Web Scraping

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

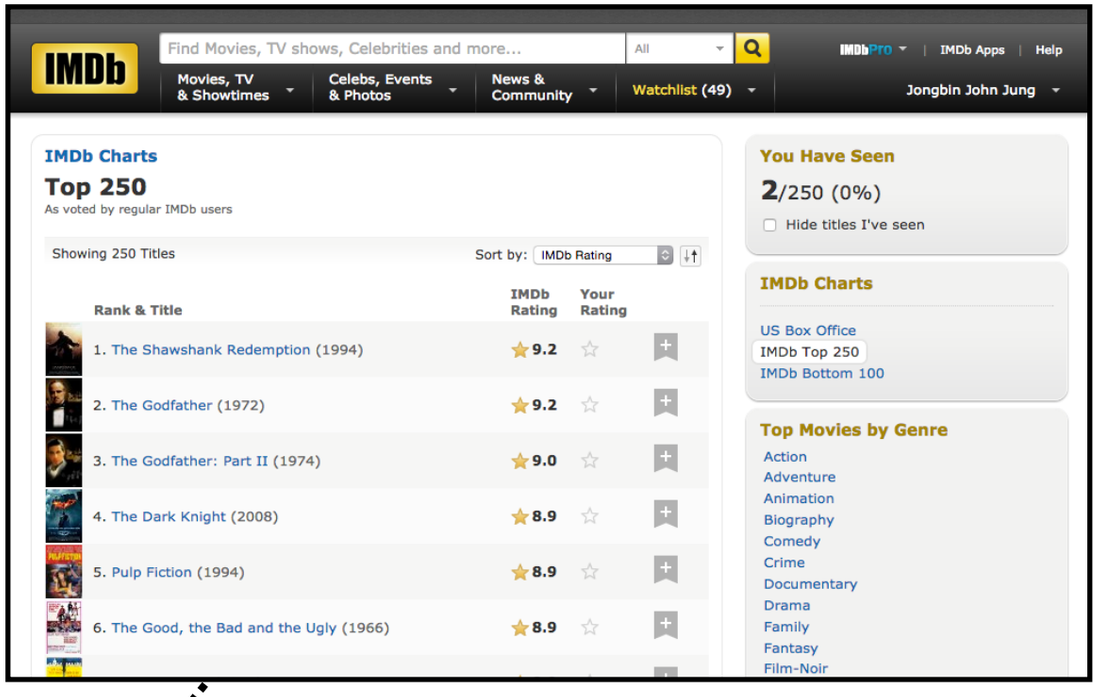
Before We Begin

- All material available at: https://github.com/5harad/css

- `python` packages required:

```
requests, beautifulsoup4[, selenium]
```

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**

```
            <div class="pending"></div>
            <div class="unseeable">NOT YET RELEASED</div>
            <div class="unseen"> </div>
            <div class="rating"></div>
            <div class="seen">Seen</div>
        </div>
    </div>
</span></td>
  <td class="watchlistColumn">  <div class="wlb_ribbon" data-tconst="tt0081398"></div>
</td>
</tr>
<tr class="even">
  <td class="posterColumn"><a href="/title/tt1049413/?ref_=chttp_tt_114"
><img src="http://ia.media-imdb.com/images/M/MV5BMTk3NDE2NzI4NF5BMl5BanBnXkFtZTgwNzE1MzEyMTE@._V1_SX34_CR0,0,34,50_AL_.jpg" width="34"
height="50" />
</a></td>
  <td class="titleColumn">
    <span name="ir" data-value="8.248">114.</span>
    <a href="/title/tt1049413/?ref_=chttp_tt_114"
title="Pete Docter (dir.), Edward Asner, Jordan Nagai" >Up</a>
    <span name="rd" data-value="2009-05-29" class="secondaryInfo">(2009)</span>
  </td>
  <td class="ratingColumn imdbRating">
    <strong name="nv" data-value="506660" title="8.2 based on 506,660 votes">8.2</strong>
  </td>
  <td class="ratingColumn"><span name="ur" data-value="0">    <div class="seen-widget seen-widget-tt1049413 pending" data-
titleid="tt1049413">
        <div class="boundary">
            <div class="popover">
<span class="delete"> </span><ol><li>1<li>2<li>3<li>4<li>5<li>6<li>7<li>8<li>9<li>10</ol>            </div>
        </div>
        <div class="inline">
            <div class="pending"></div>
            <div class="unseeable">NOT YET RELEASED</div>
            <div class="unseen"> </div>
            <div class="rating"></div>
            <div class="seen">Seen</div>
        </div>
    </div>
</span></td>
  <td class="watchlistColumn">  <div class="wlb_ribbon" data-tconst="tt1049413"></div>
</td>
</tr>
<tr class="odd">
  <td class="posterColumn"><a href="/title/tt1187043/?ref_=chttp_tt_115"
><img src="http://ia.media-imdb.com/images/M/MV5BMTMyOTg0ODQ1OF5BMl5BanBnXkFtZTcwNjc0NTMwNQ@@._V1_SY50_CR0,0,34,50_AL_.jpg" width="34"
height="50" />
```

