

# CDS Web Scraping Workshop

---

Tanish, Tyagi, Srivatsa Kundurthy



# OVERVIEW

# What is Web Scraping?

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

Web scraping: the process of automatically mining data or collecting information from the World Wide Web (typically a web page)

- might require some crawling to access information hidden behind links on a page
- involves **downloading** the web page (which a browser does when you view it) and then **extracting** the content from it

# Why Web Scrape?

## For INFO 1998:

- Web sites generally have the most up-to-date data → automate the retrieval of structured web data
- Allows more flexibility in creating custom data sets
- More powerful than using APIs

## Industry:

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

# WEB SCRAPING

with Python 

# Getting Started in Python

The following libraries are the de-facto standard when it comes to web scraping in Python. Let's install them right now:

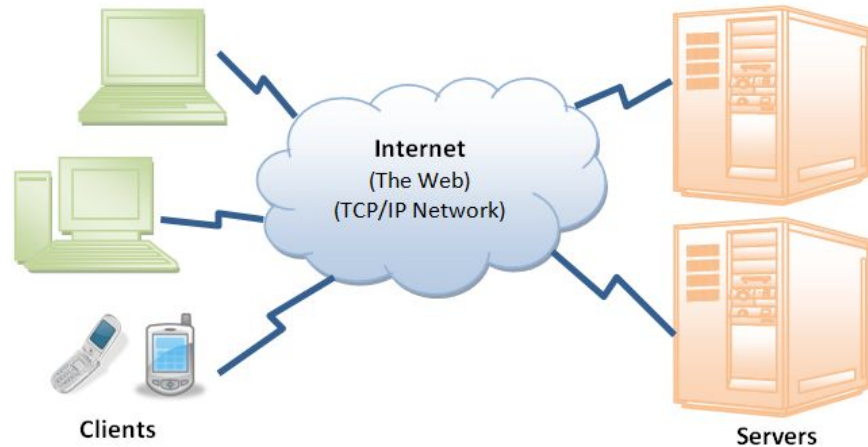
- requests - making HTTP requests in Python.
  - `$ pip install requests`
- beautifulsoup4 - pulling data out of HTML and XML files.
  - `$ pip install beautifulsoup4`
  - `$ pip install lxml`

(Note: you may need to use the command pip2/pip3 instead of pip depending on the version of Python you have)

# The Big, Beautiful Web

aka the Internet

- A massive distributed client server information system with many running applications
- How does the client and server communicate with each other?



# HTTP

The foundation of data communication for the Web



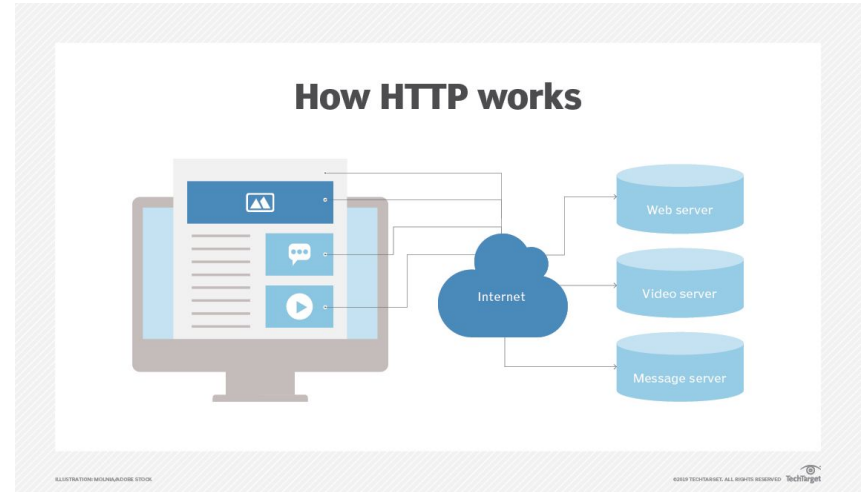
# HTTP in a Nutshell

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

To browse the web, we use a computer (e.g. laptop, phone, desktop) and open a web browser to type a Uniform Resource Locator (URL) or go to a link from an existing page.



