Web Scraping Part 1

a **Data Science Drop-in** Tutorial

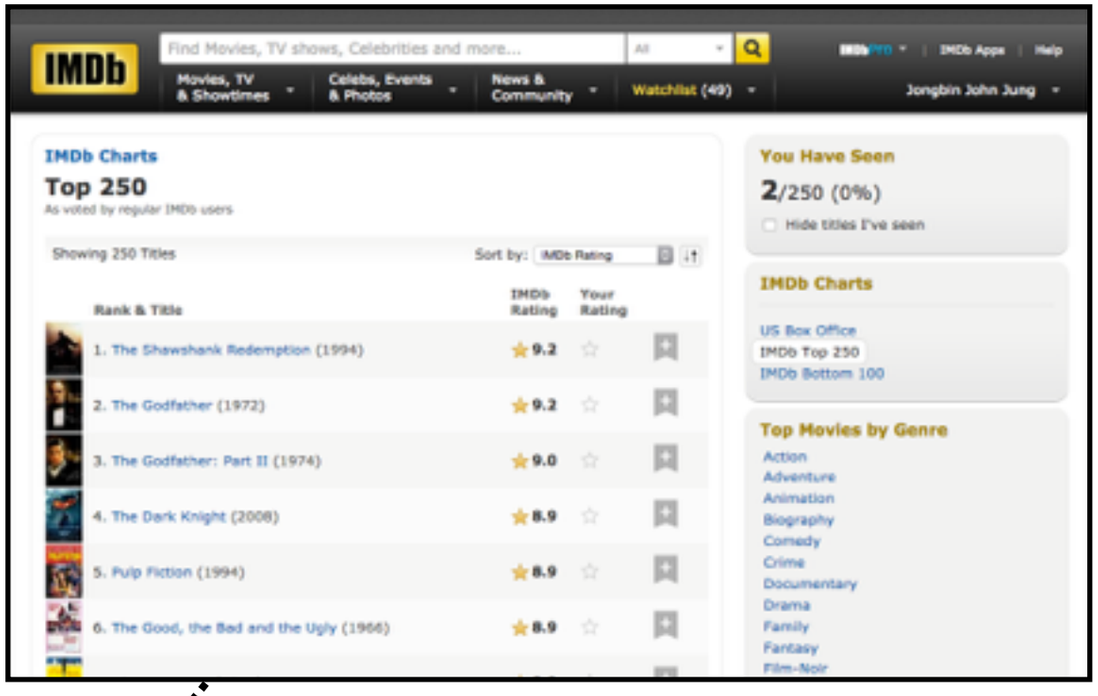
by **Jongbin Jung**
(jongbin@stanford.edu)

Before We Begin

- Slides / Code available at: https://5harad.com/drop-in/tutorials

- Git repo at: https://github.com/5harad/datascience

Web Scraping?

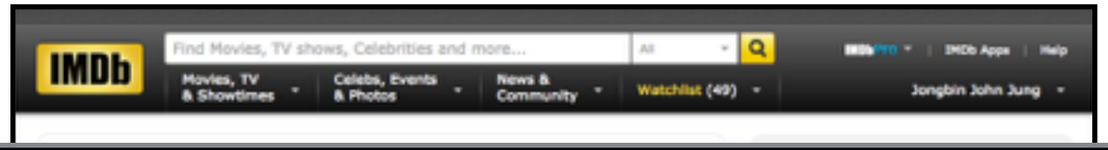


	movie_title	actor_name	character
1	The Shawshank Redemption	Tim Robbins	Andy Dufresne
2	The Shawshank Redemption	Morgan Freeman	Ellis Boyd 'Red' Redding
3	The Shawshank Redemption	Bob Gunton	Warden Norton
4	The Shawshank Redemption	William Sadler	Heywood
5	The Shawshank Redemption	Clancy Brown	Captain Hadley
6	The Shawshank Redemption	Gil Bellows	Tommy
7	The Shawshank Redemption	Mark Rolston	Bogs Diamond
8	The Shawshank Redemption	James Whitmore	Brooks Hatlen
9	The Shawshank Redemption	Jeffrey DeMunn	1946 D.A.
10	The Shawshank Redemption	Larry Brandenburg	Skeet
11	The Shawshank Redemption	Neil Giuntoli	Jigger
12	The Shawshank Redemption	Brian Libby	Floyd
13	The Shawshank Redemption	David Proval	Snooze
14	The Shawshank Redemption	Joseph Ragno	Ernie
15	The Shawshank Redemption	Jude Ciccolella	Guard Mert
16	The Godfather	Marlon Brando	Don Vito Corleone
17	The Godfather	Al Pacino	Michael Corleone
18	The Godfather	James Caan	Sonny Corleone
19	The Godfather	Richard S. Castellano	Clemenza (as Richard Castellano)
20	The Godfather	Robert Duvall	Tom Hagen
21	The Godfather	Sterling Hayden	Capt. McCluskey
22	The Godfather	John Marley	Jack Woltz
23	The Godfather	Richard Conte	Barzini

Get from here…

… to **here**

Web Scraping?



	movie_title	actor_name	character
1	The Shawshank Redemption	Tim Robbins	Andy Dufresne
2	The Shawshank Redemption	Morgan Freeman	Ellis Boyd 'Red' Redding
3	The Shawshank Redemption	Bob Gunton	Warden Norton
4	The Shawshank Redemption	William Sadler	Heywood

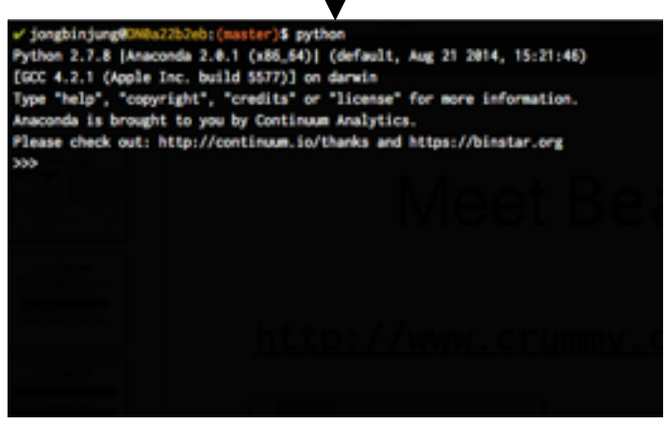
Today's **GOAL**

Collect the **cast overview** (actor and character played) for each of the **Top 10 movies** of **IMDb Charts' Top 250** (http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

`get` the Web to `python`

This can be done many ways.
But we use `requests` (for now)

from source (`html`)
to `python`




```
import requests


web_page =
requests.get('address')
web_page.content
```

[python]

Let's try this with the **IMDb Top 250** web page:

http://www.imdb.com/chart/top?ref_=nv_ch_250_4

Meet BeautifulSoup

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

from source (html)
to python

BeautifulSoup
goes **here**

	movie_title	actor_name	character
1	The Shawshank Redemption	Tim Robbins	Andy Dufresne
2	The Shawshank Redemption	Morgan Freeman	Ellis Boyd 'Red' Redding
3	The Shawshank Redemption	Bob Gunton	Warden Norton
4	The Shawshank Redemption	William Sadler	Heywood
5	The Shawshank Redemption	Clancy Brown	Captain Hadley
6	The Shawshank Redemption	Gil Bellows	Tommy
7	The Shawshank Redemption	Mark Rolston	Bogs Diamond
8	The Shawshank Redemption	James Whitmore	Brooks Hatlen
9	The Shawshank Redemption	Jeffrey DeMunn	1946 D.A.
10	The Shawshank Redemption	Larry Brandenburg	Skeet
11	The Shawshank Redemption	Neil Giuntoli	Jigger
12	The Shawshank Redemption	Brian Libby	Floyd
13	The Shawshank Redemption	David Proval	Snooze
14	The Shawshank Redemption	Joseph Ragno	Ernie
15	The Shawshank Redemption	Jude Ciccolella	Guard Mert
16	The Godfather	Marlon Brando	Don Vito Corleone
17	The Godfather	Al Pacino	Michael Corleone
18	The Godfather	James Caan	Sonny Corleone
19	The Godfather	Richard S. Castellano	Clemenza (as Richard Castellano)
20	The Godfather	Robert Duvall	Tom Hagen
21	The Godfather	Sterling Hayden	Capt. McCluskey
22	The Godfather	John Marley	Jack Woltz
23	The Godfather	Richard Conte	Barzini

Meet `BeautifulSoup`

Make the source (`html`) into a `BeautifulSoup`

```
from bs4 import BeautifulSoup


soup = BeautifulSoup(web_page.content)
```

