

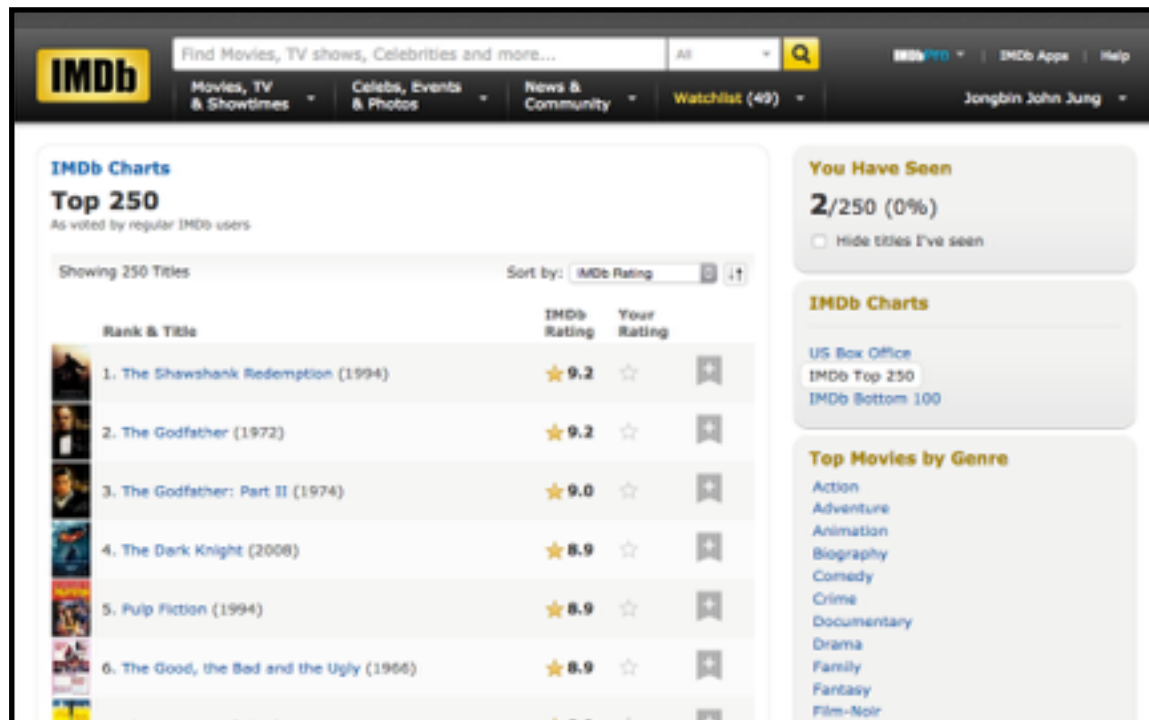
# Web Scraping Part 1

a **Data Science Drop-in** Tutorial  
by **Jongbin Jung**  
([jongbin@stanford.edu](mailto: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 Scrapping?



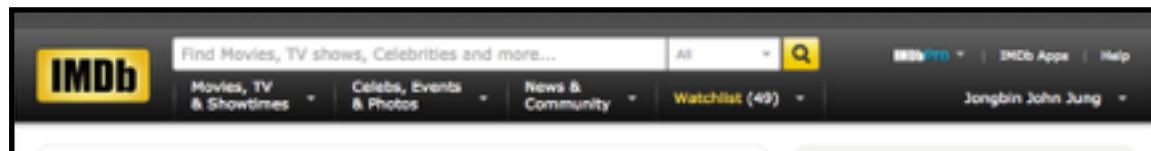
	movie_title	actor_name	character
1	The Shawshank Redemption	Tim Robbins	Andy Dufresne
2	The Shawshank Redemption	Morgan Freeman	Ellis Boyd 'Red' Redding
3	The Shawshank Redemption	Bob Gunton	Warden Norton
4	The Shawshank Redemption	William Sadler	Heywood
5	The Shawshank Redemption	Clancy Brown	Captain Hadley
6	The Shawshank Redemption	Gil Bellows	Tommy
7	The Shawshank Redemption	Mark Rolston	Bugs 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

```
<div class="pending"></div>
<div class="unavailable">NOT RATED</div>
<div class="unseen"></div>
<div class="rating"></div>
<div class="seen">Seen</div>
</div>
</div>
</div>
<div class="watchlistColumn"> <div class="wlb_ribbon" data-bbox="1108130"></div>
</div>
</div>
<div class="posterColumn"><a href="/title/tt0448153/?ref=chttp_tt_114"
+img src="http://ia.media-imdb.com/images/G/01/907x1802/26/1197/91/Shawshank Redemption.jpg" width="74"
height="50" />
</div>
</div>
<div class="titleColumn">
<span name="tr" data-value="8.948">8.9</span>
<a href="/title/tt0448153/?ref=chttp_tt_114"
title="Pete Dinkler (dir.), Edward Asner, Jordan Nagai" data-value="2009-03-20" class="secondaryInfo">(2009)</span>
</div>
<div class="ratingColumn imdbRating">
<strong name="tr" data-value="8.948" title="8.2 based on 504,460 votes">8.2</strong>
</div>
<div class="ratingColumn"><span name="tr" data-value="8">8</span> <div class="seen-widget seen-widget-tt0448153 pending" data-
title="tt0448153">
<div class="boundary">
<div class="progressbar">
<div class="pending"></div>
<div class="unavailable">NOT RATED</div>
<div class="unseen"></div>
<div class="rating"></div>
<div class="seen">Seen</div>
</div>
</div>
</div>
</div>
<div class="watchlistColumn"> <div class="wlb_ribbon" data-bbox="1108413"></div>
</div>
</div>
<div class="posterColumn"><a href="/title/tt0448153/?ref=chttp_tt_114"
+img src="http://ia.media-imdb.com/images/G/01/907x1802/26/1197/91/Shawshank Redemption.jpg" width="74"
height="50" />
</div>
```

