

# Web Scraping Workshop

Bryant Park (DE '26) and Eric Zhang (DE '24)

Adapted from Tushar Khan (Insights '22)

# What is Web Scraping?

Web crawling is the process of systematically browsing the World Wide Web, typically for the purpose of Web indexing.

Web scraping is the process of automatically mining data or collecting information from the World Wide Web. It might require some crawling to access information hidden behind links on a page.

Web scraping a web page involves **downloading** the page (which a browser does when you view it) and then **extracting** the content from it.

# Why Web Scrape?

## CDS:

- Web sites generally have the most up-to-date data
- Allows more flexibility in creating custom data sets
- APIs don't exist for everything

## Industry:

- Market Price Monitoring
- Competitor Price Monitoring
- Product Trend Monitoring
- Market Trend Analysis
- Consumer Sentiment Monitoring
- News and Content Monitoring
- Real-time Analytics

# Slight Cons

Often requires maintenance

- If site is updated you may have to update your scraper as well
  - Example: reddit

Lots of sites have scraping protection

- Easy workaround but lowers speed

# Web Scraping in Python



# Getting Started in Python

The following libraries are the de-facto standard when it comes to web scraping in Python. You will need to install them:

- [requests](#) - making HTTP requests in Python.
  - `$ pip install requests`
- [beautifulsoup4](#) - pulling data out of HTML and XML files.
  - `$ pip install beautifulsoup4`
  - `$ pip install lxml`

# HTTP in a Nutshell

HTTP (HyperText Transfer Protocol) is the standard protocol used to structure the exchange of resources over the web.

Resources on the web are typically hosted on a **server**. A client (e.g. web browser) can perform a specific action on a resource by sending an HTTP **request**. There are a number of actions that a client can perform on a resource. For example, to read a resource it sends a GET request, to send a resource it sends a POST request, etc.

When a server receives a request from a client, it sends back information in the form of an HTTP **response**. The response contains not only the requested information, but additional metadata like a status code as well.

# Requests Library - Sending Requests

The `requests` library allows you to **send HTTP requests** extremely easily with quality-of-life features like keep-alive & connection pooling, sessions with cookie persistence, and more.

- GET requests
  - `r = requests.get('https://www.google.com')`
  - `r = requests.get('https://www.amazon.com/s', params={'k': 'shoes'})`
- POST requests
  - `r = requests.post('https://example.com/', data = {'key': 'value'})`
- Also supports PUT, DELETE, HEAD, and OPTIONS requests.



# Requests Library - Adding Headers

HTTP **headers** are typically sent along with an HTTP request or a response to pass additional information between the client and the server.

Our plain request doesn't contain any headers, so sites can easily figure out that it is being sent by a bot and not by a potential consumer. Thus, the server might be configured to send a response that doesn't actually contain any data.

We can get around this by sending packets that specify a **user agent** in the header to fool the server into thinking that the request was sent by a web browser:

```
r = requests.get('https://www.amazon.com', headers={'User-Agent': 'Mozilla/5.0'})
```

# Requests Library - Response Objects

The request methods all return a Response object, which contains all the information about the response sent by the server.

- Response Code

- `print(r.status_code)`  
`>>> 200`

- URL

- `print(r.url)`  
`>>> https://www.amazon.com/s?k=shoes`

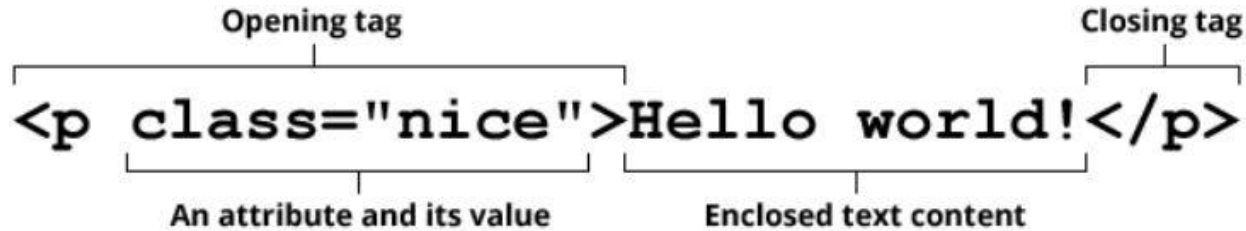
- HTML content

- `print(r.text)`  
`>>> <!DOCTYPE html> <!--[if lt IE 7]> <html lang="en-us" class="a-no- ...`

# HTML In a Nutshell

HTML (HyperText Markup Language) is a descriptive language that specifies web page structure.

