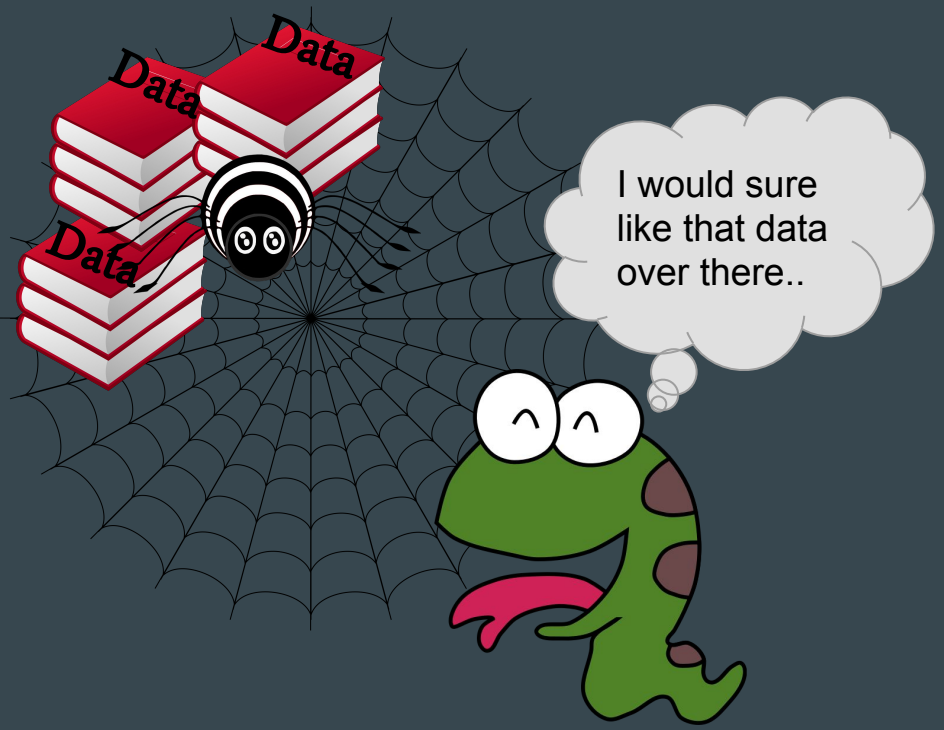
WEB SCRAPING

Intro to HTML, BeautifulSoup, and requests

What does it mean to “web-scrape”?

In simple terms..

Web scraping is the act of programmatically searching the internet for information you may need. Python is pretty good at this!



But why?

Because not all data is pre-processed for your use. There is a TON of information available to those that are willing to go fetch it.