# Web Scrapping?



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

```
<title>"%1049413">  
    <div class="boundary">  
        <div class="tempore">  
            <span class="delete"><script>/script></script></div>  
            <div class="inLine">  
                <div class="pending"></div>  
                <div class="unreadable">NOT SET RELEASED</div>  
                <div class="unseen"></div>  
                <div class="waiting"></div>  
                <div class="xxx"><script></script>  
            </div>  
        </div>  
        <div class="watchlistColumn"> <div class="win Ribbon" data-source="%1049413"></div>  
        <div>  
            <div class="add">  
                <div class="postcardColumn"><a href="/title/%107043/?ref=script%1013">  
                    <script>https://a.media-amz.com/images/O/RYBCTMYWU6Q1GZ/Bt13baasRf123wS68T8w684_W1_PYS6_C02_0.38.59_XL.jpg" width="36
```

# get the Web to python

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

[illegible]

from source (html)  
to python

```

✓ jongbinjung@0000a27b2eb:~$ python
Python 2.7.8 [Anaconda 2.0.1 (x86_64)] (default, Aug 21 2014, 15:21:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>>

```

```
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](http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

# Meet BeautifulSoup

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

[illegible]

from source (html)  
to python

```

✓ jongbinjung@KNSba22b7eb:(master)$ python
Python 2.7.8 [Anaconda 2.0.1 (x86_64)] (default, Aug 21 2014, 15:21:48)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>>

```

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	Bugs 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]

```
✓ jongbinjung@Mba22b2eb:(master)$ python
Python 2.7.8 [Anaconda 2.0.1 (x86_64)] (default, Aug 21 2014, 15:21:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>>
```

	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	Gill Bellows	Tommy
7	The Shawshank Redemption	Mark Rolston	Bugs 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	Cleomenza (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

```
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. BeautifulSoup 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 benefit" - [Python411 Podcast](#)

[ [Download](#) | [Documentation](#) | [Hall of Fame](#) | [Source](#) | [Discussion group](#) ]

If BeautifulSoup has saved you a lot of time and money, the best way to pay me back is to check out [Constellation Gamer](#), 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. BeautifulSoup 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. BeautifulSoup 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 BeautifulSoup can't detect one. Then you just have to specify the original encoding.
3. BeautifulSoup 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 BeautifulSoup.

Interested? [Read more](#).



Did we just get this?



# What's Happening?

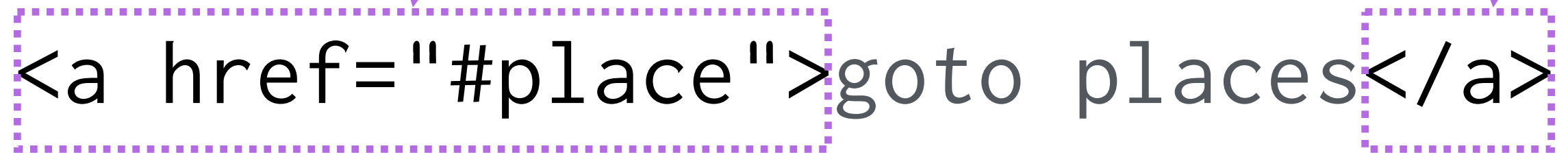
```
21 <div align="center">
22
23 <a href="bs4/download/"><h1>Beautiful Soup</h1></a>
24
25 <p>"A tremendous boon." -- <a
26 href="http://www.awaretek.com/python/index.html">Pyth
27 Podcast</a></p>
28
29 <p>[ <a href="#Download">Download</a> | <a
30 href="bs4/doc/">Documentation</a> | <a href="#Hallof
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>)



The diagram illustrates the structure of an HTML link. It shows the opening tag `<a href="#place">` and the closing tag `</a>` enclosed in dashed purple boxes. A purple arrow points from the text "opening tags(<a>)" to the opening tag box, and another purple arrow points from the text "closing tags(</a>)" to the closing tag box. The content "goto places" is positioned between the two tags.