An HTML document is structured by nested **elements**, which are surrounded by matching opening and closing **tags**. Tags can be extended with **attributes**, which provide additional information affecting how the browser interprets the element.



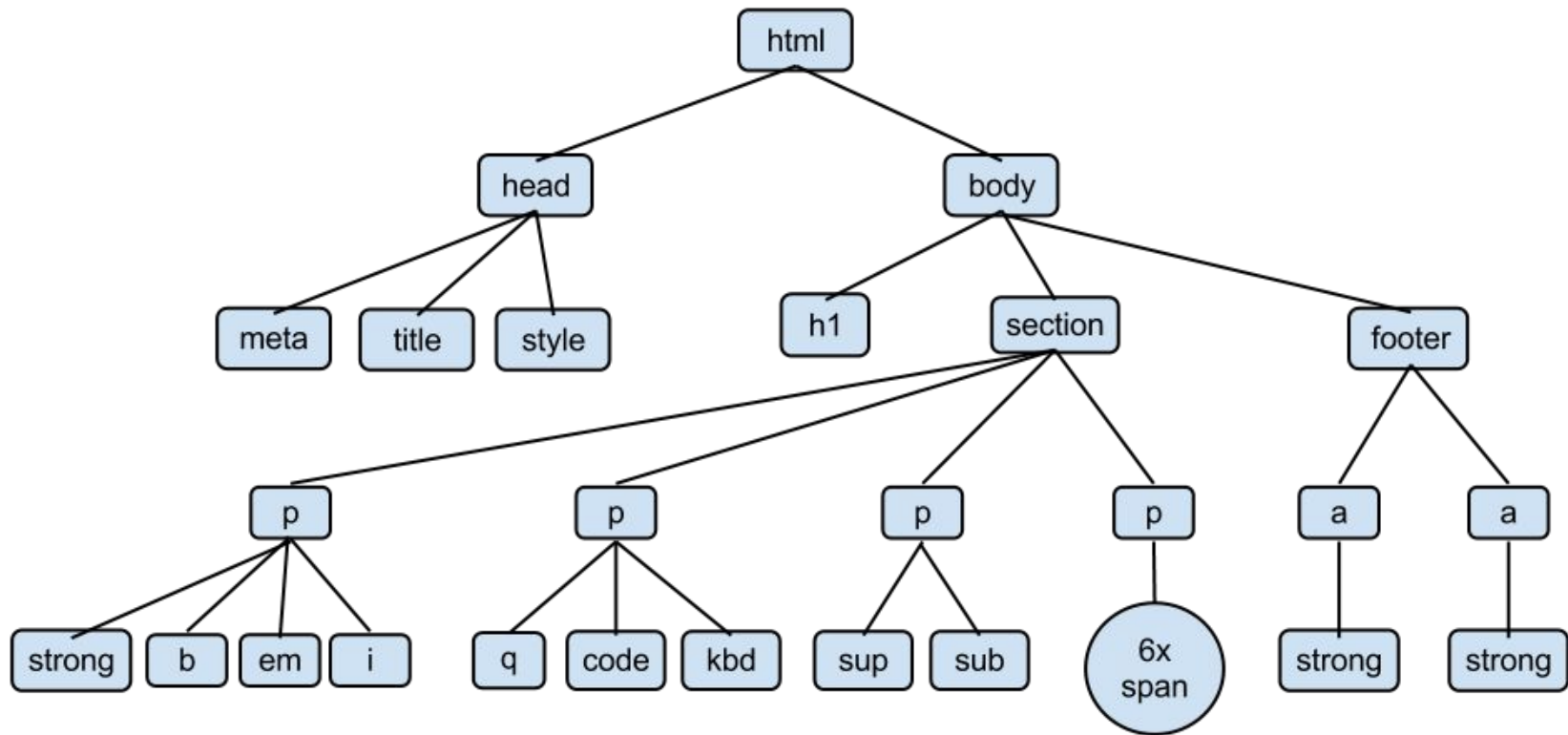
# BeautifulSoup Library - HTML Parsing

The `BeautifulSoup` library allows you to **extract data from HTML** (or XML) content. It does this by constructing a parse tree from the nested elements of the document and providing a plethora of ways to query objects in the tree.

The parse tree is constructed when we create a `BeautifulSoup` object. The first argument contains the HTML and the second argument specifies the parser:

- `soup = BeautifulSoup(r.text, 'html.parser')`
- `soup = BeautifulSoup(r.text, 'lxml')`

Note: I suggest using the `lxml` parser because it is much faster.