[python]

	movie_title	actor_name	character
1	The Shawshank Redemption	Tim Robbins	Andy Dufresne
2	The Shawshank Redemption	Morgan Freeman	Ellis Boyd 'Red' Redding
3	The Shawshank Redemption	Bob Gunton	Warden Norton
4	The Shawshank Redemption	William Sadler	Heywood
5	The Shawshank Redemption	Clancy Brown	Captain Hadley
6	The Shawshank Redemption	Gil Bellows	Tommy
7	The Shawshank Redemption	Mark Rolston	Bogs Diamond
8	The Shawshank Redemption	James Whitmore	Brooks Hatlen
9	The Shawshank Redemption	Jeffrey DeMunn	1946 D.A.
10	The Shawshank Redemption	Larry Brandenburg	Skeet
11	The Shawshank Redemption	Neil Giuntoli	Jigger
12	The Shawshank Redemption	Brian Libby	Floyd
13	The Shawshank Redemption	David Proval	Snooze
14	The Shawshank Redemption	Joseph Ragno	Ernie
15	The Shawshank Redemption	Jude Ciccolella	Guard Mert
16	The Godfather	Marlon Brando	Don Vito Corleone
17	The Godfather	Al Pacino	Michael Corleone
18	The Godfather	James Caan	Sonny Corleone
19	The Godfather	Richard S. Castellano	Clemenza (as Richard Castellano)
20	The Godfather	Robert Duvall	Tom Hagen
21	The Godfather	Sterling Hayden	Capt. McCluskey
22	The Godfather	John Marley	Jack Woltz
23	The Godfather	Richard Conte	Barzini

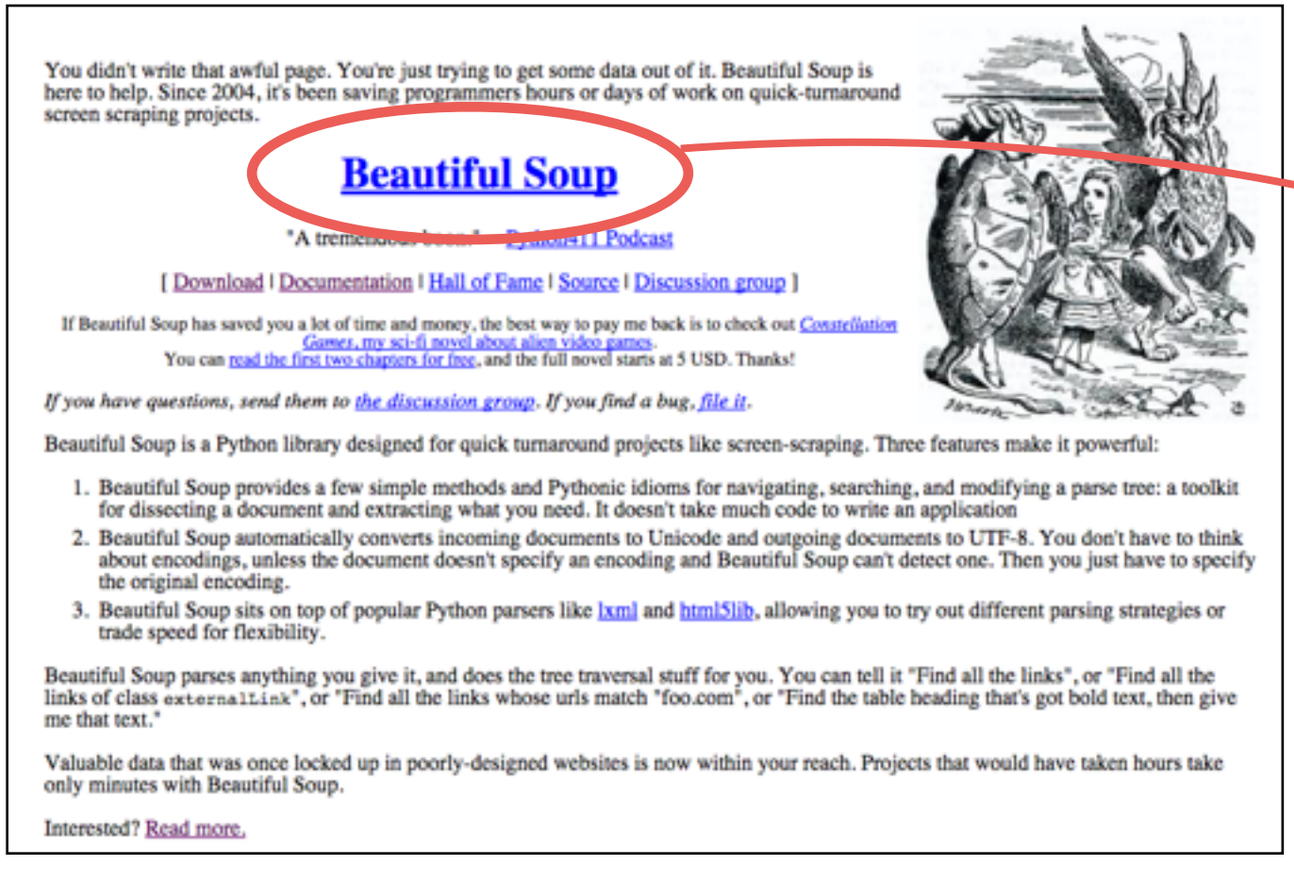
Use BeautifulSoup

```python
soup.h1
```

[python]

```
<h1>Beautiful Soup</h1>
```

[output]



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.

Beautiful Soup

"A tremendous [illegible] Podcast

[Download | Documentation | Hall of Fame | Source | Discussion group]

If Beautiful Soup has saved you a lot of time and money, the best way to pay me back is to check out *Constellation Games*, my sci-fi novel about alien video games.
You can read the first two chapters for free, and the full novel starts at 5 USD. Thanks!

If you have questions, send them to the discussion group. If you find a bug, file it.

Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. Three features make it powerful:

1. Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
2. Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.
3. Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

Beautiful Soup parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class `externalLink`", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."

Valuable data that was once locked up in poorly-designed websites is now within your reach. Projects that would have taken hours take only minutes with Beautiful Soup.

Interested? Read more.

Did we just get this?

What's Happening?

```
21 <div align="center">
22
23 <a href="bs4/download/"><h1>Beautiful Soup</h1></a>
24
25 <p>"A tremendous boon." -- <a
26 href="http://www.awaretek.com/python/index.html">Pyt
27 Podcast</a></p>
28
29 <p>[ <a href="#Download">Download</a> | <a
30 href="bs4/doc/">Documentation</a> | <a href="#HallOf
   href="https://code.launchpad.net/beautifulsoup">Sour
   href="https://groups.google.com/forum/?fromgroups#!f
   group</a> ]</p>
31
32 <small>If Beautiful Soup has saved you a lot of time
33 best way to pay me back is to check out <a
34 href="http://www.candlemarkandgleam.com/shop/constel
   <i>Constellation
35 Games</i>, my sci-fi novel about alien video games</
36 <a
```

- `soup.h1` gives us the content surrounded by `<h1>` and `</h1>`

Basics of `html`

`html` is made of tags that look something like this

opening tags(`<a>`) and closing tags(`</a>`)

```
<a href="#place">goto places</a>
```

Basics of `html`

`html` is made of tags that look something like this

```
<a href="#place">goto places</a>
```

the tag's name

contents that the tag applies to

properties of the tag. usually in the form of
`{property name}="{assigned value}"`

Web Scraper’s Rule of thumb

“If it looks different,

or does something different,

it’s probably in a different tag”

What's Happening?

```
21 <div align="center">
22
23 <a href="bs4/download/">h1>Beautiful Soup</h1></a>
24
25 <p>"A tremendous boon." -- <a
26 href="http://www.awaretek.com/python/index.html">Pyt
27 Podcast</a></p>
28
29 <p>[ <a href="#Download">Download</a> | <a
30 href="bs4/doc/">Documentation</a> | <a href="#HallOf
   href="https://code.launchpad.net/beautifulsoup">Sour
   href="https://groups.google.com/forum/?fromgroups#!f
   group</a> ]</p>
31
32 <small>If Beautiful Soup has saved you a lot of time
33 best way to pay me back is to check out <a
34 href="http://www.candlemarkandgleam.com/shop/constel
   <i>Constellation
35 Games</i>, my sci-fi novel about alien video games</
36 <a
```

- The `<a>` and `</a>` make the text between them into a *link*
- The `href="bs4/download/"` indicates where the link should link to

Get a link and its address

```python
soup.a
```

[python]

```
<a href="bs4/download/"><h1>Beautiful Soup</h1></a>
```

[output]

Now let's `get` the address (the value assigned to `href`)

```python
soup.a.get('href')
```

[python]

```
'bs4/download/'
```

[output]

html Structure

html is made of tags that look something like this

```
<html>
    <head></head>
    <body>
    </body>
</html>
```

<html> tag declares the beginning/end of document

info/scripts etc. go between <head></head>

most of the main content will be in the <body></body>

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.

