

# CDS Web Scraping Workshop

---

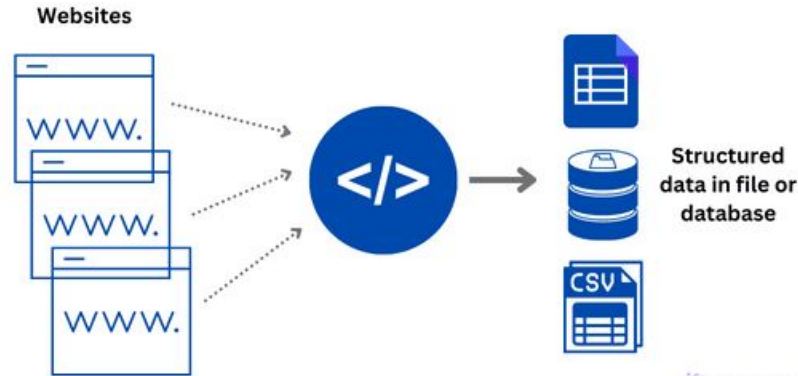
Tanvi Bhawe, Daniel Wang



# OVERVIEW

# What is Web Scraping?

1. **Downloading:** visiting a webpage and retrieving the HTML
  - May involve web crawling
2. **Extracting:** parsing the site to get the important information

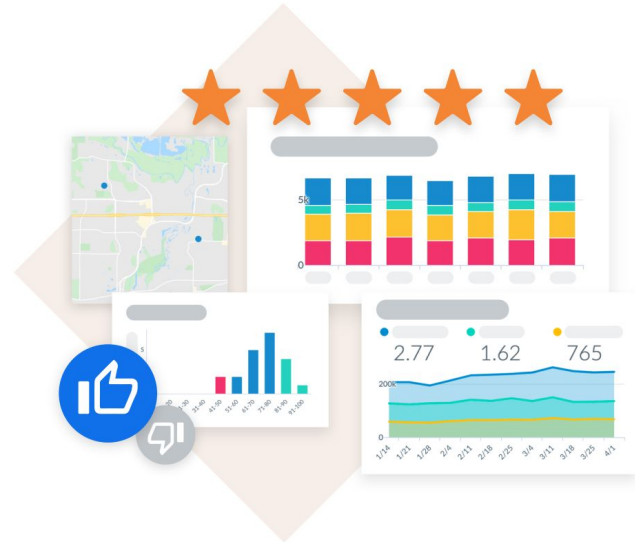


# Why Web Scrape?

- Web sites generally have the most up-to-date data
- More flexibility in creating custom data sets
- More powerful than using APIs

## Example uses:

- Competitor Price Comparison
- Marketing Lead Generation
- Sentiment Analysis
- Stock Market Monitoring



# WEB SCRAPING

with Python 

# Getting Started in Python

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

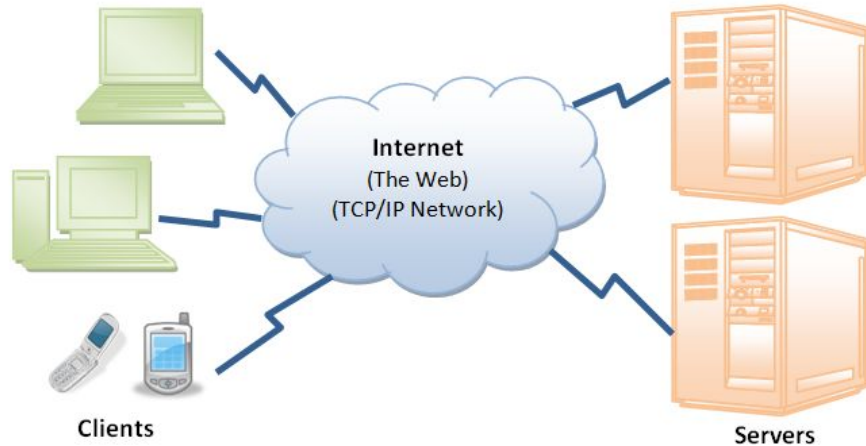
- requests - making HTTP requests in Python.
  - `$ pip install requests`
- beautifulsoup4 - pulling data out of HTML and XML files.
  - `$ pip install beautifulsoup4`
  - `$ pip install lxml`