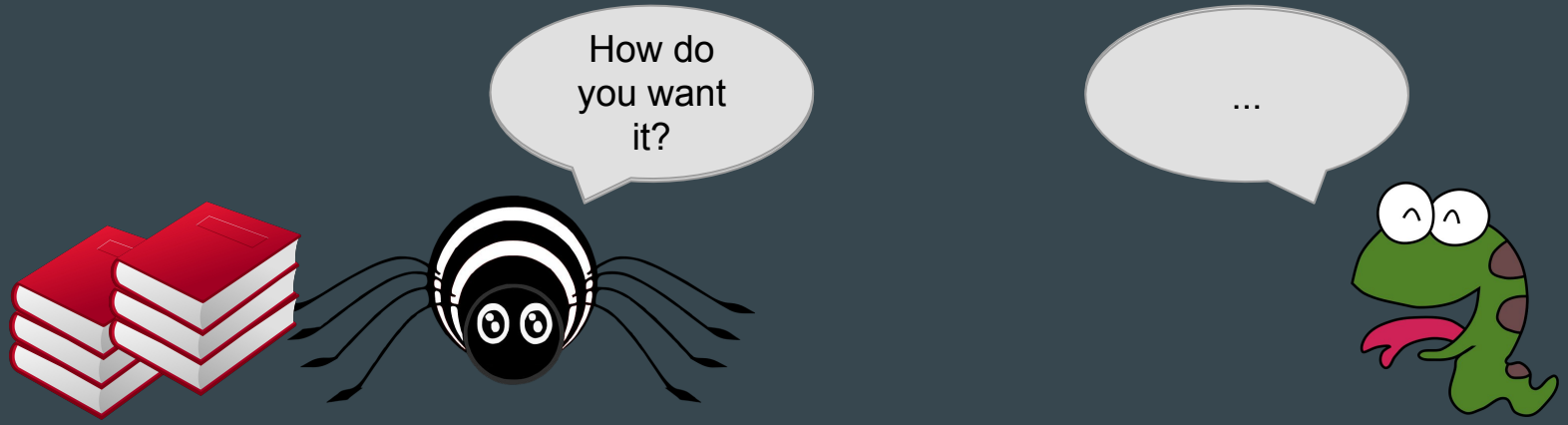
Un-processed data

Processed data

Snek

GREAT - How can I start?

Fortunately for you, accessing website data is pretty straightforward. You just need to “request” it.



Politely Requesting the Data With RESTful Commands

When you are requesting data from a website, you are interacting with an API (Application Programming Interface) defined by the website. An API is essentially a contract specifying how you should request access to data! There are different actions you can make to interact with the API.

- GET - SELECT
  - This should always ONLY query the API and fetch the data you want
- POST - INSERT
  - Typically adding records to a database. Also used when ***submitting forms***!
- PUT - UPDATE
  - Updates an existing record
- DELETE - REMOVE
  - Removes a record (or many) from a database

import requests

There is not an easy way to “request” data from a website in python by default. However, there is a package (requests) that makes this process really simple.

GET	→	requests.get(URL)
POST	→	requests.post(URL, data={...})
PUT	→	requests.put(URL, data={...})
DELETE	→	requests.delete(URL)

Notes On Status Codes

How can we know if our request went through successfully?

Check the status! Sometimes requests fail.

1. Status codes in the 200 range usually mean success. Shoot for this.
2. Codes in the 400 range are usually authentication related, basic fail.
3. Codes in the 500 range mean something went terribly wrong (usually the website's fault), definite fail.

Check here for more info on what status codes mean:

https://httpstatuses.com/

Python Requests Example

In [1]:
```
import requests
```

In [2]:
```
GITHUB_USER_URL = 'https://github.com/octocat'

resp = requests.get(GITHUB_USER_URL)
print(f'Status code is: {resp.status_code}')
```

```
Status code is: 200
```

In [3]:
```
RUN_ROCK_N_ROLL_URL = 'https://www.runrocknroll.com/Events/Nashville/The-Races/Marathon/2019-Results?gender=&agegroup=&

# POST request to simulate submitting a form!!!
resp = requests.post(RUN_ROCK_N_ROLL_URL, data={})
print(f'Status code is: {resp.status_code}')
```

```
Status code is: 200
```

What next?

So.. Now you have a valid response. Can we use it right away?

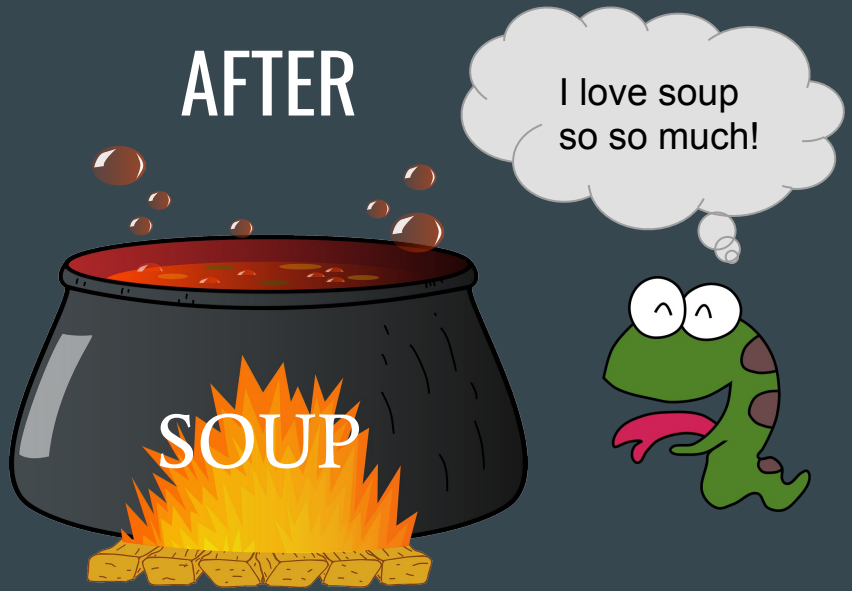
Not really. The response can be very cryptic and hard to decipher! Luckily, there are tools to help us parse this.