Beautiful Soup

"A tremendous boon." -- Python411 Podcast

[Download | Documentation | Hall of Fame | Source | Discussion group]

If Beautiful Soup has saved you a lot of time and money, the best way to pay me back is to check out *Constellation Games*, my sci-fi novel about alien video games.
You can read the first two chapters for free, and the full novel starts at 5 USD. Thanks!

If you have questions, send them to the discussion group. If you find a bug, file it.

Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. Three features make it powerful:

1. Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
2. Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.
3. Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

Beautiful Soup parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class `externalLink`", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."

Valuable data that was once locked up in poorly-designed websites is now within your reach. Projects that would have taken hours take only minutes with Beautiful Soup.

Interested? Read more.

What about all these other links?

Get more links and addresses

Lets `find_all` the links (a)

```python
soup.find_all('a')
```

```output
[<a href="bs4/download/"><h1>Beautiful Soup</h1></a>, <a href="http://www.awaretek.com/python/index.html">Python411 Podcast</a>, ...
...
... , <a href="http://www.crummy.com/">http://www.crummy.com/</a>, <a href="http://www.crummy.com/software/">software/</a>, <a href="http://www.crummy.com/software/BeautifulSoup/">BeautifulSoup/</a>]
```

- Notice `[ ..., ..., ...]` is a list in `python`
- We can `iterate` through a `list` with a `for` loop

Get more links and addresses

Lets `get` all the addresses with a `for` loop

```
for link in soup.find_all('a'):
    link.get('href')
```
[python]

```
'bs4/download/'
'http://www.awaretek.com/python/index.html'
...
'http://www.crummy.com/software/'
'http://www.crummy.com/software/BeautifulSoup/'
```
[output]

...or make a `list` of addresses with list comprehension

```
addresses = [link.get('href') for link in soup.find_all('a')]
```
[python]

```
['bs4/download/', 'http://www.awaretek.com/python/index.html', ...]
```
[output]

Goto IMDb.com

Today's **GOAL**

Collect the **cast overview** (actor and character played) for each of the **Top 10 movies** of **IMDb Charts' Top 250** (http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

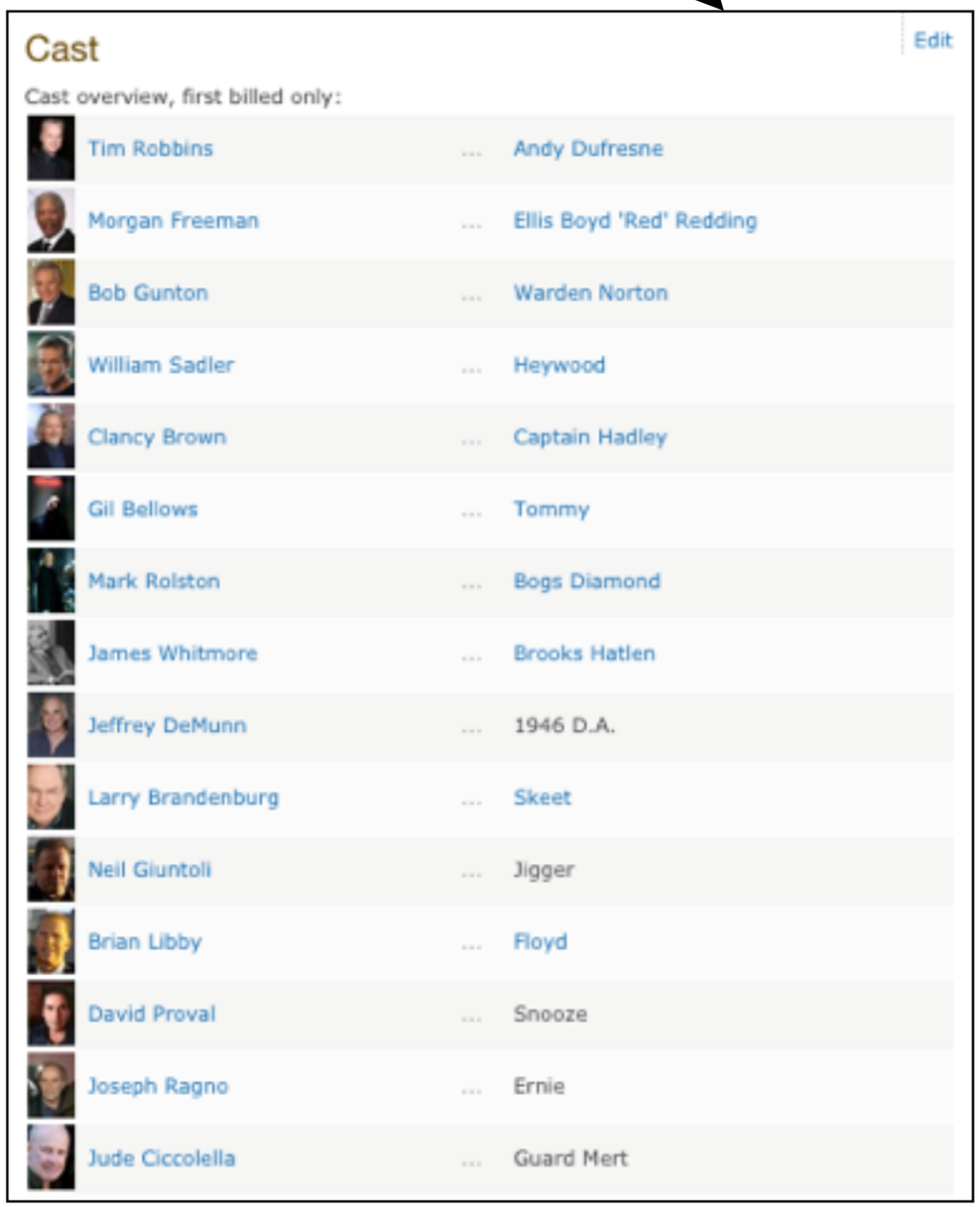
Let's start with

"the **cast overview** (actor and character played)"

for just *one* movie.

We want this

...start by making a soup



Cast

Edit

Cast overview, first billed only:

Tim Robbins	...	Andy Dufresne
Morgan Freeman	...	Ellis Boyd 'Red' Redding
Bob Gunton	...	Warden Norton
William Sadler	...	Heywood
Clancy Brown	...	Captain Hadley
Gil Bellows	...	Tommy
Mark Rolston	...	Bogs Diamond
James Whitmore	...	Brooks Hatlen
Jeffrey DeMunn	...	1946 D.A.
Larry Brandenburg	...	Skeet
Neil Giuntoli	...	Jigger
Brian Libby	...	Floyd
David Proval	...	Snooze
Joseph Ragno	...	Ernie
Jude Ciccolella	...	Guard Mert

(http://www.imdb.com/title/tt0111161/?ref_=chttp_tt_1)

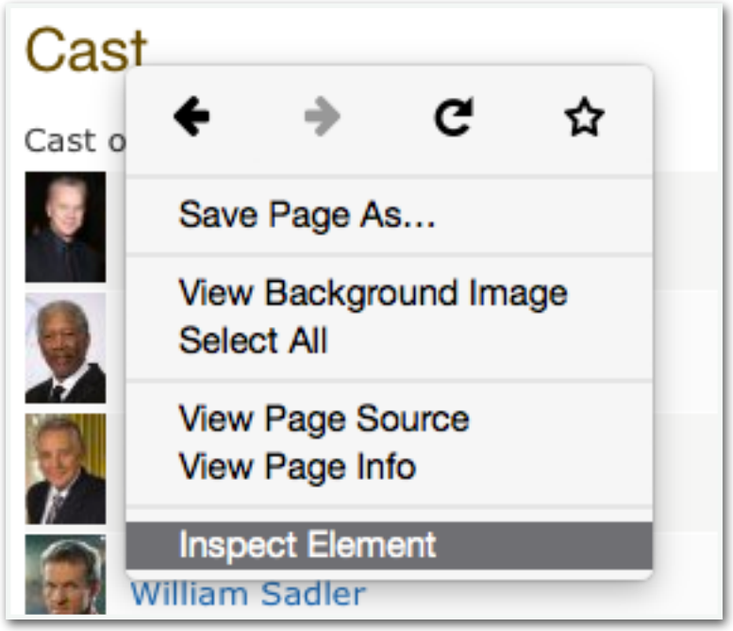
```python
from bs4 import BeautifulSoup
import requests


web_page = requests.get('http://...')
soup = BeautifulSoup(web_page.content)
```

[python]

Find Stuff in a Soup of `html`

Using your browser's Developer Mode

(A demo is better than a thousand slides)



But in case you forget, it's

`[right click]`

`> [Inspect Element]`

... in most modern browsers

finding a Specific tag

```
<table class='cast_list'>
```

```python
soup.find('table', class_='cast_list')
```

Note that we use `class_` instead of `class`.