Example HTML parse tree

# BeautifulSoup Library - Searching for Tags

BeautifulSoup makes it very easy to extract specific tags from the parse tree using the `find()` and `find_all()` methods. These methods respectively find the first tag or a list of all tags in the HTML parse tree that match the query you provide.

- Search by tag type
  - `tags = soup.find('body') # finds the first <body> tag`
- Search by id or CSS class attributes
  - `tags = soup.find_all(id='bar', class_='icon')`
- Search using a filter function
  - `tags = soup.find(lambda t: t.has_attr('href') and not t.has_attr('img'))`
- Search the nested structure of the document
  - `nested_tags = soup.find('div', id='preview').find_all('a')`

# BeautifulSoup Library - Tag Objects

A Tag object corresponds to a tag in the original HTML document. The `find()` and `find_all()` methods return a Tag object or a list of Tag objects respectively.

The attributes of the tag can be retrieved using dictionary syntax. The text enclosed by the tag can be retrieved using the `.text` or `.string` attributes.

```
soup = BeautifulSoup('<div><p align="left">Ho ho ho</p><p>Yo</p></div>', 'lxml')
tag = soup.find('div').find('p') # returns the first <p> tag in the first <div> tag
print(tag['align']) # prints the value associated with the 'align' key

>>> left

print(tag.text) # prints text enclosed by tag

>>> Ho ho ho
```

# Inspecting Web Pages

We've just learned how to extract data from HTML content, but how do we know which elements we actually want to extract?

The answer is by **inspection** of the actual page.

- Windows: Ctrl + Shift + C
- Mac: ⌘ + Shift + C

It is helpful to first write the returned page to a file to inspect the HTML:

```
with open('page.html', mode='wb') as f:  
    f.write(r.content)
```



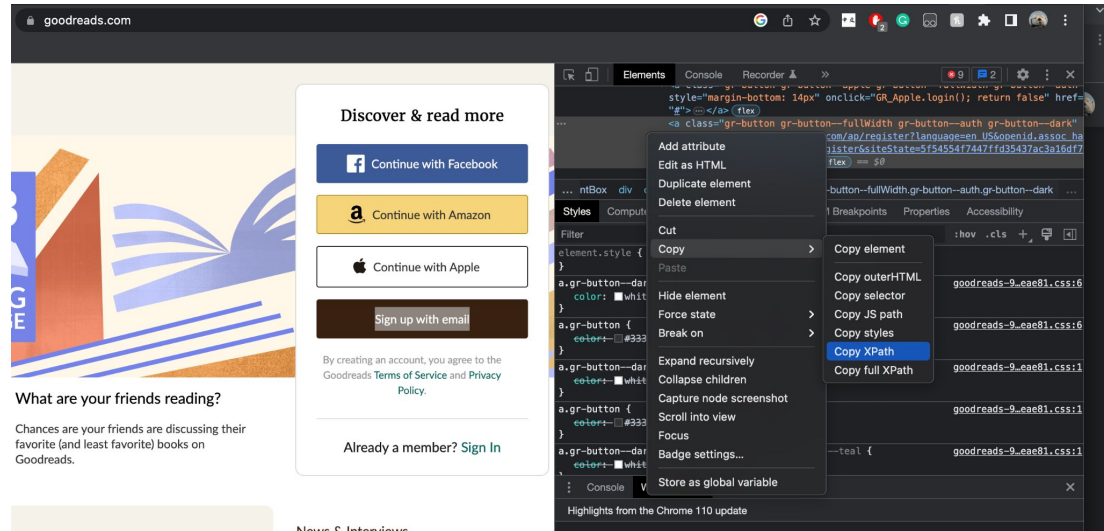
# Are there ways to find stuff without using tags/digging through html to find correct var names?

Yes, just not supported by default beautifulsoup

- Selenium supports the use of XPath

cmd + option + i -> right click ->  
inspect -> copy -> XPath

Very simple, almost always breaks  
everytime site is changed



# Demo



# Demo Project - Scraping Reddit Comments

We're going to be scraping r/Cornell.

Specifically, we will be building a dataset of the most popular comments of the top 5 posts on r/Cornell. This dataset probably doesn't exist already, especially because the top posts and comments become outdated all the time. So we turn to web scraping.