(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

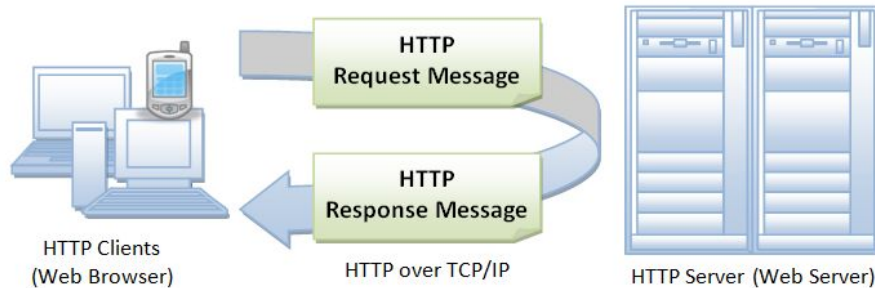
The standard protocol used to structure the exchange of resources over the web.

## 1. Request

- Client can perform an action on a resource by sending an HTTP **request**

## 2. Response

- Server sends back information in the form of an HTTP **response**



# Requests Library - Sending Requests

The requests library allows you to **send HTTP requests** extremely easily

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

# 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- ...`

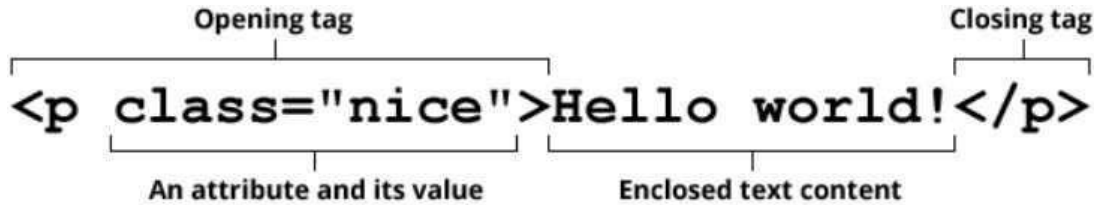
# HTML

and soup that is beautiful

# HTML In a Nutshell

HTML is a language that specifies web page structure.

Contains **tags** and **attributes** that can be used to identify relevant information while web scraping.



## HTML Page Structure

```
<!DOCTYPE html> ← Tells version of HTML
<html> ← HTML Root Element

<head> ← Used to contain page HTML metadata
  <title>Page Title</title> ← Title of HTML page
</head>

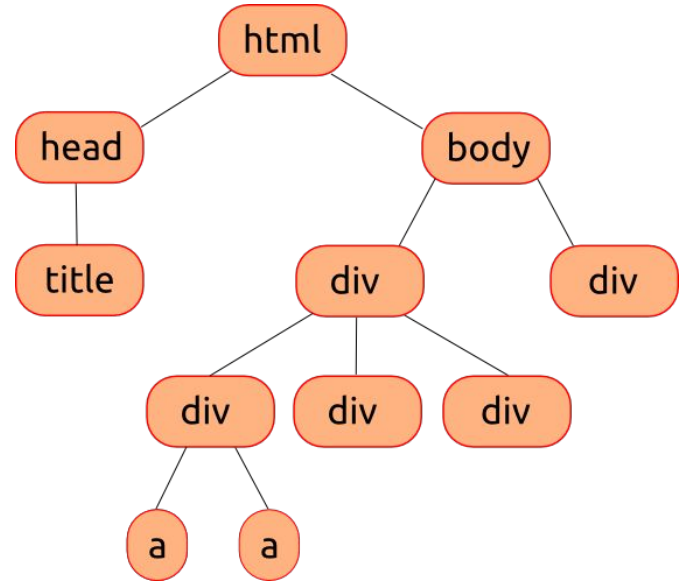
<body> ← Hold content of HTML
  <h2>Heading Content</h2> ← HTML heading tag
  <p>Paragraph Content</p> ← HTML paragraph tag
</body>

</html>
```

# BeautifulSoup Library - HTML Parsing

The BeautifulSoup library allows you to **parse data** from the HTML document by constructing a parse tree.