Web Scraping?

Different ways to "scrape" the web:

1. **Standard** - load and scrape web page
2. The **Con** - fly like a browser, sting like a robot
3. The **Plumber** - look for the source of data

✳ The **Suit** - formal public API (all proper and stuff)

method of choice will likely be determined by how the target website is designed

Word of Caution

- Different people have different ideas regarding what's OK or not - read the fine print!
- It's possible to cause harm if you aren't careful
- Scraping is more art than science

Standard

load and scrape web page

the **GOAL**

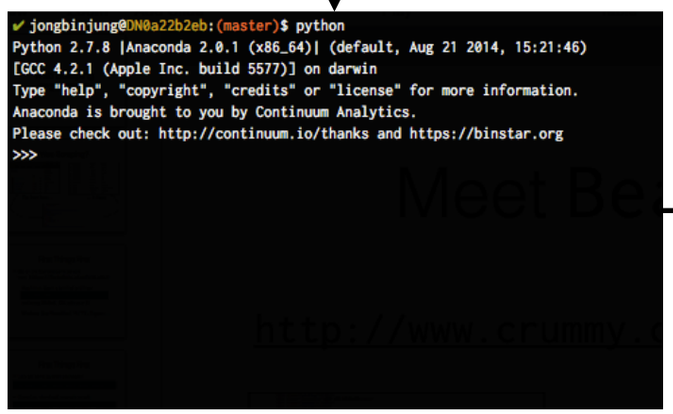
Collect the **cast overview** (actor and character played) for each of the **Top 10 movies** of **IMDb Charts' Top 250** (http://www.imdb.com/chart/top?ref_=nv_ch_250_4)