# Old Reddit vs. New Reddit





**Welcome to Reddit.**  
Where a community about your favorite things is waiting for you.

**BECOME A REDDITOR** and subscribe to one of thousands of communities.

**Chance Me! and Prospective Student Q&A** (self: Cornell)  
submitted 1 month ago by pw111111 **ECE '23** [M] - announcement  
48 comments share save hide report

**POLL: Please let us know your thoughts on moderating posts containing sensitive content (i.e., suicide, suicidal ideation)** (self: Cornell)  
submitted 10 days ago by with\_alice\_please **AM A&S '23** [M] - announcement  
9 comments share save hide report

**To whoever got stabbed** (self: Cornell)  
submitted 9 hours ago by annatw2  
29 comments share save hide report

**Hey r/parents, a new collection has bloomed, and it's as sweet as can be. Introducing Magnolia, a floral print featuring elegant white magnolias on Khaki and Midnight. Shop pjs, sleep bags, footies and more from Kyte Baby™ at the link below.**  
(kytebaby.com)  
promoted by kyte\_baby  
promoted save hide report

**Can I use a land acknowledgment as the first 183 words of my 500-word essay?** (self: Cornell)  
submitted 3 hours ago by conccornall  
3 comments share save hide report

search

username password

☐ remember me reset password login

**Submit a new post**

**Submit a new text post**

**reddit premium**

Get an ad-free experience, benefits, and directly support creators.

**Get Reddit Premium**

**Cornell**

join 47,627 readers  
539 users here now

Far above Cayuga's waters,  
with its waves of blue,  
Stands our noble Alma Mater,  
glorious to view.  
Lift the chorus, speed it onward  
loud her praises tell;  
Hail to thee our Alma Mater!

```
<!doctype html><html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en"><head><title>Cornell</title><meta name="keywords"></meta></head></html>

<p></p>with its waves of blue,</p>

<p></p>Stands our noble Alma Mater, </p>

<p></p>glorious to view.</p>

<p></p>Lift the chorus, speed it onward, </p>

<p></p>loud her praises tell;</p>

<p></p>Hail to thee our Alma Mater! </p>

<p></p>Hail, all hail, Cornell!</p>

<hr/>

<p></p>Questions? Check out the <a href="https://www.reddit.com/r/Cornell/wiki/index">r/Cornell wiki</a>!</p>

<hr/>

<p></p><strong>Rules</strong></p>

<ol>
<li></li>Posts should be somehow relevant to Cornell or the Cornell community</li>
<li></li>NO SPAM/ADS - If you are advertising for a Cornell-related event, please check with the mods first. </li>
<li></li>Be nice - antagonistic and/or offensive comments or posts will be removed at the discretion of the mods. You are free to appeal.
<li></li>Search the subreddit to see if your question has already been asked recently before posting it.</li>
<li></li>Do not post personal/identifying information about yourself or anyone else (this does not apply in the case of questions about Cornell)</li>
<li></li><strong>ALL ADMISSIONS-RELATED POSTS GO IN THE MEGATHREAD</strong> Any posts outside of the megathread will be removed.</li>
<li></li>Posts and comments will be removed (and users can be banned) at the discretion of the moderators. </li>
</ol>

<hr/>

<hr/>

<p></p>Important/Useful Links:</p>

<p></p><a href="https://www.reddit.com/r/Cornell/wiki/index">r/Cornell</a> FAQs</a></p>

<p></p><a href="http://www.cornell.edu/">University Website</a></p>

<p></p><a href="http://now.cornell.edu/cv/">Cornell Now</a></p>

<p></p><a href="http://classes.cornell.edu/">Courses of Study</a></p>

<p></p><a href="http://www.cornell.edu/academics/calendar/">Academic Calendar</a></p>

<p></p><a href="http://cornellsun.com/">The Cornell Daily Sun</a></p>

<p></p><a href="http://www.metaesra.com/images/ctown_map.JPG">Collegietown Map</a></p>

<hr/>
```

Old Reddit



r/Cornell

Join

Current Student FAO

...

 PINNED BY MODERATORS

Posted by u/pw11111  ECE '23 1 month ago 

## Chance Me! and Prospective Students

48 Comments [Share](#) [Save](#) ...

Posted by u/with\_aloe\_please **A&S '23** 10 days ago

POLL: Please let us know your thoughts on moderating posts containing sensitive content (i.e., suicide, suicidal ideation)

9 Comments [Share](#) [Save](#) ...

Posted by u/annatwd 10 hours ago

To whoever got stabbed

that is really scary i hope you're okay

29 Comments [Share](#) [Save](#) ...

Create an account to follow your favorite communities and start taking part in conversations.