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



# BeautifulSoup Library - Searching for Tags

Extract specific tags from the parse tree:

- `find()`: finds the first tag
- `find_all()`: generates a list of all tags
- 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

The `find()` and `find_all()` methods return a Tag object or a list of Tag objects

- Tag objects correspond to a tag in the original HTML document.

Tag attributes can be retrieved using dictionary syntax.

```
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 **inspecting** the actual page. (inspect element)

- 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 Book to Scrape

Home / All products

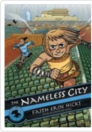
## All products

1000 results - showing 61 to 80.

Warning! This is a demo website for web scraping purposes. Prices and ratings here were randomly assigned and have no real meaning.

Books

- Travel
- Mystery
- Historical Fiction
- Sequential Art
- Classics
- Philosophy
- Romance
- Womens Fiction
- Fiction
- Childrens
- Religion
- Nonfiction
- Music
- Default
- Science Fiction
- Sports and Games
- Add a comment
- Fantasy
- New Adult
- Young Adult
- Science
- Poetry
- Paranormal
- Art
- Psychology
- Autobiography
- Biography



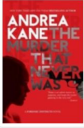
★★★★☆

The Nameless City (The ...

£38.16

✓ In stock

Add to basket



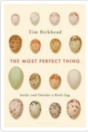
★★★★☆

The Murder That Never ...

£54.11

✓ In stock

Add to basket



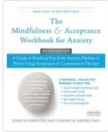
★★★★☆

The Most Perfect Thing: ...

£42.96

✓ In stock

Add to basket



★★★★☆

The Mindfulness and Acceptance ...

£23.89

✓ In stock

Add to basket

Task: Build a dataset that shows

- Book names
- price

# 0. Setting Up Google Colab

Import necessary libraries:

```
import requests  
from bs4 import BeautifulSoup  
import pandas as pd
```



# 1. Downloading the Books Page

We want to scrape all books in the catalogue from: <https://books.toscrape.com/>

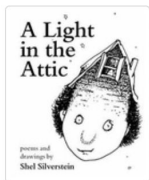
We can request the page like so:

```
base_url = 'https://books.toscrape.com/'
```

```
response = rq.get(base_url)
```

## 2. Inspecting the Books Page

We need to find the specific HTML tag, id, class, etc. each feature uniquely corresponds to.



A Light in the ...

£51.77

✓ In stock

Add to basket

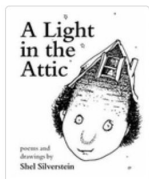
```
▼ <article class="product_pod">
  ▶ <div class="image_container"> ... </div>
  ▶ <p class="star-rating Three"> ... </p>
  ▼ <h3>
```

```
  <a href="catalogue/a-light-in-the-attic 1000/index.html" title="A Light in
    the Attic">A Light in the ...</a> == $0
</h3>
```



## 2. Inspecting the Books Page

Let's make that a little cleaner



A Light in the ...

£51.77

✓ In stock

Add to basket

```
<article class="product_pod">
  <h3>
    <a title="The Book Title" href="some-link.html">The Book Title</a>
  </h3>
  <div class="product_price">
    <p class="price_color">£51.77</p>
  </div>
</article>
```

### 3. Let's start off by seeing if we can scrape page 1

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.

```
▶ response = requests.get(base_url)
  soup = BeautifulSoup(response.content, "html.parser")
  books = soup.select(".product_pod")
```

We get the response from the base\_url and then select .product\_prod as that is the outer article that encompasses both the title and price.



### 3. Let's start off by seeing if we can scrape page 1

Now let's first get the list of books

Recall!

```
[35] for book in books:  
      name = book.h3.a["title"]  
      print(name)
```

```
▶ for book in books:  
   name = book.find("h3").find("a").get_text()  
   print(name)
```

```
<article class="product_pod">  
  <h3>  
    <a title="The Book Title" href="some-link.html">The Book Title</a>  
  </h3>  
  <div class="product_price">  
    <p class="price_color">£51.77</p>  
  </div>  
</article>
```



There are multiple ways to get the same values depending on how you parse the html

### 3. Let's start off by seeing if we can scrape page 1

Now let's first get the list of books

Recall!