BeautifulSoup

beautifulsoup4 is a Python package used to traverse and manipulate an HTML tree. It makes elements within the tree easily accessible and query-able.

BEFORE

AFTER



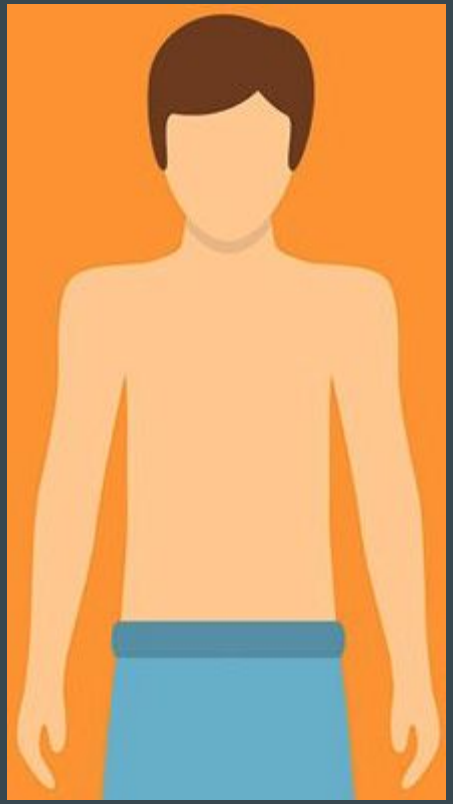
What Makes Up a Website?

What Makes Up a Website?

First off, the HTML itself! HTML makes up the building blocks of a website, and can be considered the “skeleton” of the website.

What Makes Up a Website?

Skipping a section.. The CSS! This is what we see whenever we view a website. This makes up the colors, fonts, basic animations, and nice layout.



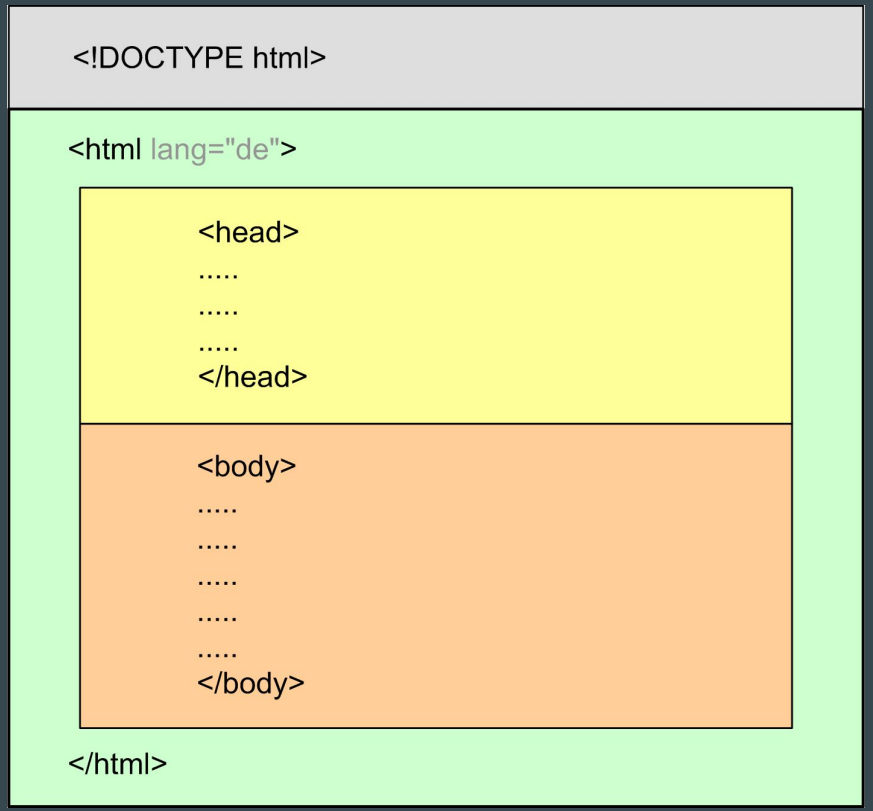
What Makes Up a Website?