```
<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">Pyth
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

```
soup.a
```

[python]

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

[output]

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

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

[python]

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

[output]

# html Structure

html is made of tags that look something like this

`<html>`

`<html>` tag declares the beginning/end of document

`<head></head>`

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

`<body>`

most of the main content

`</body>`

will be in the `<body></body>`

`</html>`



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)

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

[python]

```
[<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>]
```

[output]

- 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](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



The screenshot shows the IMDb Cast page for the movie 'The Shawshank Redemption'. The title 'Cast' is at the top left, and an 'Edit' link is at the top right. Below the title, it says 'Cast overview, first billed only:'. The cast members are listed in two columns, each with a small headshot, their name, and the character they played. The names are in blue text, and the character names are in a lighter blue text. An arrow points from the text 'We want this' to the cast list.

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

```
from bs4 import BeautifulSoup
import requests

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

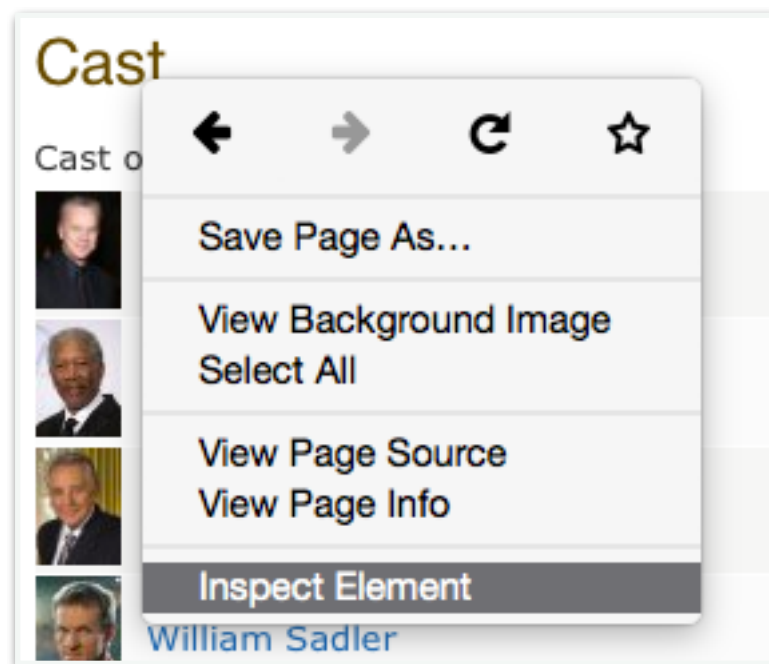
[python]

([http://www.imdb.com/title/tt0111161/?ref\\_=chttp\\_tt\\_1](http://www.imdb.com/title/tt0111161/?ref_=chttp_tt_1))

# 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'>

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

[python]

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')`)

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

[python]

# 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"`

```
▶ <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>
```

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

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

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

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



[python]

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

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

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

[python]

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

[output]

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

```
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



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

[output]





# Manage with `try-except` Blocks

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

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 will try this



[python]



## **WARNING!**

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

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

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

    except AttributeError:
        pass
```

[python]

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

[output]

What's with all the  
\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())

```
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.

```
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())

```
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]

```
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`

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

[python]

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

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

[terminal]

# So far ...

```
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](http://www.imdb.com/chart/top?ref_=nv_ch_250_4))

# Workflow



A screenshot of the IMDb top 10 movies list. The list is numbered 1 to 10, with each entry including a small movie poster icon, the movie title, and the year in parentheses. The movies are: 1. The Shawshank Redemption (1994), 2. The Godfather (1972), 3. The Godfather: Part II (1974), 4. The Dark Knight (2008), 5. Pulp Fiction (1994), 6. The Good, the Bad and the Ugly (1966), 7. 12 Angry Men (1957), 8. Schindler's List (1993), 9. The Lord of the Rings: The Return of the King (2003), and 10. Fight Club (1999).

1.	The Shawshank Redemption	(1994)
2.	The Godfather	(1972)
3.	The Godfather: Part II	(1974)
4.	The Dark Knight	(2008)
5.	Pulp Fiction	(1994)
6.	The Good, the Bad and the Ugly	(1966)
7.	12 Angry Men	(1957)
8.	Schindler's List	(1993)
9.	The Lord of the Rings: The Return of the King	(2003)
10.	Fight Club	(1999)

[http://www.imdb.com/chart/top?ref\\_=nv\\_ch\\_250\\_4](http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

for these links do ***this***

```
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`

```
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



A screenshot of the IMDb top 10 movies list. The list is numbered 1 to 10, with each entry showing a small movie poster icon, the movie title, and the year in parentheses. The movies are: 1. The Shawshank Redemption (1994), 2. The Godfather (1972), 3. The Godfather: Part II (1974), 4. The Dark Knight (2008), 5. Pulp Fiction (1994), 6. The Good, the Bad and the Ugly (1966), 7. 12 Angry Men (1957), 8. Schindler's List (1993), 9. The Lord of the Rings: The Return of the King (2003), and 10. Fight Club (1999).