This is because `class` means something else in `python`.

Anything other than `class`, you should use as is.

What's in a `table`?


```
<table>
    <tr>
        <td></td>
        <td></td>
    </tr>
    <tr>
        <td></td>
    </tr>
</table>
```

within a `table` tag,

`<tr>` defines rows

while `<td></td>` defines columns within rows.

`html` tables are written row-by-row

Iterate a `table` by its rows

We want to `iterate` each row of our strained `soup`

(`soup.find('table', class_='cast_list')`)

```python
for row in soup.find('table', class_='cast_list').find_all('tr'):
    # do useful things with each row
```

Picking out the Cherries

How should we identify the **actor's name** and **character** played, given a single row (`<tr>`)?

```
▶ <tr>…</tr>
▼ <tr class="odd">
  ▶ <td class="primary_photo">…</td>
  ▼ <td class="itemprop" itemprop="actor" itemscope itemtype="http://
  schema.org/Person">
    ▼ <a href="/name/nm0000209/?ref_=tt_cl_t1" itemprop="url">
        <span class="itemprop" itemprop="name">Tim Robbins</span>
      </a>
    </td>
    <td class="ellipsis">
                ...
            </td>
  ▼ <td class="character">
    ▼ <div>
        <a href="/character/ch0001388/?ref_=tt_cl_t1">Andy Dufresne</a>
      </div>
    </td>
  </tr>
```

An actor's name seems to be uniquely identified by the property `itemprop="name"`

The column containing the character name has `class="character"`

```
▶ <tr>…</tr>
▼ <tr class="odd">
  ▶ <td class="primary_photo">…</td>
  ▼ <td class="itemprop" itemprop="actor" itemsc
schema.org/Person">
    ▼ <a href="/name/nm0000209/?ref_=tt_cl_t1" itemprop="url">
        <span class="itemprop" itemprop="name">Tim Robbins</span>
      </a>
    </td>
    <td class="ellipsis">
                ...
            </td>
  ▼ <td class="character">
    ▼ <div>
        <a href="/character/ch0001388/?ref_=tt_cl_t1">Andy Dufresne</a>
      </div>
    </td>
```

(There's usually more than one way ...)

An actor's name seems to be uniquely identified by the property `itemprop="name"`

```python
for row in soup. ... .find_all('tr'):
    actor = row.find(itemprop='name').text
    role = row.find(class_='character').text
```

[python]

Save the values so we can format them appropriately later (in our case, into tab-separated value)

The column containing the character name has `class="character"`

```python
for row in soup. ... .find_all('tr'):
    actor = row.find(itemprop='name').text
    role = row.find(class_='character').text
```

[python]

```output
...
AttributeError: 'NoneType' object has no
attribute 'text'
```

[output]

`find_all('tr')` gives an extra row, which doesn't have anything that matches `itemprop='name'`

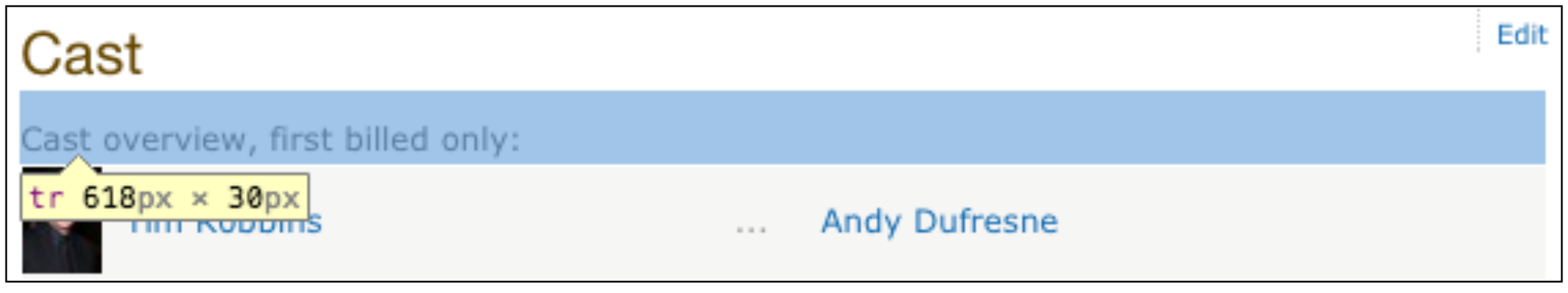
```python
for row in soup. ... find_all('tr')
    actor = row.find(itemprop='name').text
    role = row.find(class_='character').text
```

[python]

we want `python` to ignore these errors

```output
...
AttributeError: 'NoneType' object has no attribute 'text'
```

[output]



Manage with `try-except` Blocks

```python
for row in soup. ... .find_all('tr'):
    try:
        
    except AttributeError:
        
```

python will `try` this

WARNING!

Practice caution with `try-except`.

Don't `pass` an error, unless you're certain you it's an error you *want to* ignore.

Manage with `try-except` Blocks

```
for row in soup. ... .find_all('tr'):
    try:
        actor = row.find(itemprop='name').text
        role = row.find(class_='character').text

    except AttributeError:
        pass
```

[python]

python will `try` this

if an `AttributeError` happens, it will do this (in this case, ignore the error and `pass`)

WARNING!

Practice caution with `try-except`. Don't `pass` an error, unless you're certain you it's an error you *want to* ignore.

```python
for row in soup. ... .find_all('tr'):
    try:
        actor = row.find(itemprop='name').text
        role = row.find(class_='character').text

    except AttributeError:
        pass
```

[python]

```output
u'Tim Robbins'
u'\n\nAndy Dufresne\n\n'
...
```

[output]

What's with all the
`\n\n\n\n\n\n` ... ?

Web designers sometimes use hidden white spaces for layout purposes.

A good way to deal with these in `python` is to surround your string with `' '.join(string.split())`

```python
for row in soup. ... .find_all('tr'):
    try:
        actor = row.find(itemprop='name').text
        role = row.find(class_='character').text

    except AttributeError:
        pass
```

[python]

Write a `function` that does this with any given `string`. You might find yourself doing this quite often.

```python
def clean_text(text):
    return ' '.join(text.split())
```

[python]

Web designers sometimes use hidden white spaces for layout purposes.

A good way to deal with these in `python` is to surround your string with `' '.join(string.split())`

```python
for row in soup. ... .find_all('tr'):
    try:
        actor = clean_text(row.find(itemprop='name').text)
        role = clean_text(row.find(class_='character').text)

    except AttributeError:
        pass
```
[python]

```python
def clean_text(text):
    return ' '.join(text.split())
```
[python]

Write the Data to a File

Usually, `print` is sufficient in `python`.

You can make tab-separated values, from a `list` of `strings`

```python
print '\t'.join([actor, role])
```

`print` will send strings to stdout, which means you can save the output to a file by redirecting like:

```terminal
> python script_name.py > movie_data.py
```

So far ...

```python
from bs4 import BeautifulSoup, SoupStrainer
import requests

def clean_text(text):
    return ' '.join(text.split())

web_page = requests.get('http://...')
soup = BeautifulSoup(web_page.content)

for row in soup.find('table', class_='cast_list').find_all('tr'):
    try:
        actor = clean_text(row.find(itemprop='name').text)
        role = clean_text(row.find(class_='character').text)

        print '\t'.join([actor, role])

    except AttributeError:
        pass
```

[python]

Finally!

Today's **GOAL**

Collect the **cast overview** (actor and character played) for each of the **Top 10 movies** of **IMDb Charts' Top 250** (http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

Workflow

http://www.imdb.com/chart/top?ref_=nv_ch_250_4

`for` these `links` do ***this***

```python
from bs4 import BeautifulSoup, SoupStrainer
import requests

def clean_text(text):
    return ' '.join(text.split())

web_page = requests.get('http://...')
soup = BeautifulSoup(web_page.content)

for row in soup.find('table', class_='cast_list').find_all('tr'):
    try:
        actor = clean_text(row.find(itemprop='name').text)
        role = clean_text(row.find(class_='character').text)

        print '\t'.join([actor, role])

    except AttributeError:
        pass
```

[python]

Be Nice

Don’t harass the servers

Rule-of-thumb(?): one request every 3~5 seconds

Space your requests using `sleep`

```python
from time import sleep

...
for link in movie_list:
    # request the link and do your thing
    ...
    sleep(3)
```

[python]

Unit is seconds.

