbr title="Cascading Style Sheets">CSS</
            <p>Download the example <a href="/examples/index" title="This page's source HTML code, not to be
        </div>

        <div class="preamble" id="zen-preamble" role="article">
            <h3>The Road to Enlightenment</h3>
            <p>Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible <a
            <p>We must clear the mind of the past. Web enlightenment has been achieved thanks to the tireless
            <p>The CSS Zen Garden invites you to relax and meditate on the important lessons of the masters. T
        </div>
    </section>
```

Dave Shea's CSS Zen Garden

Dave Shea's CSS Zen Garden

galvanize

Dave Shea's CSS Zen Garden

galvanize

galvanize

CSS Classes, IDs and Selectors

- Web developers may define CSS *classes* (starting with .) to identify HTML elements for styling
    - The same CSS class can be assigned to multiple page elements
    - Applying a new style to that class in the CSS changes the appearance of all elements using that class
- CSS IDs (starting with #) identify a single, unique element within a page.
- CSS *selectors* are patterns used to select HTML element(s) (including those matching CSS classes and IDs) for formatting.

galvanize

HTML & CSS Breakout

Go to:

https://www.w3schools.com/cssref/trysel.asp

Explore the various types of CSS selectors.

HTML and Web Scraping

- All those HTML tags, style classes and ids make great hooks for finding exactly the content you want in a web page.
- In practice, when web scraping, we will issue a get request for an entire page's html, and then pull specific elements either by their HTML tags or CSS selectors.

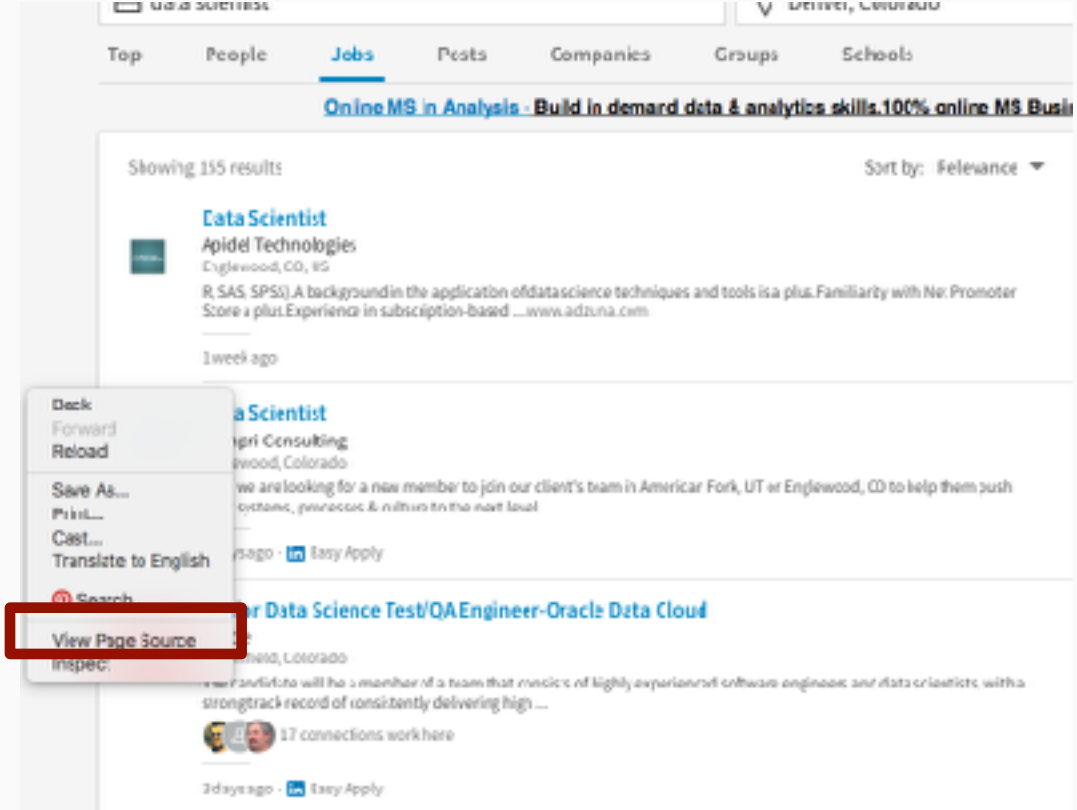
```
<body id="css-zen-garden">
<div class="page-wrapper">