1.	The Shawshank Redemption	(1994)
2.	The Godfather	(1972)
3.	The Godfather: Part II	(1974)
4.	The Dark Knight	(2008)
5.	Pulp Fiction	(1994)
6.	The Good, the Bad and the Ugly	(1966)
7.	12 Angry Men	(1957)
8.	Schindler's List	(1993)
9.	The Lord of the Rings: The Return of the King	(2003)
10.	Fight Club	(1999)

[http://www.imdb.com/chart/top?ref\\_=nv\\_ch\\_250\\_4](http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

for these links do ***this***

```
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)

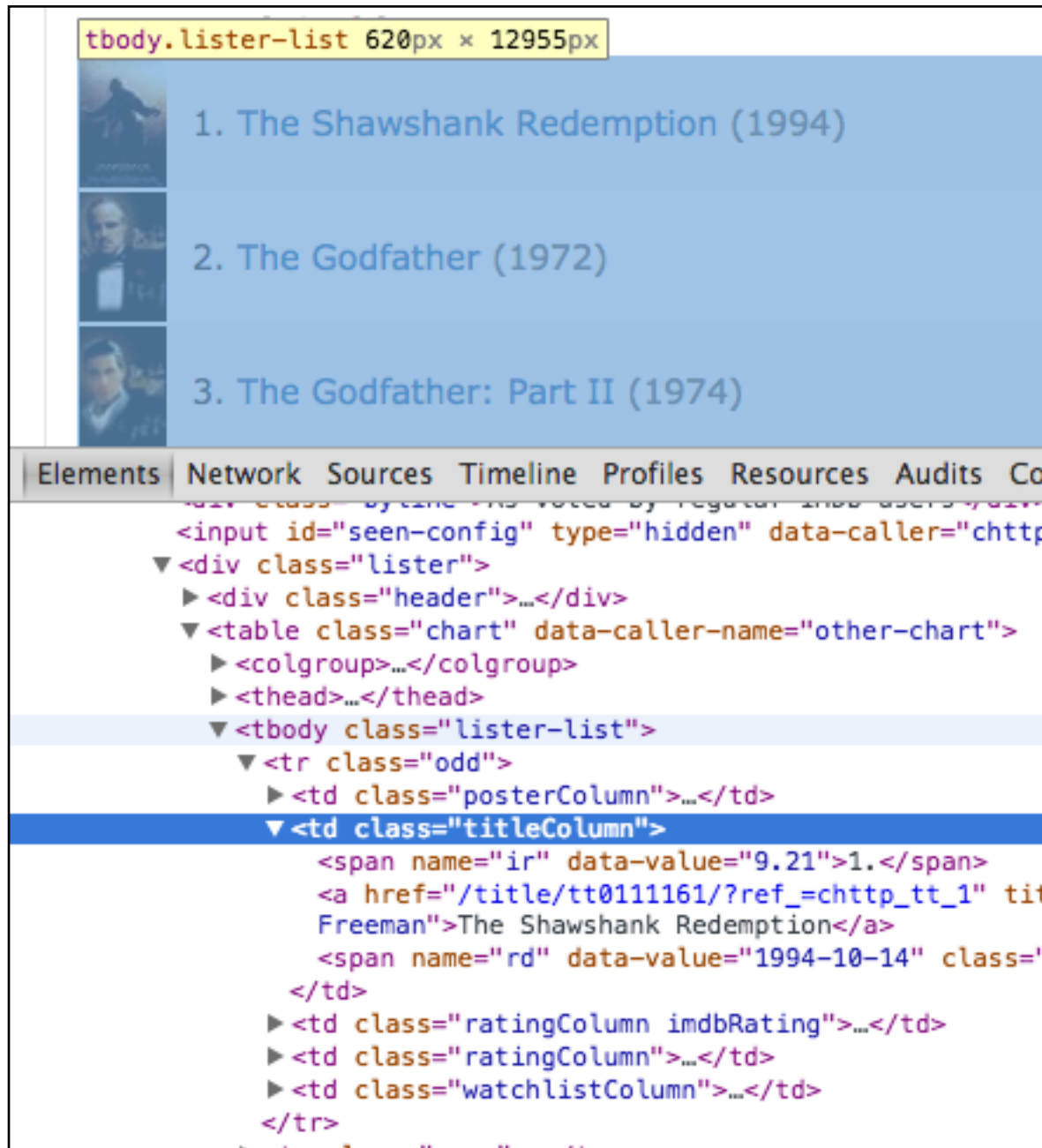
for row in soup.find('table', class_='cast_list').find_all('tr'):
    try:
        actor = clean_text(row.find(itemprop='name').text)
        role = clean_text(row.find(class_='character').text)

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

    except AttributeError:
        pass
```

[python]

# The Links



The screenshot shows a web browser's developer tools with the 'Elements' tab selected. The top part of the browser window displays a list of movies:

- 1. The Shawshank Redemption (1994)
- 2. The Godfather (1972)
- 3. The Godfather: Part II (1974)

The bottom part of the browser window shows the HTML structure of the page. The following code is visible:

```
<input id="seen-config" type="hidden" data-caller="http">  
<div class="lister">  
  <div class="header">...</div>  
  <table class="chart" data-caller-name="other-chart">  
    <colgroup>...</colgroup>  
    <thead>...</thead>  
    <tbody class="lister-list">  
      <tr class="odd">  
        <td class="posterColumn">...</td>  
        <td class="titleColumn">  
          <span name="ir" data-value="9.21">1.</span>  
          <a href="/title/tt0111161/?ref_=http_tt_1" tit  
            Freeman">The Shawshank Redemption</a>  
          <span name="rd" data-value="1994-10-14" class="'  
        </td>  
        <td class="ratingColumn imdbRating">...</td>  
        <td class="ratingColumn">...</td>  
        <td class="watchlistColumn">...</td>  
      </tr>  
    </tbody>  
  </table>  
</div>
```

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

remember:

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

[python]

# The Links

tbody.lister-list 620px x 129px

1. The Shawshank Redemption (1994)

2. The Godfather (1972)

3. The Godfather: Part II (1974)

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

[python]

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

Elements Network Sources Timeline Profiles Resources Audits Console

<input id="seen-config" type="hidden" data-caller="http://www.imdb.com/...>

<div class="lister">

<div class="header">...</div>

<table class="chart" data-caller-name="other-chart">

<colgroup>...</colgroup>

<thead>...</thead>

<tbody class="lister-list">

<tr class="odd">

<td class="posterColumn">...</td>

<td class="titleColumn">

<span name="ir" data-value="9.21">1.</span>

<a href="/title/tt0111161/?ref\_=http\_tt\_1" title="The Shawshank Redemption">

<span name="rd" data-value="1994-10-14" class="release-date">

</td>

<td class="ratingColumn imdbRating">...</td>

<td class="ratingColumn">...</td>

<td class="watchlistColumn">...</td>

</tr>

<tr class="even">

# The Links

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

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

The screenshot shows the 'Elements' panel of a web browser's developer tools. It displays a table with movie information. A dashed purple arrow points from the text on the right to a specific row in the HTML table, which is highlighted with a blue background. The HTML structure is as follows:

```
<div class="list">
  <div class="header">...</div>
  <table class="chart" data-caller="other-chart">
    <colgroup>...</colgroup>
    <thead>...</thead>
    <tbody class="list-list">
      <tr class="odd">
        <td class="posterColumn">...</td>
        <td class="titleColumn">
          <span name="ir" data-value="9.21">1.</span>
          <a href="/title/tt0111161/?ref=chttp_tt_1" tit
            Freeman">The Shawshank Redemption</a>
          <span name="rd" data-value="1994-10-14" class=
        </td>
        <td class="ratingColumn imdbRating">...</td>
        <td class="ratingColumn">...</td>
        <td class="watchlistColumn">...</td>
      </tr>
    </tbody>
  </table>
</div>
```

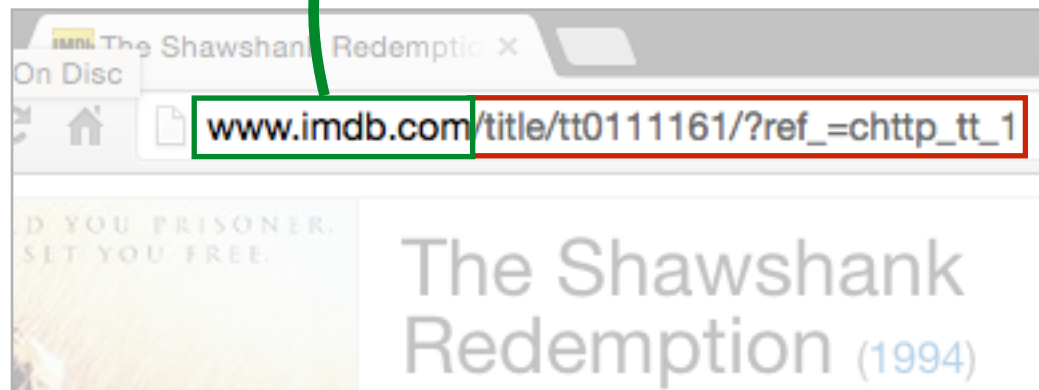


# The Links

**Build** a link from the href value

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

[python]



```
<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”**

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

[python]

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

# Workflow

requests ► BeautifulSoup ► .get('href') ► requests ...

```
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

# Straining 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 **finding** things in it.

```
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`

```
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, ...)