So, `sleep(3)` = one request every three seconds

DIY

Workflow

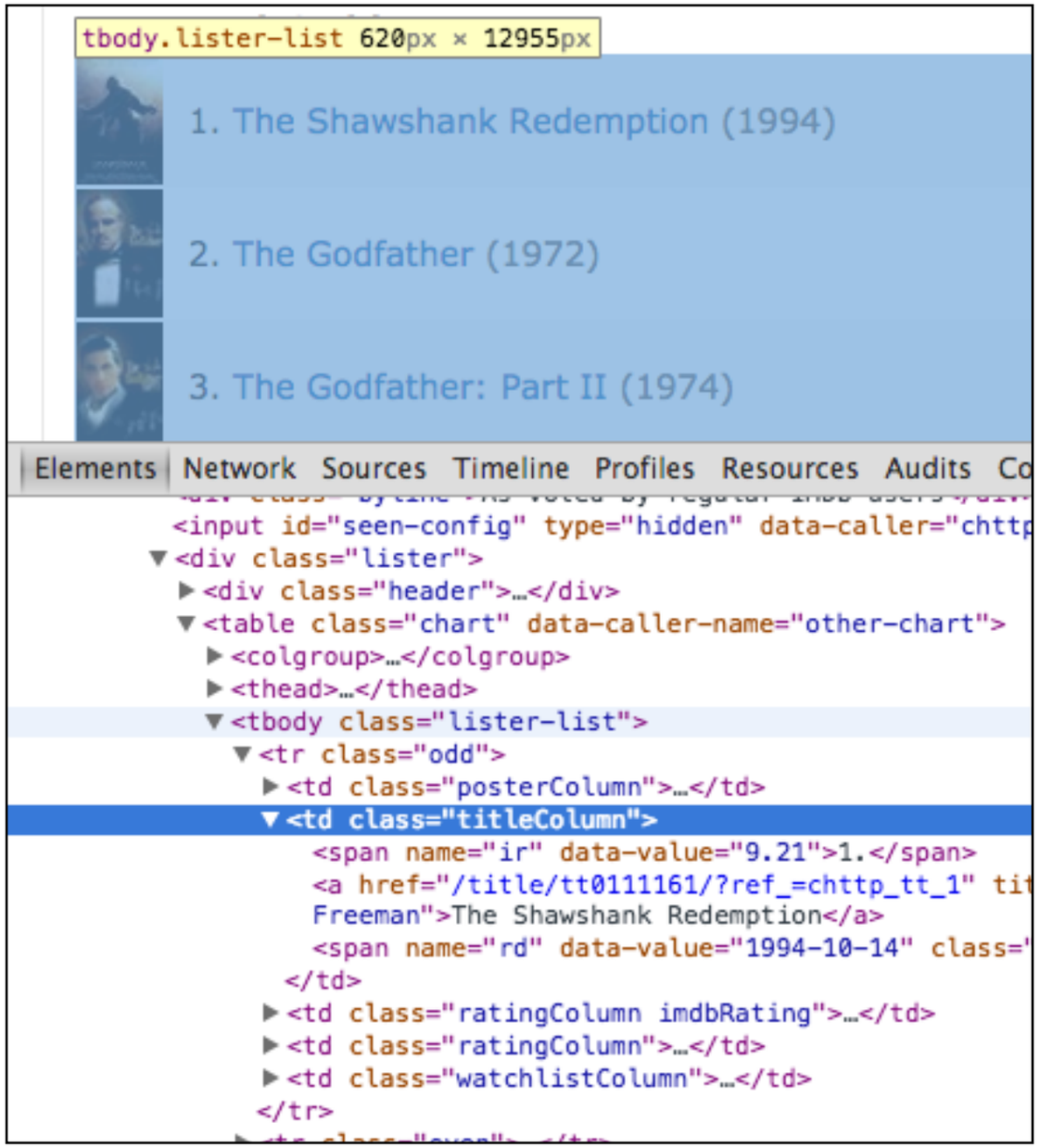
http://www.imdb.com/chart/top?ref_=nv_ch_250_4

`for` these `links` do ***this***

```python
from bs4 import BeautifulSoup, SoupStrainer
import requests

def clean_text(text):
    return ' '.join(text.split())

web_page = requests.get('http://...')
soup = BeautifulSoup(web_page.content)

for row in soup.find('table', class_='cast_list').find_all('tr'):
    try:
        actor = clean_text(row.find(itemprop='name').text)
        role = clean_text(row.find(class_='character').text)

        print '\t'.join([actor, role])

    except AttributeError:
        pass
```

[python]

The Links

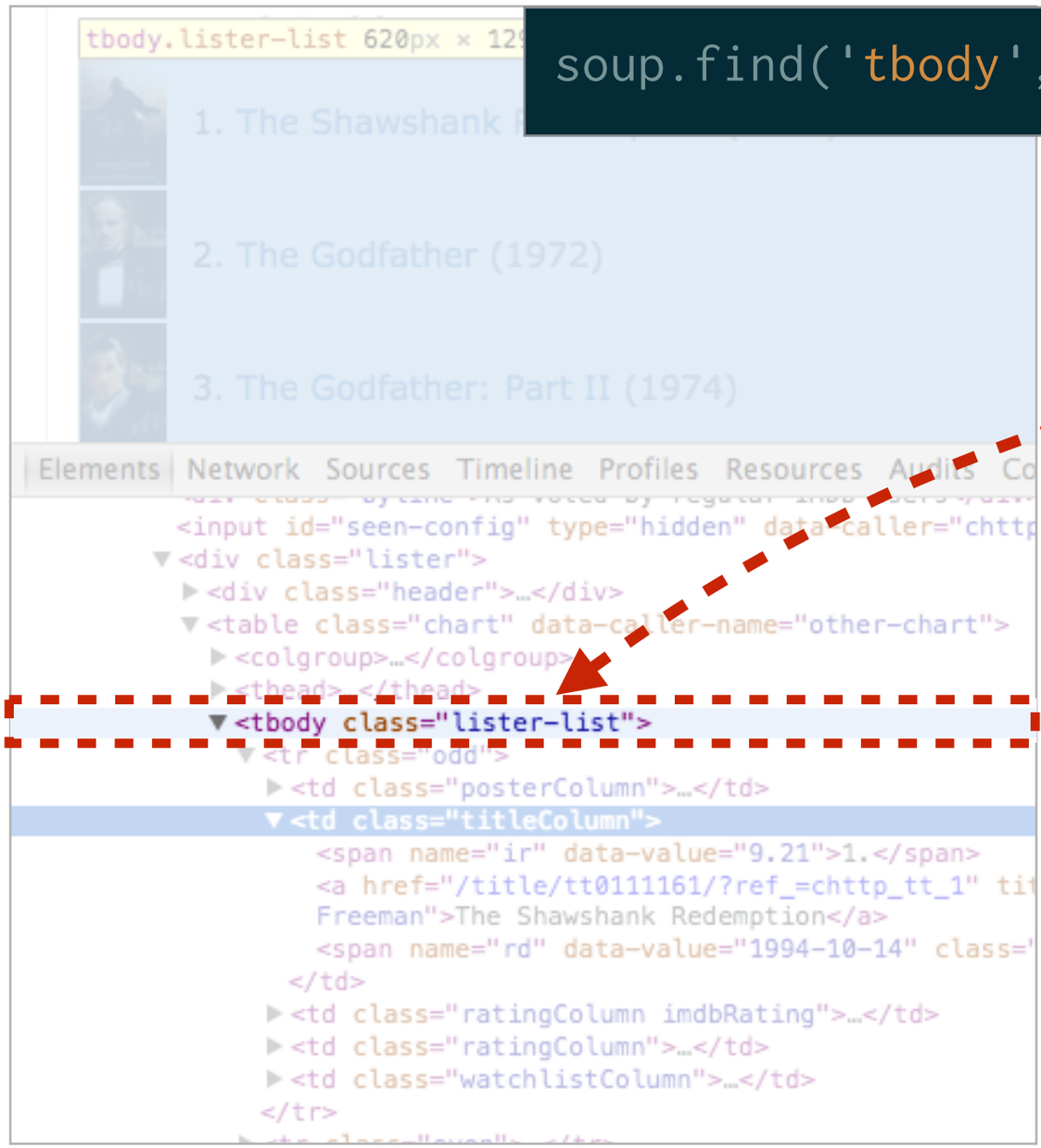


What do we need, and how should we get it?