And.. Javascript! This does a lot of heavy lifting behind the scenes. Passes data around, communicates to APIs, performs cool animations and charts.. Etc.

We primarily focus on the HTML

What's HTML?

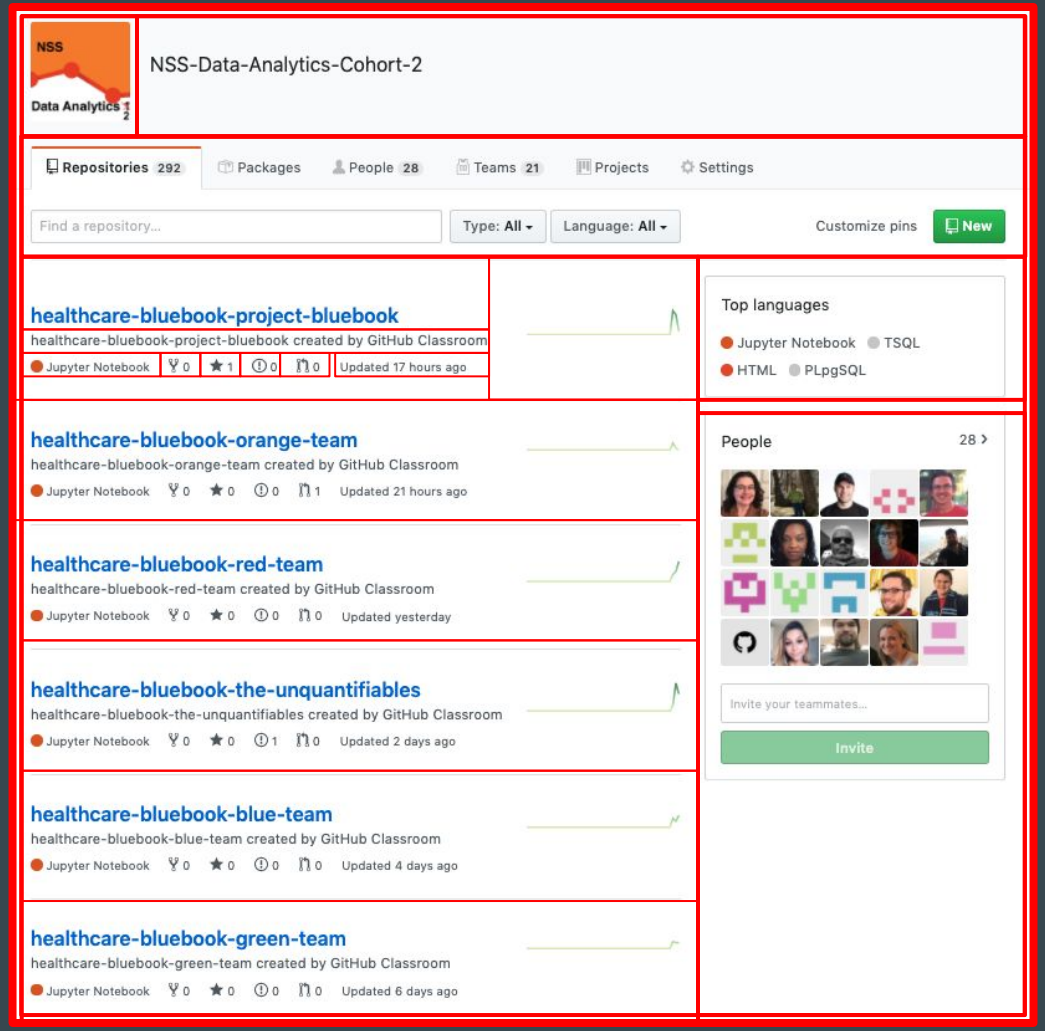
Stands for Hypertext Markup Language. HTML is the skeleton of all websites! If you want to build a website, or want to query against a website, you must understand the basics of HTML.