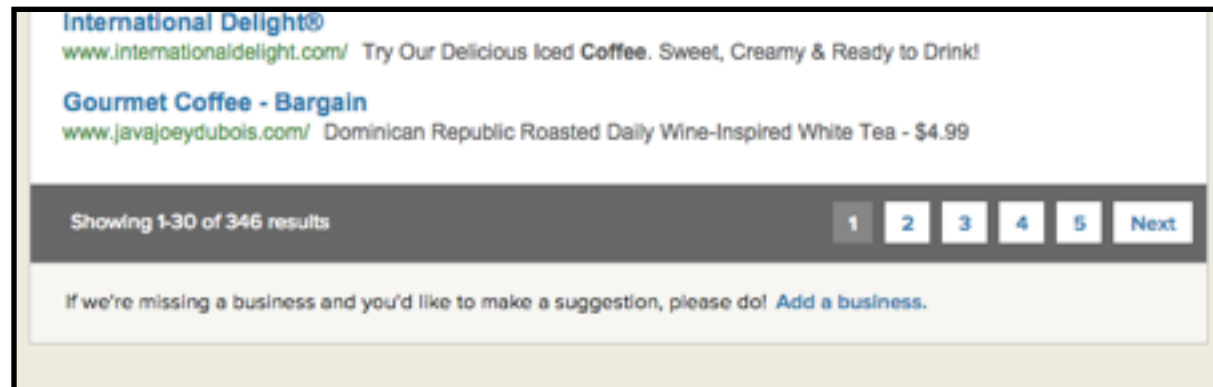
CON: Trickier to pipe/redirect output using command line



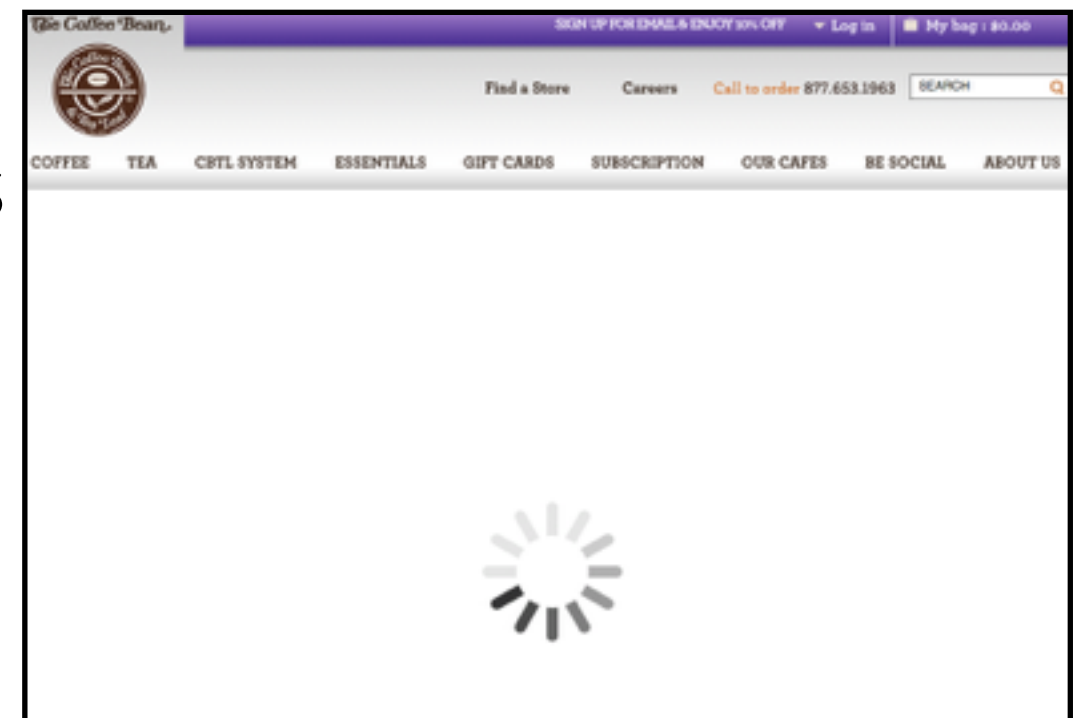
# Web Scraping Part2

a **Data Science Drop-in** Tutorial  
by **Houshmand Shirani-Mehr**  
([hshirani@stanford.edu](mailto: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](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.

49.



**22 Jump Street** (2014)

Add to Watchlist

\$192M

★★★★★☆☆☆☆ 7.2/10

After making their way through high school (twice), big changes are in store for officers Schmidt and Jenko when they go deep undercover at a local college.  
Dir: Phil Lord, Christopher Miller With: Channing Tatum, Jonah Hill, Ice Cube  
Action | Comedy | Crime112 mins. 

50.



**Cars 2** (2011)

Add to Watchlist

\$191M

★★★★★☆☆☆☆ 6.4/10

Star race car Lightning McQueen and his pal Mater head overseas to compete in the World Grand Prix race. But the road to the championship becomes rocky as Mater gets caught up in an intriguing adventure of his own: international espionage.  
Dir: John Lasseter, Brad Lewis With: Owen Wilson, Larry the Cable Guy, Michael Caine  
Animation | Adventure | Comedy | Family106 mins. 