Meet BeautifulSoup

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

```
        <div class="pending"></div>
        <div class="unseeable">NOT YET RELEASED</div>
        <div class="unseen"> </div>
        <div class="rating"></div>
        <div class="seen">Seen</div>
      </div>
    </div>
</span></td>
  <td class="watchlistColumn">  <div class="wlb_ribbon" data-tconst="tt0081398"></div>
</td>
</tr>
<tr class="even">
  <td class="posterColumn"><a href="/title/tt1049413/?ref_=chttp_tt_114"
><img src="http://ia.media-imdb.com/images/M/MV5BMTkJNDE2Nz14N75BMl5BanBnXkFtZTcwNzE1MzByMTE@._V1_SX34_CR0,0,34,50_AL_.jpg" width="34"
height="50" />
</a></td>
  <td class="titleColumn">
    <span name="ir" data-value="8.248">114.</span>
    <a href="/title/tt1049413/?ref_=chttp_tt_114"
title="Pete Docter (dir.), Edward Asner, Jordan Nagai" >Up</a>
    <span name="rd" data-value="2009-05-29" class="secondaryInfo">(2009)</span>
  </td>
  <td class="ratingColumn imdbRating">
    <strong name="nv" data-value="506660" title="8.2 based on 506,660 votes">8.2</strong>
  </td>
  <td class="ratingColumn"><span name="ur" data-value="0">    <div class="seen-widget seen-widget-tt1049413 pending" data-
titleid="tt1049413">
        <div class="boundary">
            <div class="popover">
<span class="delete"> </span><ol><li>1<li>2<li>3<li>4<li>5<li>6<li>7<li>8<li>9<li>10</ol>           </div>
        </div>
        <div class="inline">
            <div class="pending"></div>
            <div class="unseeable">NOT YET RELEASED</div>
            <div class="unseen"> </div>
            <div class="rating"></div>
            <div class="seen">Seen</div>
        </div>
    </div>
</span></td>
  <td class="watchlistColumn">  <div class="wlb_ribbon" data-tconst="tt1049413"></div>
</td>
</tr>
<tr class="odd">
  <td class="posterColumn"><a href="/title/tt1187043/?ref_=chttp_tt_115"
><img src="http://ia.media-imdb.com/images/M/MV5BMTMyOTg0OQ1OF5BMl5BanBnXkFtZTcwNjc0NTMwMQ@@._V1_SY50_CR0,0,34,50_AL_.jpg" width="34"
height="50" />
```

from source (`html`)
to `python`



```
✔ jongbinjung@DN0a22b2eb:(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
>>>
```