    <section class="intro" id="zen-intro">
        <header role="banner">
            <h1>CSS Zen Garden</h1>
            <h2>The Beauty of <abbr title="Cascading Style Sheets">CSS</abbr> Design</h2>
        </header>

        <div class="summary" id="zen-summary" role="article">
            <p>A demonstration of what can be accomplished through <abbr title="Cascading Style Sheets">C
```

Viewing Page Source

- To see all of the HTML on a web page, right click and use ‘View Page Source’

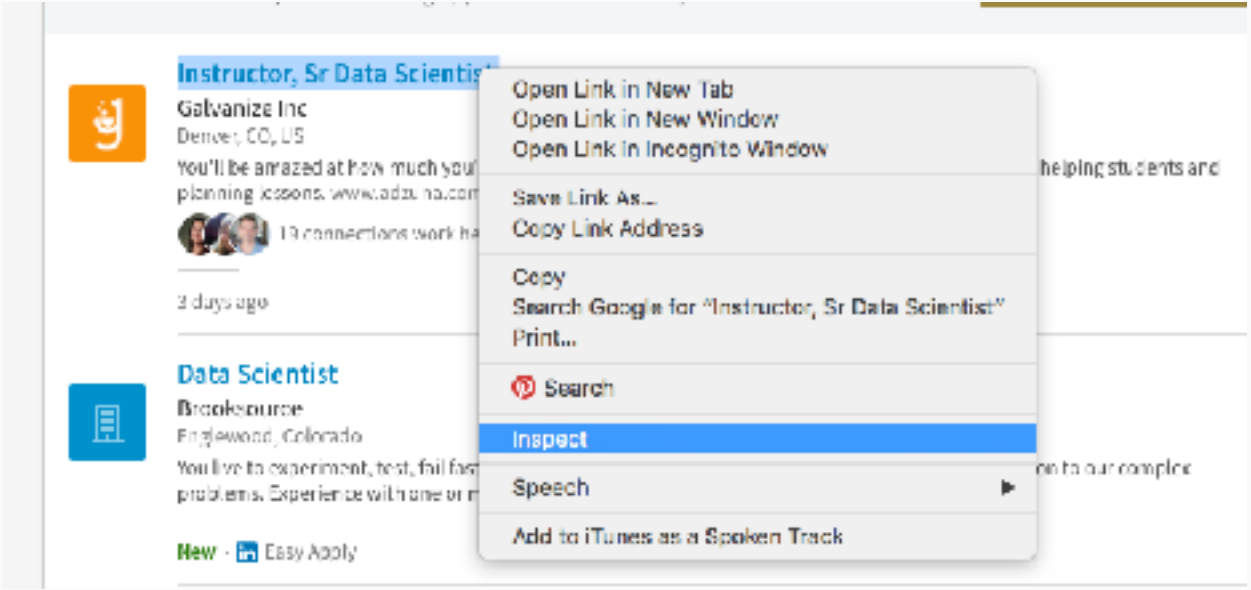


Which yields this mess…

galvanize

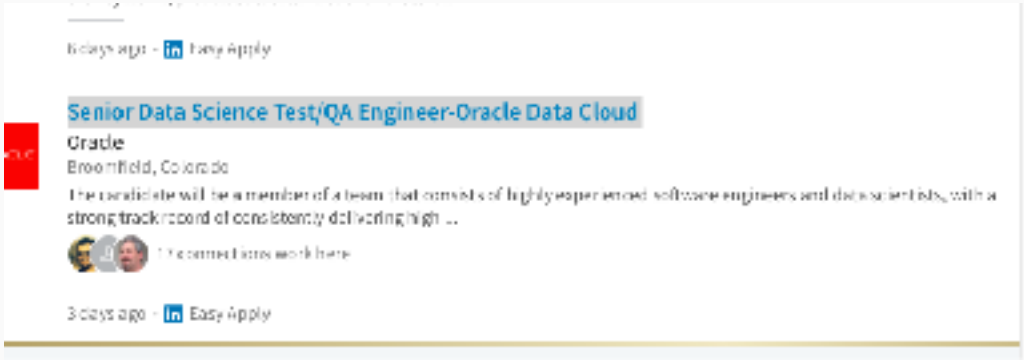
Inspecting a single element

- Better: right click and choose ‘Inspect’ (or ‘Inspect Element’) (Chrome, Firefox and Safari)



Resulting HTML from Inspect or Inspect Element

- Shows you the html that corresponds to the individual element you chose to look at



Scraping Individual Elements

galvanize

- Examine HTML corresponding to individual element we want to scrape, and then use either its HTML tag or CSS selectors to parse it out in Python
- Workflow:
    1) Find the page you want to scrape information from
    2) Find the element(s) that you want to grab
    3) Use inspect element to figure out what HTML tag or CSS selectors to use to grab the elements
    4) Use Python to fetch those elements

Beautiful Soup

“You didn't write that awful page. You're just trying to get some data out of it. Beautiful Soup is here to help. Since 2004, it's been saving programmers hours or days of work on quick-turnaround screen scraping projects.”

https://www.crummy.com/software/BeautifulSoup/

galvanize

Using Python and BeautifulSoup

- Use *requests* library to issue a 'GET' request on the url, then use *BeautifulSoup* to parse the html