remember:

```
soup.a.get('href')
[python]
```

The Links



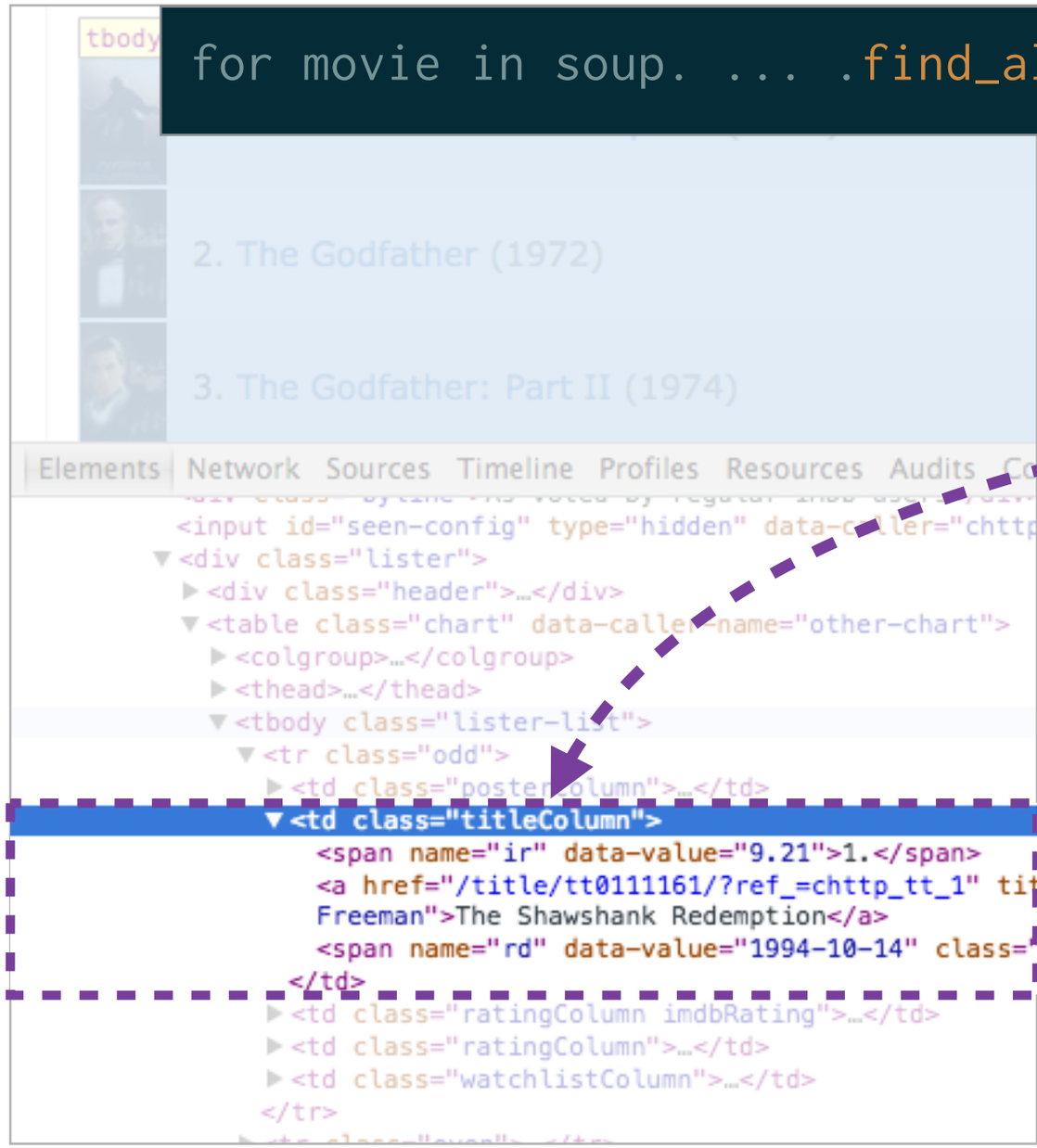

```python
soup.find('tbody', class_='lister-list')
```

[python]

<tbody> seems to be a tag indicating the body of a table!

The Links

```python
for movie in soup. ... .find_all('td', class_='titleColumn')
```

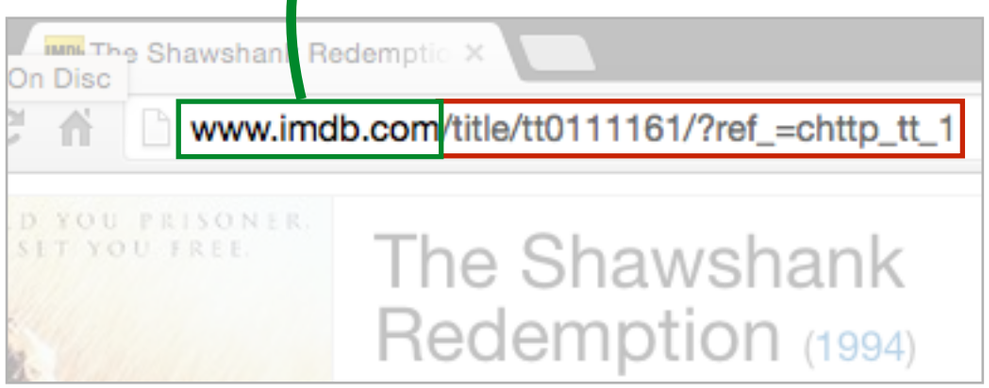


Each movie title, surrounded by an <a> tag, seems to be contained in `<td>` with `class="titleColumn"`

The Links

Build a link from the `href` value

```python
link = 'http://imdb.com' + movie.a.get('href')
```




```
<a href="/title/tt0111161/?ref_=chttp_tt_1" title="Frank Darabont
(dir.), Tim Robbins, Morgan Freeman">The Shawshank Redemption</a>
```

"**Top 10 movies** of **IMDb Charts' Top 250**"

```python
soup. ... .find_all('td', class_='titleColumn', limit=10)
```

The `limit` option will return only the first `10` results of `find_all`

Workflow

`requests` ▶ `BeatifulSoup` ▶ `.get('href')` ▶ `requests` …

```python
web_page = requests.get('http://...')

soup = BeautifulSoup(web_page.content)

table_body = soup.find('tbody', class_='lister-list')
movie_list = table_body.find_all('td', class_='titleColumn', limit=10)

for movie in movie_list:
    movie_title = movie.a.text

    link = 'http://imdb.com' + movie.a.get('href')
    get_cast_list_from_one_movie(link)
    ...
```

[python]

Tips & Tricks

`Strain`ing the Soup

When dealing with only a small portion of the entire page (like ourselves), it might speed things up a little to `strain` the soup with `SoupStrainer`, before `find`ing things in it.

```python
from bs4 import BeautifulSoup, SoupStrainer
import requests

cast_strainer = SoupStrainer('table', class_='cast_list')

web_page = requests.get('http://...')
soup = BeautifulSoup(web_page.content, parse_only=cast_strainer)
```

[python]

Write the Data to a File

Usually, `print` is sufficient in `python`.

But `print` doesn't play well with weird characters ... (i.e., non `ascii`)

But the web is full of weird characters!

One workaround is to use `codecs`

```python
import codecs
...
with codecs.open('file_name.tsv', 'a', encoding='utf-8') as fid:
    print>> fid, '\t'.join([actor, role])
```

[python]

`PRO:` Works with most languages on the web (Chinese, Korean, ...)