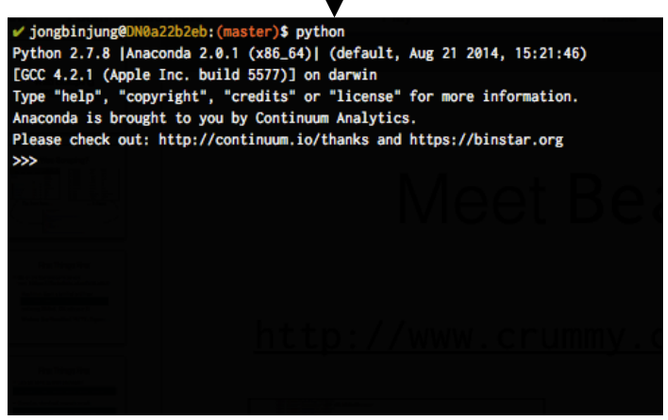
`BeautifulSoup`
goes **here**

	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

First, `get` the Web to `python`

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

from source (`html`)
to `python`




```python
import requests


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

[python]

Let's try this with the `BeautifulSoup` web page:

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

Use `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

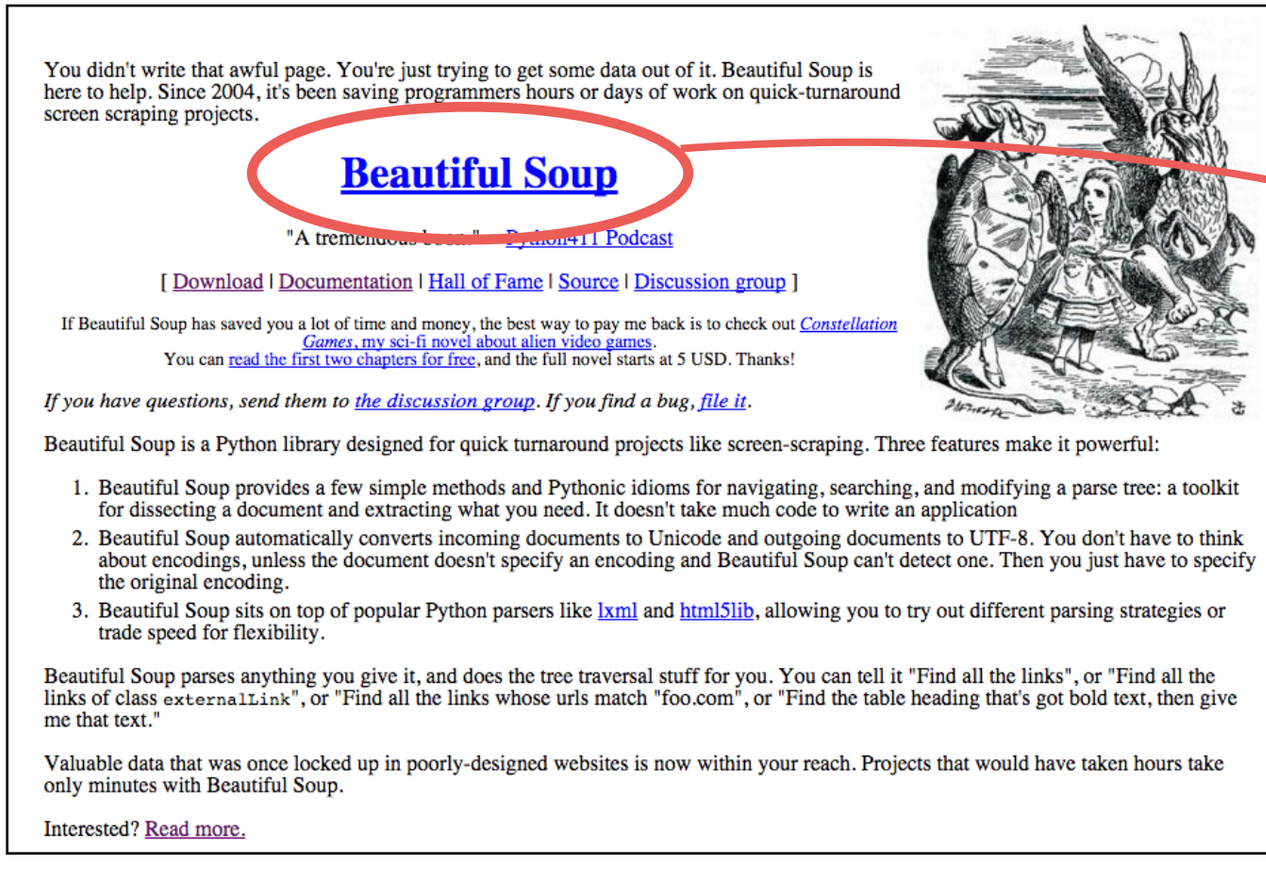
Use BeautifulSoup

```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 book." -- 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.

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)

```
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 `python` list
- We can `iterate` through a `list` with a `for` loop

Get more links and addresses

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

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

[python]

```output
'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

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

[python]

Goto IMDb.com

the **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)