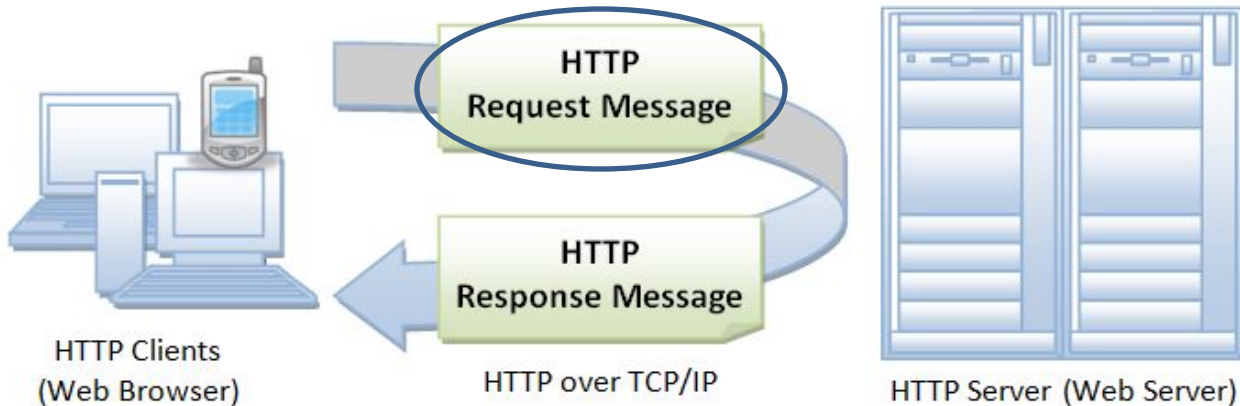
from Khan Academy: "Hypertext Transfer Protocol (HTTP)"



<https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol>

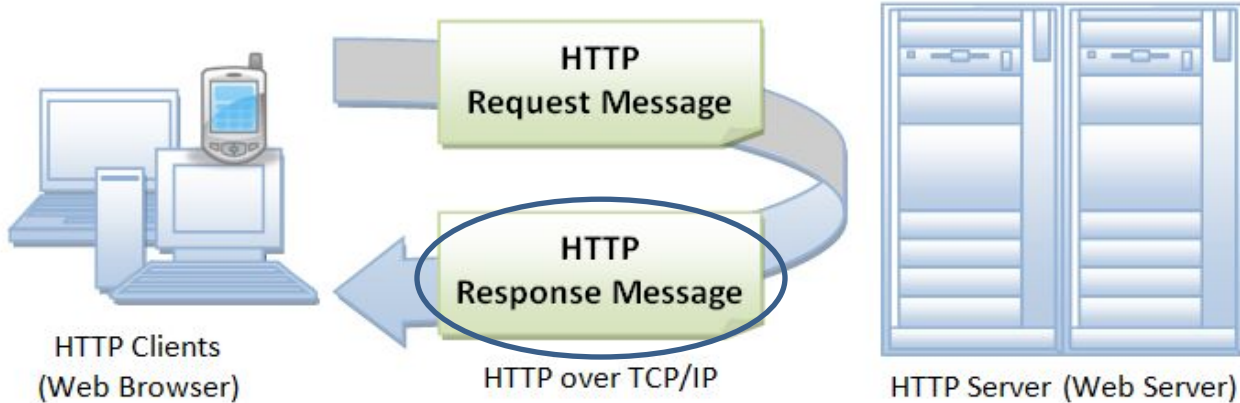
# HTTP Request

- 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.



# HTTP Response

- 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 also 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- ...`

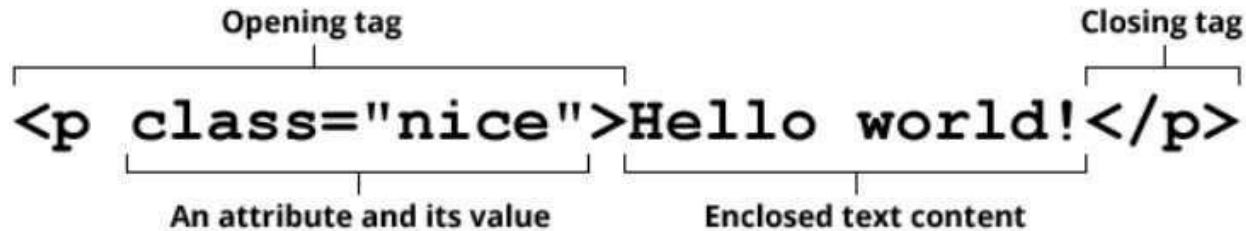
# HTML

and soup that is beautiful

# 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: the lxml parser is supposedly much faster.

# 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)
```

# INTERACTIVE DEMO

It's time to scrape

# Demo Project - Scraping Reddit Comments

We're going to be scraping Reddit, everybody's favorite potty pastime.

Specifically, we will be building a dataset of the most popular comments of the most popular posts of all-time! 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.



# 0. Setting Up Jupyter Notebook

First, let's set up our IDE: Jupyter Notebook.

- Make a new directory: `$ mkdir directoryname`
- Remove a directory: `$ rm -R directoryname`
- Change directory: `$ cd directoryname`
- Navigate one level up in directory: `$ cd ..`
- Run Jupyter Notebook: `$ jupyter notebook`

In Jupyter Notebook, create a new notebook and import necessary libraries:

```
import requests as rq
from bs4 import BeautifulSoup as bs
import lxml as lxml
import pandas as pd
```

# 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 of all time**. The page we want specifically is <https://old.reddit.com/r/all/top/?sort=top&t=all>.

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

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

r = rq.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)
```

*Find some set of attributes (i.e. tag, id, class, or some combination of the three) that uniquely correspond with the comments section link of every post.*

## 2. Inspecting the Top Posts Page

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

links from: **all time** ▾

1 ↑  
439k ↓



I've found a few funny memories during lockdown. This is from my 1st tour in 89, backstage in Vegas. [OC](#)

(i.redd.it)

submitted 2 years ago by

[a.bylink.comments.may-blank](#) 99.13 x 12

19118 comments

share save hide report

» See gilded comments and submissions

Submit a new link

Submit a new text post



reddit premium

Get an ad-free experience with special benefits, and directly support Reddit.