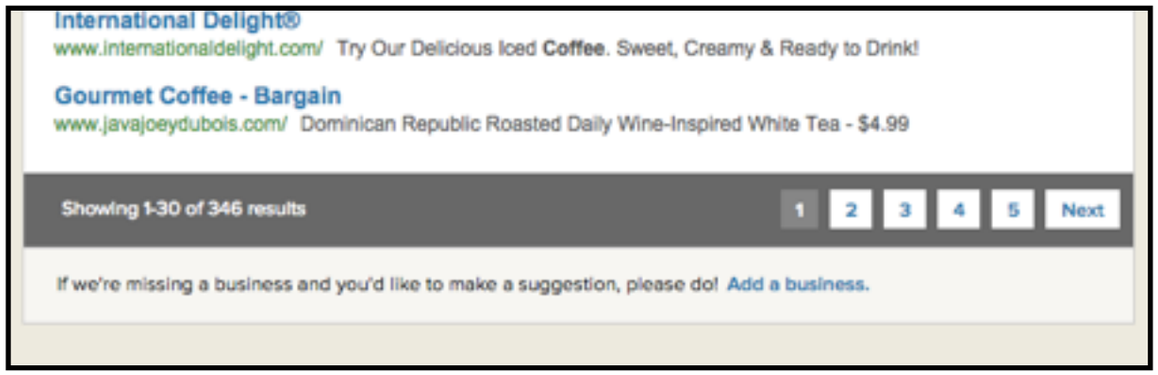
`CON:` Trickier to pipe/redirect output using command line

Web Scraping Part2

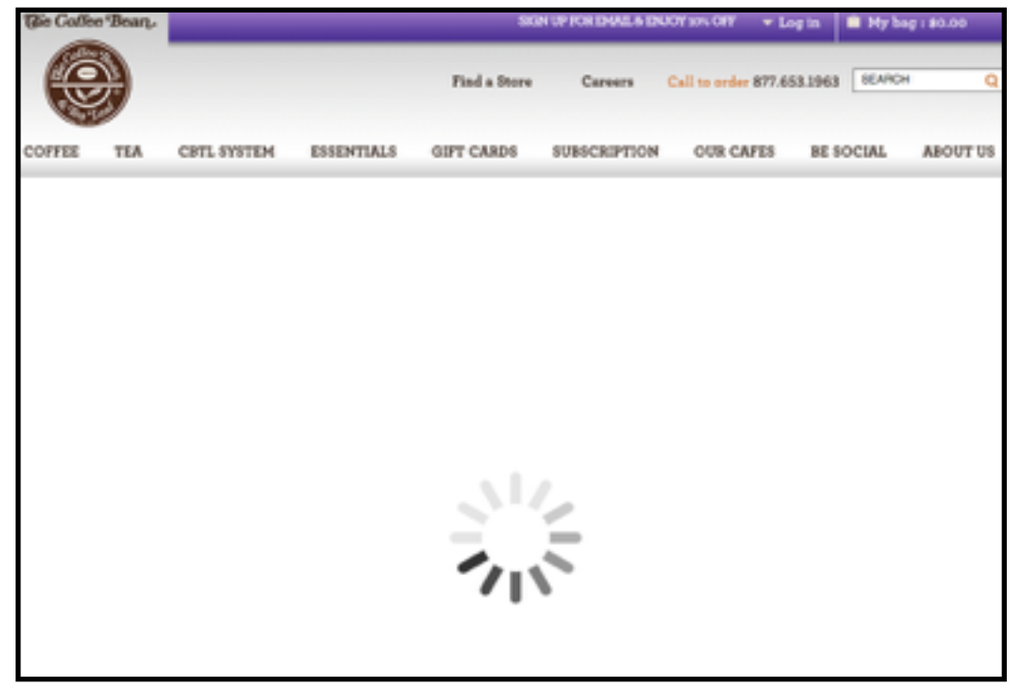
a **Data Science Drop-in** Tutorial

by **Houshmand Shirani-Mehr**
(hshirani@stanford.edu)

Some Obstacles



Results on Multiple Pages



Dynamic Web Page

Results on Multiple Pages

Today's **GOAL**

Collect the **cast list** of **Top 250 comedy movies** sorted based on US Box Office Income (http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us)

Pagination

Sometimes data we are interested in is on multiple pages. For instance, IMDB list of top comedy movies.

http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us
http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us&start=51
http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us&start=101

Structure of URL

http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us&start=51

List of comedy movies

Sorted by US box office gross income

Offset = 51

Change the offset and you'll have different pages of the list.

```python
url_base = 'http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us'
for i in range(1,250,50):
    # Concatenate two parts of the URL string
    url = url_base + '&start=' + str(i)
```

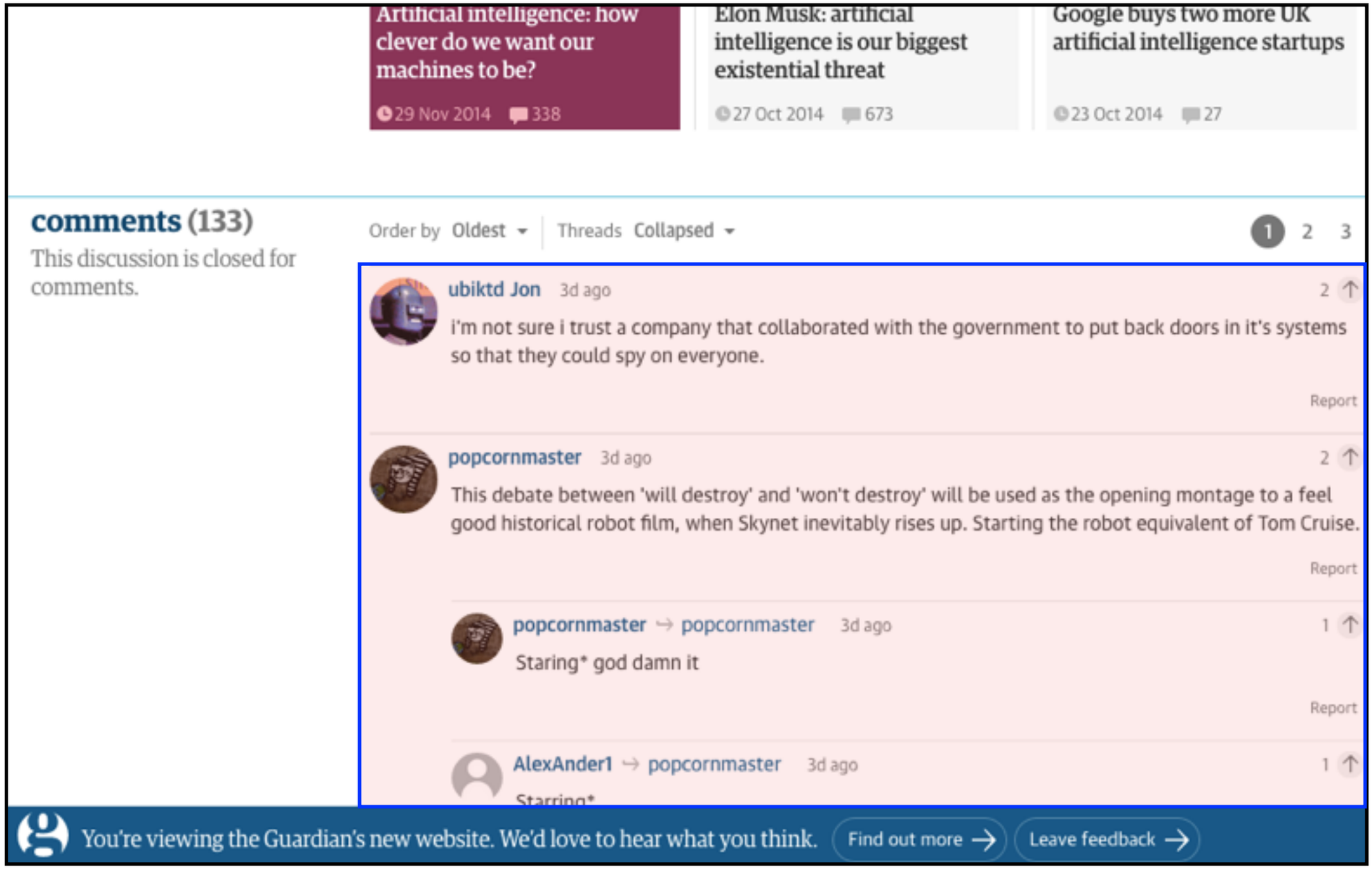
[python]