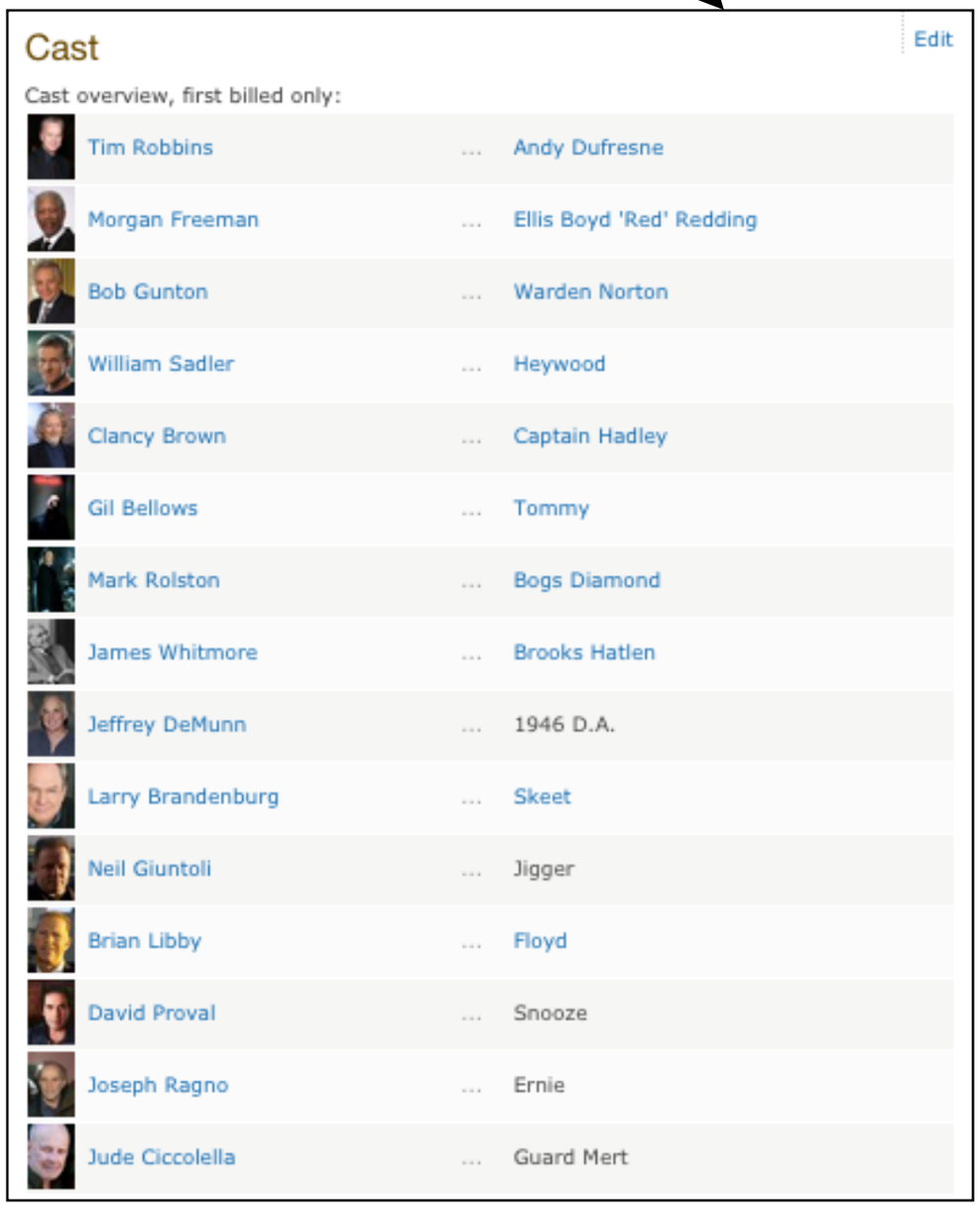
Let's start with

"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)

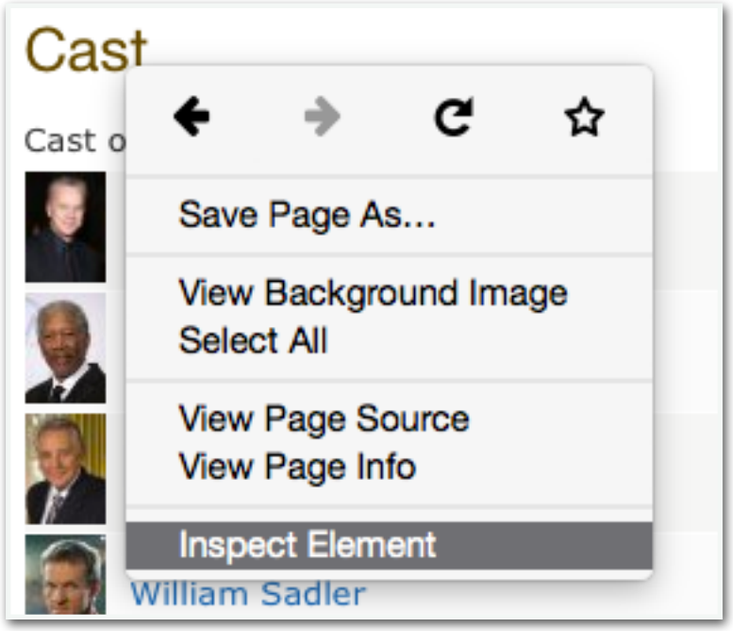
```python
from bs4 import BeautifulSoup
import requests


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`

(`<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"`

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
```