```
▶ for book in books:  
    price = book.select(".price_color")[0].get_text()  
    print(price)
```

```
<article class="product_pod">  
  <h3>  
    <a title="The Book Title" href="some-link.html">The Book Title</a>  
  </h3>  
  <div class="product_price">  
    <p class="price_color">£51.77</p>  
  </div>  
</article>
```

We get the response from the base\_url and then select .product\_prod as that is the outer article that encompasses both the title and price.

### 3. Let's start off by seeing if we can scrape page 1

We can look at the print output

```
➦ A Light in the ...  
  Tipping the Velvet  
  Soumission  
  Sharp Objects  
  Sapiens: A Brief History ...  
  The Requiem Red  
  The Dirty Little Secrets ...  
  The Coming Woman: A ...  
  The Boys in the ...  
  The Black Maria  
  Starving Hearts (Triangular Trade ...  
  Shakespeare's Sonnets  
  Set Me Free  
  Scott Pilgrim's Precious Little ...  
  Rip it Up and ...  
  Our Band Could Be ...  
  Olio  
  Mesaerion: The Best Science ...  
  Libertarianism for Beginners  
  It's Only the Himalayas
```

```
➦ £51.77  
  £53.74  
  £50.10  
  £47.82  
  £54.23  
  £22.65  
  £33.34  
  £17.93  
  £22.60  
  £52.15  
  £13.99  
  £20.66  
  £17.46  
  £52.29  
  £35.02  
  £57.25  
  £23.88  
  £37.59  
  £51.33  
  £45.17
```

### 3. Extracting Data from HTML

Now we can get the same data from all the pages

```
[39] books = []
     page = 1

     while True:
         url = f"{base_url}catalogue/page-{page}.html"
         response = requests.get(url)

         if response.status_code != 200:
             break

         soup = BeautifulSoup(response.content, "html.parser")
         for book in soup.select(".product_pod"):
             title = book.h3.a["title"]
             price = book.select_one(".price_color").text
             books.append({"title": title, "price": price})

         page += 1

     df = pd.DataFrame(books)
```

## 4. 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
```

```
df = pd.DataFrame({  
    'title': titles,  
    'price' : prices,  
})
```

```
# Save it as a csv file
```

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

# Resulting Dataset

W00000000!!!!!!!

	<b>title</b>	<b>price</b>
<b>0</b>	A Light in the Attic	£51.77
<b>1</b>	Tipping the Velvet	£53.74
<b>2</b>	Soumission	£50.10
<b>3</b>	Sharp Objects	£47.82
<b>4</b>	Sapiens: A Brief History of Humankind	£54.23
...	...	...
<b>995</b>	Alice in Wonderland (Alice's Adventures in Won...	£55.53
<b>996</b>	Ajin: Demi-Human, Volume 1 (Ajin: Demi-Human #1)	£57.06
<b>997</b>	A Spy's Devotion (The Regency Spies of London #1)	£16.97
<b>998</b>	1st to Die (Women's Murder Club #1)	£53.98
<b>999</b>	1,000 Places to See Before You Die	£26.08

## Bonus: How would we get availability

```
▼ <div class="product_price">
  <p class="price_color">£51.77</p>
  ▼ <p class="instock availability"> == $0
    ▶ <i class="icon-ok"> ... </i>
      " In stock "
    </p>
    ▶ <form> ... </form>
  </div>
```

avail = book.find("p", class\_="instock availability").get\_text() or  
avail = book.select(".instock availability").get\_text()

# UN-ETHICS

"With great power comes great responsibility"



# Denial of Service Attacks

A human browser can only send so many requests to a server at a time, but a web crawler can easily execute thousands of requests in a few seconds.

The server could become overloaded and **stop fulfilling legitimate requests.**

- This is called a Denial of Service (DoS) attack.

You should be smart about how often you send a request:

- **send one request per page** and save it locally
- **send requests slowly**

# 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.

# THANK YOU

Questions?