Get Reddit Premium

```
Elements Console Sources Network Performance >> 2 14 7
▼<div class="entry unvoted">
  ▼<div class="top-matter">
    ▶<p class="title">...</p>
    <div class="expando-button collapsed hide-when-pinned video"></div>
    ▶<p class="tagline">...</p>
  ..
  ▼<ul class="flat-list buttons"> == $0
    ▼<li class="first">
      <a href="https://old.reddit.com/r/pics/comments/haucpf/ive_found_a_few_funny_memories_during_lockdown/" data-event-action="comments" class="bylink comments may-blank" rel="nofollow">19118 comments</a>
    </li>
    ▶<li class="share">...</li>
    ▶<li class="link-save-button save-button login-required">...</li>
    ▶<li>...</li>
    ▶<li class="report-button login-required">...</li>
  </ul>
  <div class="reportform report-t3_haucpf"></div>
</div>
```

### 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 = bs(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 17 Jun 2020

**438,827 points** (99% upvoted)

shortlink: <https://redd.it/haucpf>

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 = rq.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.*

```
soup = bs(r.text, 'lxml')
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', encoding="utf-8") 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 extract the relevant data for each comment by finding the tags that they correspond to. Iterate over the list and store each comment in a pandas DataFrame object.

```
features = ['post score', 'comment score', 'link', 'commenter', 'comment']
data = pd.DataFrame(columns=features)
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.strip().replace('\n',
    ' ')
    score = int(container.find(class_='score unvoted')['title'])
    data = data.append(dict(zip(features, [op_score, score, links[0], commenter,
    comment])), ignore_index=True)
```



## 9. Saving the Scraped Data

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

As we saw just now, 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; remember to iterate over all the links on the page!
...

# 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 Reddit! We can inspect our dataset using pandas Dataframe methods.

```
data.sort_values(by='comment score', ascending=False).head(8)
```

	post score	comment score	post link	commenter	comment
<b>2</b>	438828	133960	<a href="https://old.reddit.com/r/pics/comments/haucpf/...">https://old.reddit.com/r/pics/comments/haucpf/...</a>	theMalleableDuck	I think I might cry!!! It's actually you. I m...
<b>3</b>	438828	94588	<a href="https://old.reddit.com/r/pics/comments/haucpf/...">https://old.reddit.com/r/pics/comments/haucpf/...</a>	ReallyRickAstley	
<b>1460</b>	293165	45818	<a href="https://old.reddit.com/r/videos/comments/88ll0...">https://old.reddit.com/r/videos/comments/88ll0...</a>	ChromaticSideways	I really didn't think this video was going to ...
<b>1641</b>	313948	44117	<a href="https://old.reddit.com/r/aww/comments/90bu6w/h...">https://old.reddit.com/r/aww/comments/90bu6w/h...</a>	wyslan	Frogs drink through their skin, so you cooled ...
<b>1836</b>	294211	44030	<a href="https://old.reddit.com/r/wallstreetbets/commen...">https://old.reddit.com/r/wallstreetbets/commen...</a>	JoeyIsActive	IF HE'S STILL IN, I'M STILL IN.
<b>1</b>	438828	42477	<a href="https://old.reddit.com/r/pics/comments/haucpf/...">https://old.reddit.com/r/pics/comments/haucpf/...</a>	ReallyRickAstley	Yes!
<b>1871</b>	294211	39426	<a href="https://old.reddit.com/r/wallstreetbets/commen...">https://old.reddit.com/r/wallstreetbets/commen...</a>	Bearmay33	WE HOLD
<b>0</b>	438828	37165	<a href="https://old.reddit.com/r/pics/comments/haucpf/...">https://old.reddit.com/r/pics/comments/haucpf/...</a>	[deleted]	Wait, you're actually Rick Astley tho...?

# ETHICS

"With great power comes great responsibility"

# 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.

# TIPS

Learning web scraping beyond this presentation

# How to Google 101

Learning to Google isn't hard, but learning how to Google well is only slightly harder so why not just master it?

- Useful for debugging, finding documentation, figuring out what to learn and how to best learn it
- (Semi-blind) Debugging: copy-paste error messages BUT do not blindly follow online solutions without understanding the context
- Documentation: view it from the creator's perspective and figure out how things work using documentation
  - <https://requests.readthedocs.io/en/latest/>
- Ask the right questions that help you progress to a solution

# THANK YOU

Questions?