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 values

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

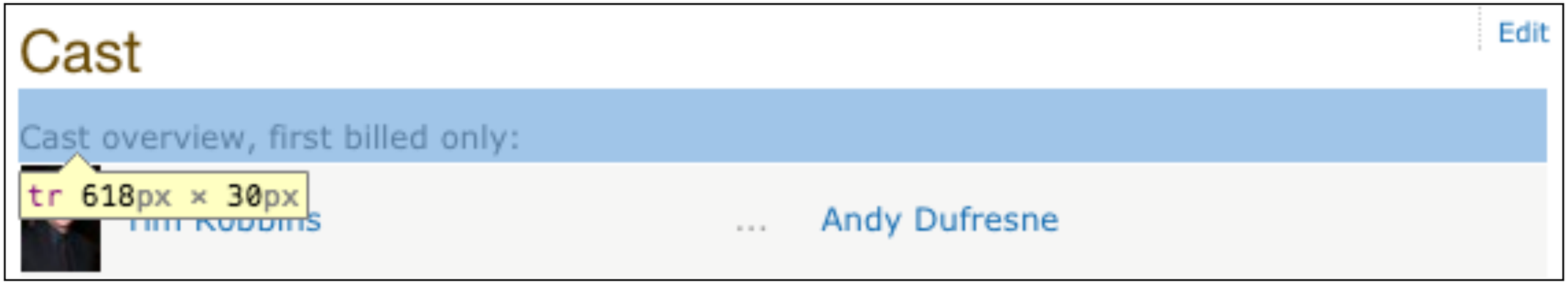
```python
for row in soup. ... find_all('tr')
    actor = row.find(itemprop='name').text
    role = row.find(class_='character').text
```

[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

if an `AttributeError` happens, it will do this

Manage with `try-except` Blocks

```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]

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 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 = clean_text(row.find(itemprop='name').text)
        role = clean_text(row.find(class_='character').text)

    except AttributeError:
        pass
```
[python]

Let's write a `function` that does this with any given `string`. You might find yourself needing this quite often.

```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, e.g.:

```
$ python script_name.py > movie_data.tsv
```
[command prompt]

So far ...

```python
from bs4 import BeautifulSoup
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]

Exercise 1

the **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***

```
from bs4 import BeautifulSoup
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
```

Be Nice

Don’t harass the servers

Rule-of-thumb(?): act like a human (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