1-50 of 705,050 titles.

Next »

[http://www.imdb.com/search/title?genres=comedy&sort=boxoffice\\_gross\\_us](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=51)

[http://www.imdb.com/search/title?genres=comedy&sort=boxoffice\\_gross\\_us&start=101](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](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.

```
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

Artificial intelligence: how clever do we want our machines to be?

29 Nov 2014 338

Elon Musk: artificial intelligence is our biggest existential threat

27 Oct 2014 673

Google buys two more UK artificial intelligence startups


23 Oct 2014 27

comments (133)

This discussion is closed for comments.

Order by Oldest Threads Collapsed


1 2 3

ubiktd Jon 3d ago

i'm not sure i trust a company that collaborated with the government to put back doors in it's systems so that they could spy on everyone.

2 ↑


Report

popcornmaster 3d ago

This debate between 'will destroy' and 'won't destroy' will be used as the opening montage to a feel good historical robot film, when Skynet inevitably rises up. Starting the robot equivalent of Tom Cruise.

2 ↑


Report

popcornmaster → popcornmaster 3d ago

Staring\* god damn it

1 ↑


Report

AlexAnder1 → popcornmaster 3d ago

Starring\*

1 ↑

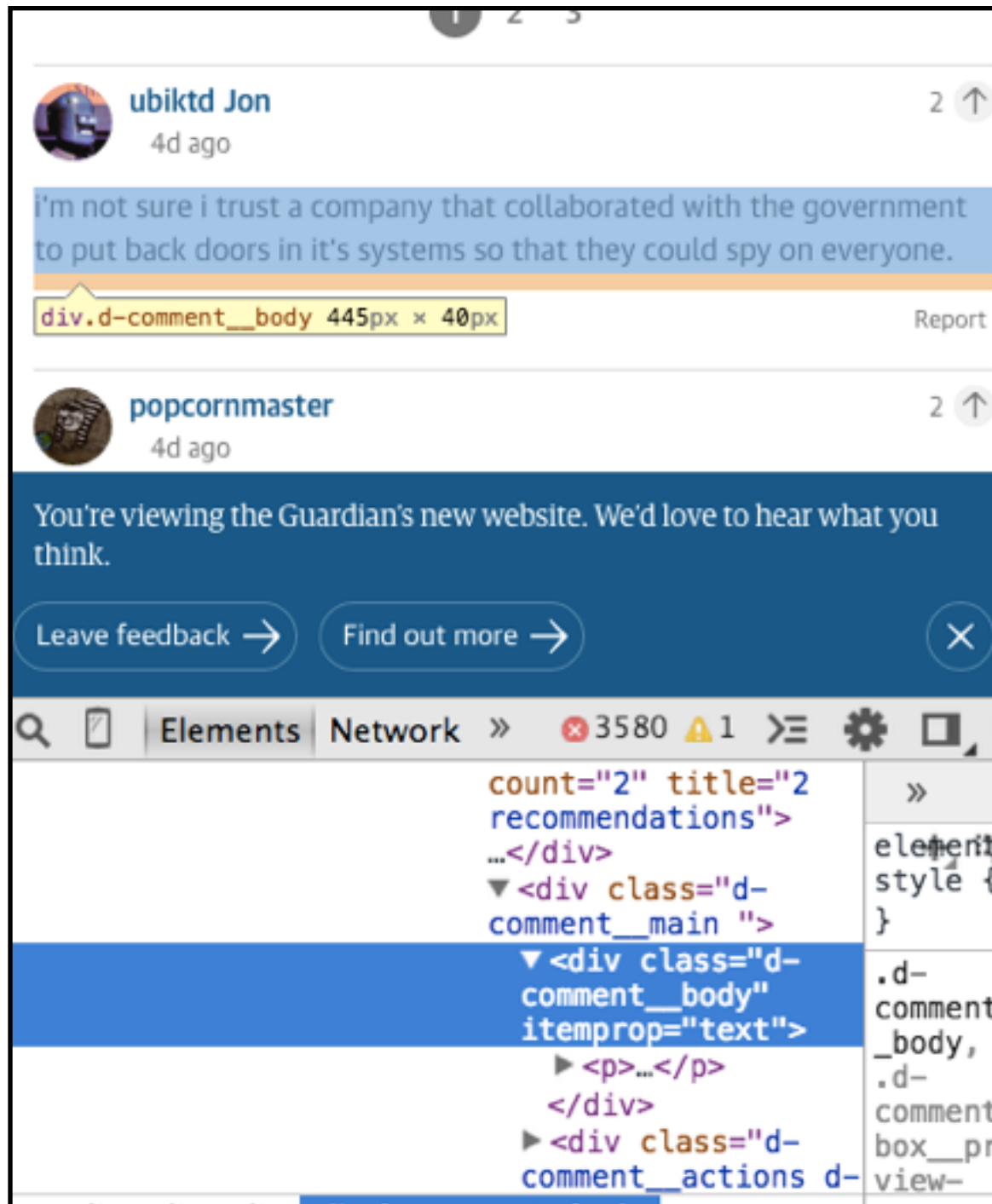
Report

You're viewing the Guardian's new website. We'd love to hear what you think.

Find out more →

Leave feedback →

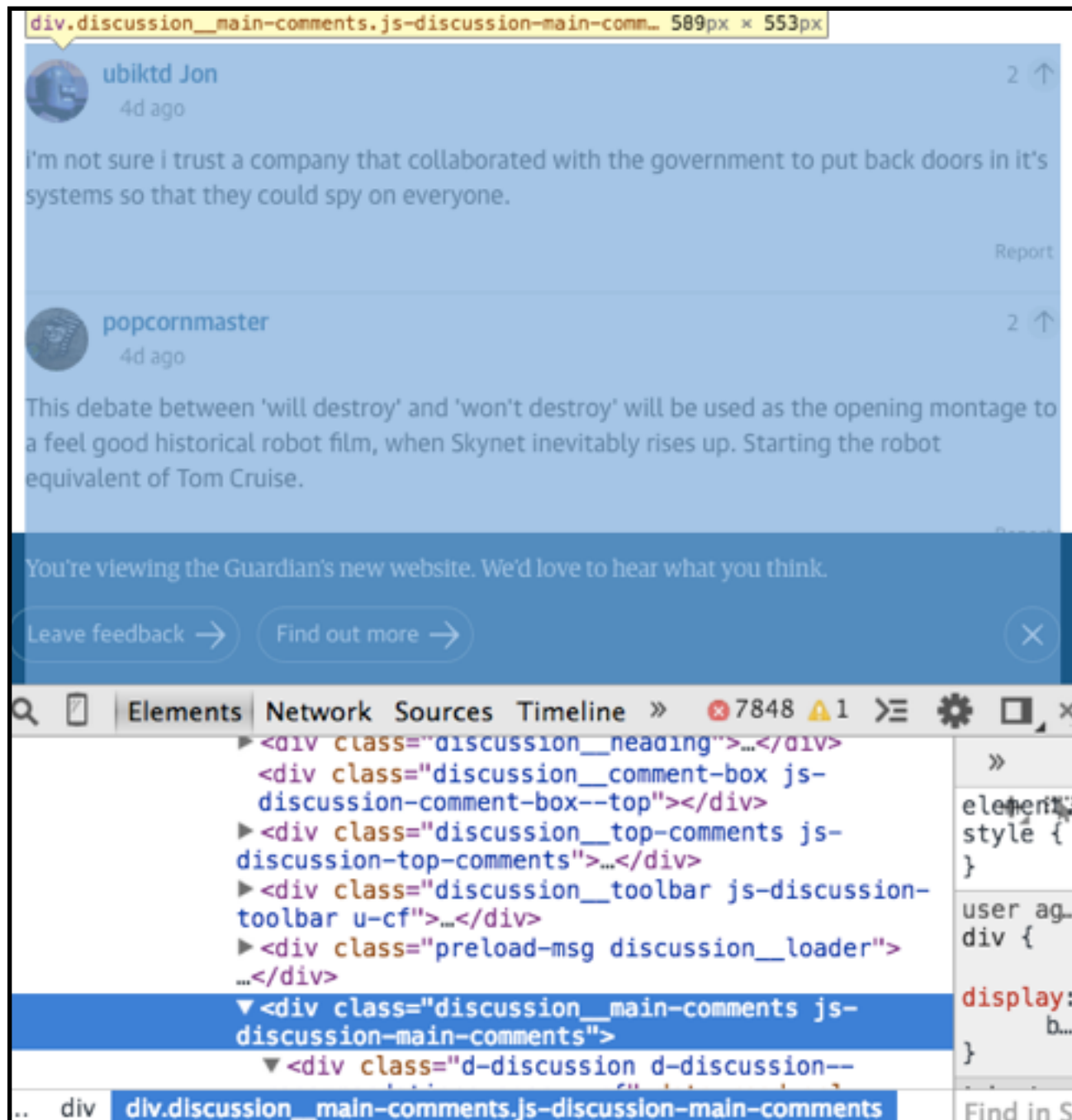
# Our Old Ways





# Compare

## Inspect Element



## Page Source

```
pagination"></div>
</div>
<div class="preload-msg
discussion_loader">Loading comments... <a
href=/discussion/p/45a9a class="accessible-
link">Trouble loading?</a><div class="is-
updating"></div></div>
<div class="discussion_main-comments js-
discussion-main-comments"></div>
<div class="discussion__comment-box
discussion__comment-box--bottom js-discussion-
comment-box--bottom"></div>
<button class="discussion__show-button button-
-show-more button button--large button--
primary js-discussion-show-button" data-link-
name="more-comments">
<i class="i i-plus-white"></i>
View more comments
</button>
```

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.

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

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

[python]



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.

```
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**