Next

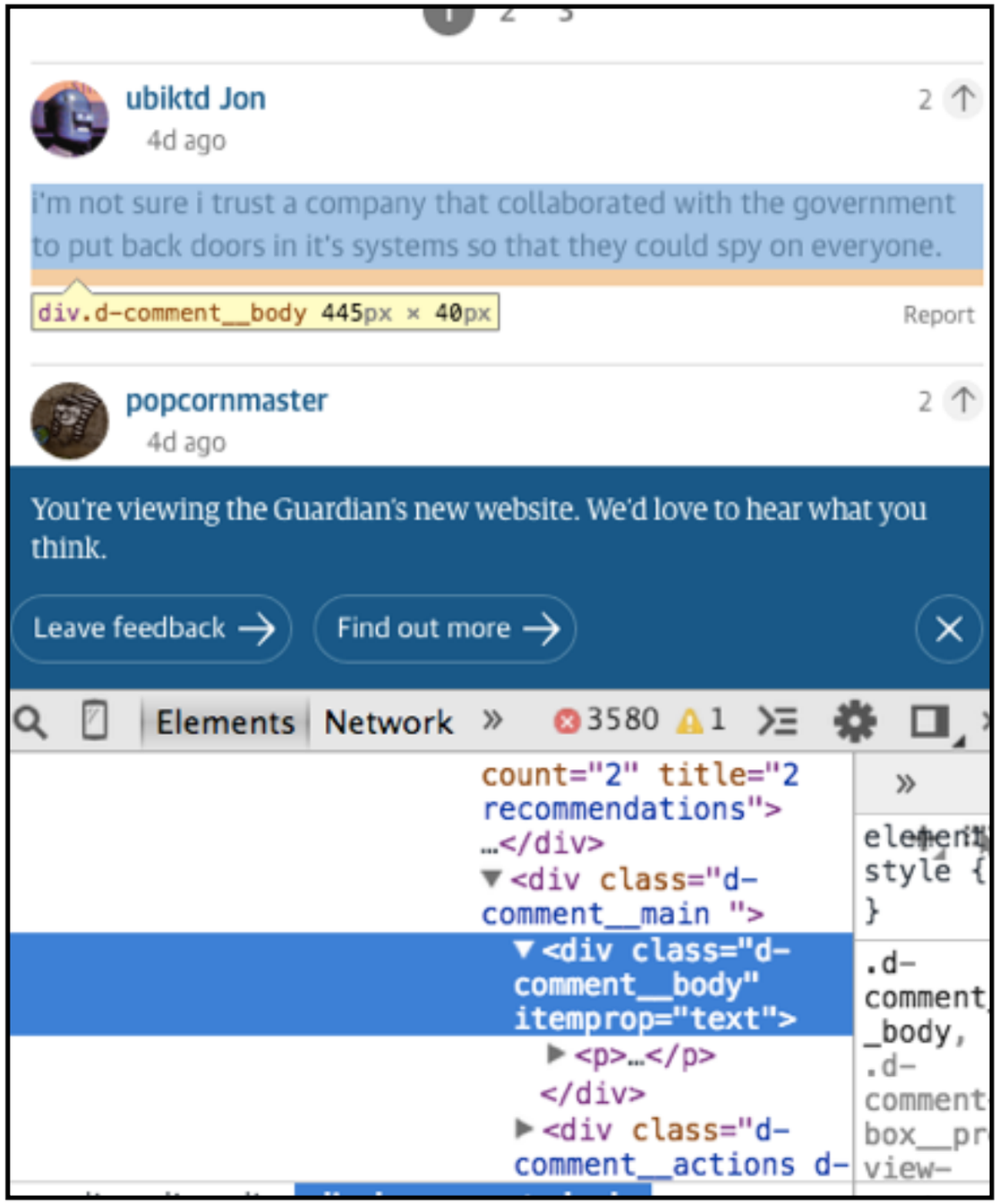
Today's **GOAL**

Collect **comments** of a news article on **The Guardian** (http://www.theguardian.com/technology/2015/jan/28/artificial-intelligence-will-not-end-human-race)

Extract Comments



Our Old Ways



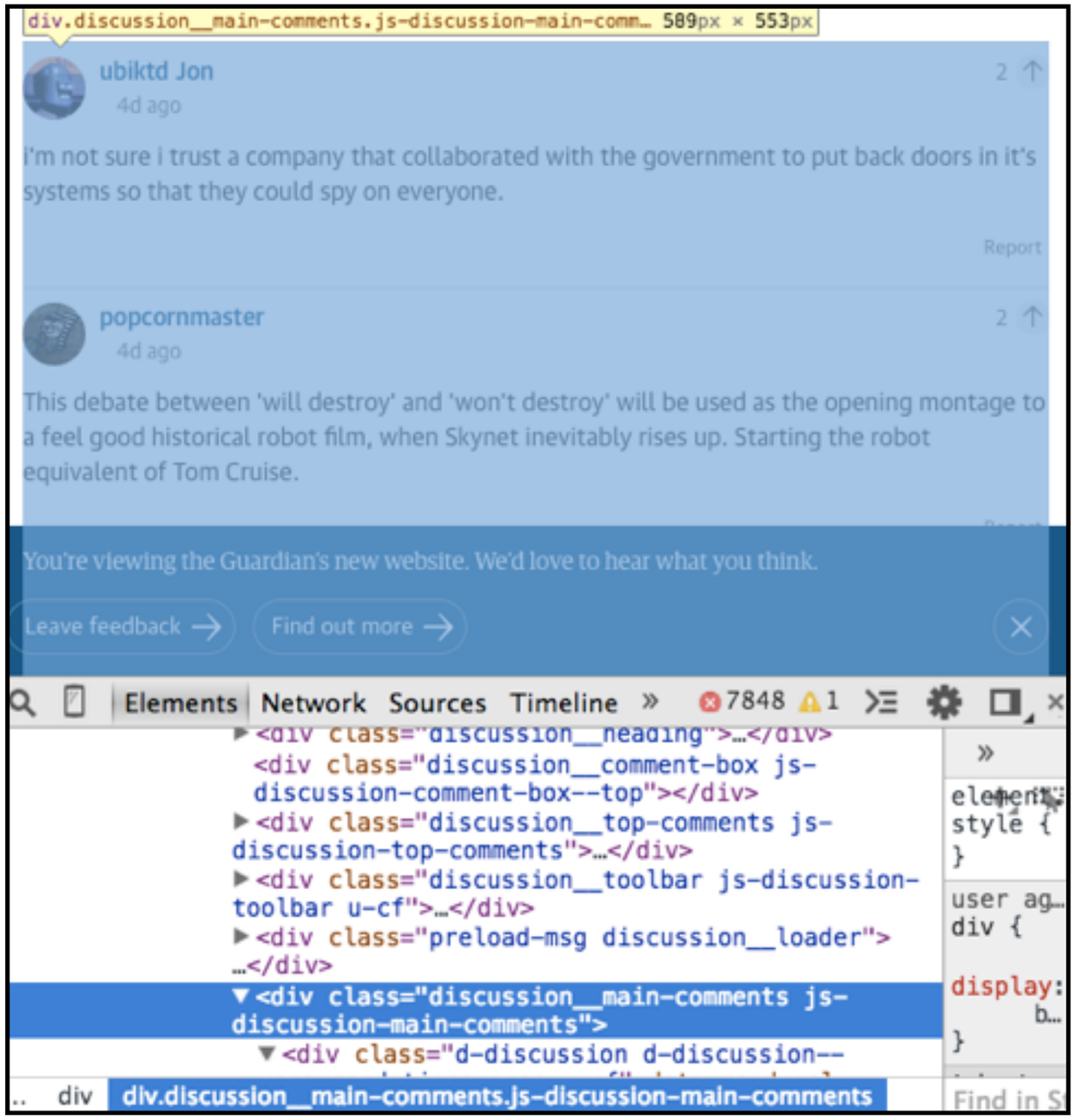

```python
url_base = 'http://www.theguardian.com/technology/2015/jan/28/artificial-intelligence-will-not-end-human-race'
response = requests.get(url)
soup = BeautifulSoup(response.content)
mydivs = soup.findAll('div', class_ = 'd-comment__body', limit = 1)
```

[python]

mydivs will be empty!

Compare

Inspect Element



Page Source

The element containing comments is empty in the source code!

Dynamic Content

Inspect Element

Shows the page content after scripts are run on the server side.

Page Source

Shows the initial content of the page before scripts are run.

`Requests` gets the initial content of the page, same as what we see from Page Source.

`Solution:`

`Use Selenium`

Selenium

Python library to mechanize the browser.

```python
from selenium import webdriver
browser = webdriver.Firefox()

url = 'http://www.imdb.com'
browser.get(url)
```

Opens a FireFox window,
loads IMDB website.

Scrape Dynamic Web Pages

When using `Selenium`, we can wait for a certain element to load, or wait for a certain amount of time, and then look for the data we want to extract in the Page.

```python
url_base = 'http://www.theguardian.com/technology/2015/jan/28/
artificial-intelligence-will-not-end-human-race'
browser.get(url)

time.sleep(5)
page_content = browser.page_source
```

[python]

`page_content` now contains the comments.

Also Consider API's

API (Application Program Interface):

Provides programmatic access to a website's data.

+ **Structured Data**
+ **Easier than web scraping**
- **You have to register for a key and use it with every request, so not anonymous**
- **Limited rate and access**

Thank you.

Data Science Drop-in

https://5harad.com/drop-in/

Mondays @ 4–6 pm in Y2E2 253