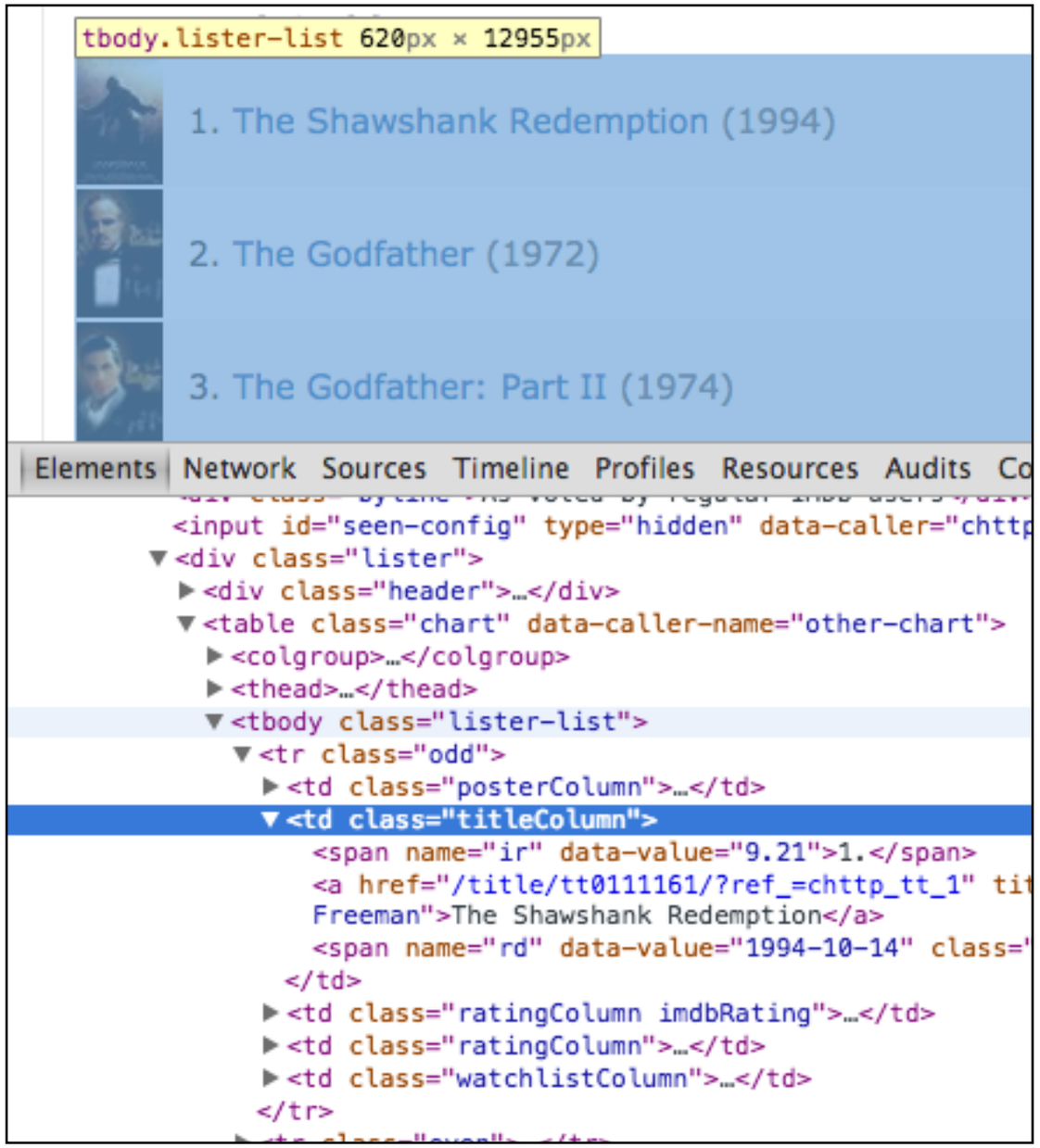
Workflow

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

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

```
from bs4 import BeautifulSoup
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
```