[Join Reddit](#)

```

<!DOCTYPE html>
<html lang="en-US" class="theme-beta theme-light">
<head>
<script>
var __SUPPORTS_TIMING_API = typeof performance === 'object' && !performance.mark && !performance.measure && !performance.getEntriesByType;
function __perfMark(name) { __SUPPORTS_TIMING_API && performance.mark(name); };
var __firstPostLoaded = false;
function __markFirstPostVisible() {
  if ( __firstPostLoaded ) { return; }
  __firstPostLoaded = true;
  __perfMark("first_post_title_image_loaded");
}
var __firstCommentLoaded = false;
function __markFirstCommentVisible() {
  if ( __firstCommentLoaded ) { return; }
  __firstCommentLoaded = true;
  __perfMark("first_comment_loaded");
}
</script>
<script> __perfMark('head_tag_start');</script>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="referrer" content="origin-when-cross-origin" />
<style>
/* http://meyerweb.com/eric/tools/css/reset/
v2.0 | 20110126
License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, button, cite, code,
del, dfn, em, img, input, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}

/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
  display: block;
}

```

# 1. Downloading the Top Posts Page

The first thing we need to do if we want to scrape the top comments of the top posts on Reddit is to **download the page containing the top posts on r/Cornell**. This page we want in particular is <https://old.reddit.com/r/all/Cornell/?t=all>.

We can request the page like so. Note the parameters and the user agent.

```
base_url = 'https://old.reddit.com/r/Cornell/top/'
parameters = {'t': 'all', 'limit': 5} # this is to sort by top of all time
(check API)
user_agent = {'User-Agent': 'Mozilla/5.0'}

r = requests.get(base_url, params=parameters, headers=user_agent)
```

## 2. Inspecting the Top Posts Page

Next, we need to collect a **list of links to the comments section** of every post on the page. To find the specific HTML tags those links correspond to, we can visually inspect the HTML of the page we were returned.

```
with open('page.html', mode='wb') as f:  
    f.write(r.content)
```





### 3. Extracting Links to Comment Sections

Now that we've found a search parameter for the HTML elements we are trying to extract, we can easily collect those elements using BeautifulSoup.

```
soup = BeautifulSoup(r.text, 'lxml')  
link_tags = soup.find_all('a', class_='bylink comments may-blank')
```

The returned list contains tag elements, not the actual links. To extract these links, we need to extract the value of the “href” attribute for each tag.

```
links = [tag['href'] for tag in link_tags]
```

## 4. Feature Selection

For each comment section link in our list, we want to extract every single comment with some associated data about that comment. Deciding what data might or might not be useful for our dataset is called **feature selection** and is a crucial step in producing good results.

For each comment, we will collect the following associated data:

- Original post score
- Original post link
- Comment string
- Commenter name
- Comment score

## 5. Extracting Original Post Score

this post was submitted on 05 Dec 2016

**283,489 points** (97% upvoted)

shortlink: <https://redd.it/5gn8ru>

For each link to a post in our list of links, we need to extract the score of the original post. Like before, we can request the page of one link and write it to a file to visually inspect the HTML.

```
r = requests.get(links[0], params={'sort': 'top'}, headers=user_agent)
with open('link.html', mode='wb') as f:
    f.write(r.content)
```

*Find some set of attributes (possibly nested) that uniquely correspond with the original post's score.*

```
score_str = soup.find('div', class_='linkinfo').find('span', class_='number').text
op_score = int(score_str.replace(',', '')) # because `Tag.text` is a string
```

## 6. Inspecting the Comment Section

Extracting the post score was relatively easy. Extracting each comment together with its metadata is a little more difficult and requires some more creativity.

We can parse down specifically to the comment section of the HTML like so:

```
comment_section = soup.find('div', class_='commentarea')
```

At this point it seems like visually inspecting the HTML could be helpful again.

```
with open('comment_section.html', mode='w') as f:  
    f.write(str(comment_section)) # Note we use mode 'w' for writing a str
```

*Find some set of attributes (i.e. tag, id, class, or some combination of the three) that uniquely correspond with the container around each comment.*

## 7. Extracting Comment Containers

Every comment is enclosed in a “div” element with the class attribute “entry unvoted”. However, there are other “div” elements with that same class attribute.

The “div” elements that corresponded to comments exclusively contain a child “span” element with the class attribute “score unvoted”, since that element actually corresponds to the comment score. The “div” element that contains this “span” element also contains the other data we want to extract from each comment, so we want to find and store all of these “div” elements specifically.