Website Structure

In the end, HTML is just a bunch of containers within containers. Some containers have special names, or represent different things..

However, it's still just a container.



HTML Containers

Great. So now we know that HTML consists of a bunch of containers.. But what do the containers look like?

<element>This is a basic element!</element>

<parent><child>This is a nested element!</child></parent>

Common Containers

-
-
- <form></form>
- <table></table>
  - <tr></tr> - Table row
  - <th></th> - Table header value
  - <td></td> - Table row value

Container Attributes

Usually, developers will tag a container with an attribute to help organize / separate it from other containers. You can set an attribute or combination of attributes to help identify a specific container.

There are two primary ways to separate and identify similar containers: id and class.

- ID - <div id=unique-div'>...
  - The id should be considered a unique identifier for an element, meaning there should only be one PER PAGE
- CLASS - ...
  - Classes can be shared between elements. This is a way for developers to specify a relationship or common behaviour between them. There might be one, might be many. Assume there are many!

Soup

OK! Now we have a decent understanding of what HTML is. So how can we use python to query it?

With BeautifulSoup

Making the Soup

It's about as simple as making the request itself! Import the BeautifulSoup object, then feed it the text from the request response. Make sure to specify the features.. Then you can start to use it.

```
In [1]: import requests
        from bs4 import BeautifulSoup

In [2]: GITHUB_USER_URL = 'https://github.com/octocat'

        resp = requests.get(GITHUB_USER_URL)
        print(f'Status code is: {resp.status_code}')

        Status code is: 200

In [3]: soup = BeautifulSoup(resp.text, features='html.parser')
        soup.find('title')

Out[3]: <title>octocat (The Octocat) · GitHub</title>
```

Drinking the Soup

There are 2 primary methods to help you traverse your HTML soup!

- soup.find()
- soup.findAll()

These 2 methods will return either a new “soup” object representing the element you selected, or a list of them! From these.. You can find other elements you’re interested in, or just extract the information you want.

soup.find*()

This method says.. Give me the first element you find for a given constraint, if any!

- soup.find('table')
  - Find the first "table" element
- soup.find('div')
  - Find the first "div" element
- soup.find('span', id='foo')
  - Find the first "span" element that has an "id" equal to "foo"
- soup.find('form', attrs={'class': 'submission-form'}, recursive=False)
  - Find the first "form" element with the "class" equal to "submission-form", but only look at the DIRECT children (don't look to children recursively).

You use soup.findAll() in the same way. It just returns a list vs a single element.

Text and Attributes

Once you have the element you're interested in, you can access properties of that element such as the text or hyperlinks.

- soupElement.get_text()
  - This will give you all of the inner text for a given object. Note that this will can also pull text from child elements.
- soupElement[<attribute>]
  - Use this pattern to get at the attributes you are interested in (such as links)
  - Link = soupElement['href']
  - Id = soupElement['id']

Digging Deeper..

- Requests:
  - https://requests.readthedocs.io/en/master/user/quickstart/
- Beautifulsoup4:
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- HTML Basics:
  - https://www.w3schools.com/html/html_basic.asp