```
import requests
from bs4 import BeautifulSoup

req = requests.get('www.example.com')
html = BeautifulSoup(req.content, 'html.parser')
```

galvanize

Using Python and BeautifulSoup

- Next, use methods on the html object returned from BeautifulSoup to get content from the page

```
html.find('a') # Returns the **first** 'a' tag.
html.find_all('a') # Returns **all** 'a' tags.

html.find('a').text # Returns the text of the
                    # **first** 'a' tag.
```

Beautiful Soup: Selecting Tags

galvanize

- **.select()** - This method allows us to select tags using CSS selectors
- **.find_all()** - This method allows us to select all tags matching certain parameters and returns them as a list. For example:
  - `soup.find_all('div')`: returns a list of all div tags in the soup object
  - `soup.find_all('div', attrs={'class' : 'content'})`: returns only the div tags that also have class = content.
- **.find()** - This method is the exact same as calling `soup.find_all(limit=1)`. Rather than returning a list, it only returns the first match that it finds.

Beautiful Soup: Navigating the HTML Tree

galvanize

Tags may contain strings and other tags. These elements are the tag's children. Beautiful Soup provides a lot of different attributes for navigating and iterating over a tag's children. For example:

```
head_tag = soup.head
title_tag = head_tag.contents[0]
for child in title_tag.children:
    print(child)
```

You can do .find() and .select() on a tag's contents too.

See https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-the-tree

Pandas and Web Scraping

Yes, pandas can read and parse HTML. So, if you only need data from tables from a single page this is probably the easiest approach.

```
tables = pd.read_html("http://www.website.com/page.html")
```

This returns a list of DataFrames where each DF is made from a table in the source page. But, depending on formatting quirks, this might be significantly worse than using *Beautiful Soup*.

Using Web APIs

Application Programming Interfaces (APIs)

Webmasters don't want you pinging their websites every 10ms to scrape some information.

Often times they build APIs to help you collect just the info you need (and so that you don't crush their servers).

It's win-win because you use less bandwidth on their server, and you get the data you need in a consistently formatted fashion.

galvanize

API Keys

You will often need to register to gain access to a site's API.

Sometimes to to track usage, often to limit the number of calls, and occasionally as a business model where you pay per call or by amount of transferred data.

These are essentially your login credentials, so **never** publish your keys! *Don't push up any script to GitHub* that has your keys in plain text.

galvanize

API Key Security

One option: Store your key as an environment variable on your local machine.

In your bash profile (`~/.bash-profile`, `~/.bashrc`, or `~/.profile`) include the following:

```
export API_KEY="my api key"
```

Using API Keys

Now we can use the variable API_KEY in our Python scripts using the `os` package without fear of publishing our private key when we push our repos.

```
import os

my_key = os.environ['API_KEY']
```

APIs: They're All Different

You'll have to dig into the docs to figure out how to use the API you need. But, if you want an intro on the broad strokes of how to interact with a web API in Python, this blog does a good job:

https://www.dataquest.io/blog/python-api-tutorial/

Beautiful Soup, so rich and green,
Waiting in a hot tureen!
Who for such dainties would not stoop?
Soup of the evening, beautiful Soup!
Soup of the evening, beautiful Soup!
Beau—ootiful Soo—oop!
Beau—ootiful Soo—oop!
Soo—oop of the e—e—evening,
Beautiful, beautiful Soup!

'Beautiful Soup! Who cares for fish,
Game, or any other dish?
Who would not give all else for two
Pennyworth only of beautiful Soup?
Pennyworth only of beautiful Soup?
Beau—ootiful Soo—oop!
Beau—ootiful Soo—oop!
Soo—oop of the e—e—evening,
Beautiful, beauti—FUL SOUP!


galvanize