We can use the information above to search for the tags using a filter function:

```
comment_filter = lambda t: t.find('span', class_='score unvoted') is not None
comments = soup.find_all(comment_filter, attrs={'class': 'entry unvoted'})
```

## 8. Extracting Comment Data

Now that we have a list of every comment container in the comment section, we can simply extract the relevant data for each comment by finding the tags that they correspond to. We can do this for each comment by iterating over the list.

*Find some set of attributes (i.e. tag, id, class, or some combination of the three) that uniquely correspond with the commenter name, comment text, and comment score in each comment container.*

```
for container in comments:
    commenter_tag = container.find(class_='author')
    commenter = commenter_tag.text if commenter_tag is not None else '[deleted]'
    comment = container.find('div', class_='usertext-body').text
    score = container.find(class_='score unvoted')['title']
```

## 9. Saving the Scraped Data

Once we've scraped all the relevant data, we need to be sure to save it.

One way to save our data is to store it in a `pandas` `Dataframe` object. Then, we can simply write that `Dataframe` object to a CSV, JSON, etc. after collecting the data.

```
# Define dataframe
```

```
features = ['post score', 'post link', 'score', 'commenter', 'comment']
```

```
data = pd.DataFrame(columns=features)
```

```
# Data collection code
```

```
...
```

```
# Write dataframe to csv
```

```
data.to_csv('reddit_comments.csv', index=False)
```

# Resulting Dataset

We just finished building a simple, custom dataset of the top comments on the top posts of r/Cornell! We can inspect our dataset using pandas Dataframe methods.

```
print(data.sort_values(by='score', ascending=False).head(5))
```

```
(venv) bryant@Bryants-MBP cds-webscraping-workshop % python demo.py
```

```
Scraping data from post 1 of 5
```

```
Scraping data from post 2 of 5
```

```
Scraping data from post 3 of 5
```

```
Scraping data from post 4 of 5
```

```
Scraping data from post 5 of 5
```

	post	score	comment	score	post link	commenter	comment
9		1134		378	<a href="https://old.reddit.com/r/Cornell/comments/qx9b...">https://old.reddit.com/r/Cornell/comments/qx9b...</a>	The_Clink123	Cornell: "Wear a mask and test weekly to stop ...
71		1138		321	<a href="https://old.reddit.com/r/Cornell/comments/qov7...">https://old.reddit.com/r/Cornell/comments/qov7...</a>	Wonderful-Long-5079	I can't wait to read Martha's email tonight th...
75		1138		294	<a href="https://old.reddit.com/r/Cornell/comments/qov7...">https://old.reddit.com/r/Cornell/comments/qov7...</a>	jiucaihezi	Your honor this is not what I meant when I sai...
11		1134		285	<a href="https://old.reddit.com/r/Cornell/comments/qx9b...">https://old.reddit.com/r/Cornell/comments/qx9b...</a>	Malvire	You can, and should, file/report to the univer...
76		1138		273	<a href="https://old.reddit.com/r/Cornell/comments/qov7...">https://old.reddit.com/r/Cornell/comments/qov7...</a>	cheedicken	Lol I love how they test the Cornell alert sys...

```
(venv) bryant@Bryants-MBP cds-webscraping-workshop %
```



# Ethics



# Denial of Service Attacks

As a human browsing the web, you can only send so many requests to a server at a time. But with a web crawler, you can easily execute thousands of requests in just a few seconds. If the server receives more requests than it can handle, it could become overloaded and **stop fulfilling legitimate requests** altogether. This is called a [Denial of Service](#) (DoS) attack.

You should be smart about how often you send a request. For example, you should only **send one request per page** and save it locally if you need to use it again. If you are sending many requests, **send them slowly** instead of all at once.

Some websites will even throttle or block your IP address altogether if they detect an unusually high amount of requests.

# Robots Exclusion Standard

The [robots exclusion standard](#) is a web standard used by websites to communicate to web crawlers which pages it can and cannot request. It is primarily used to manage crawler traffic by whitelisting and blacklisting certain parts of the site.

It can be found by simply appending “robots.txt” to the base URL of any website (e.g. <https://www.reddit.com/robots.txt>, <https://www.amazon.com/robots.txt>, etc.)

This is just a standard, meaning it is not explicitly enforced. But as a programmer utilizing a site’s resources at their expense, you should respect this standard.

If you really need to access a blacklisted part of a website, your web crawler should emulate a human by **throttling the rate of requests** to prevent getting blocked.