The Links

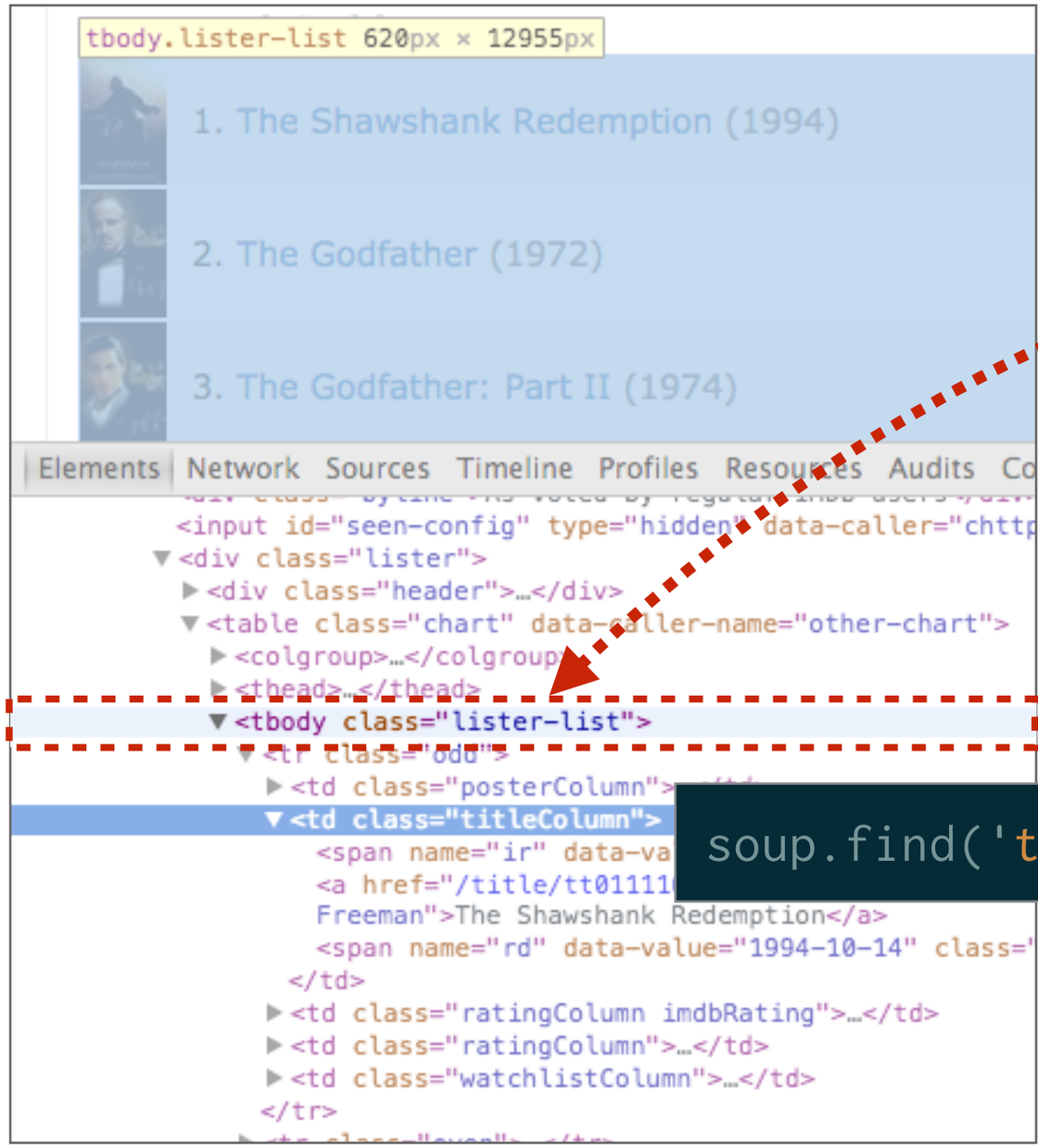


What do we need, and how should we get it?

remember:

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

The Links



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

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

The Links

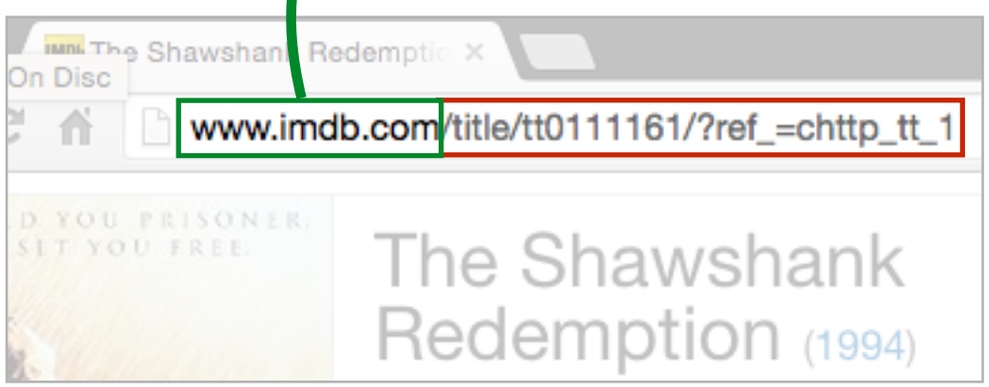
```python
for movie in soup. ... .find_all('td', class_='titleColumn')
```

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 ...

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])
```

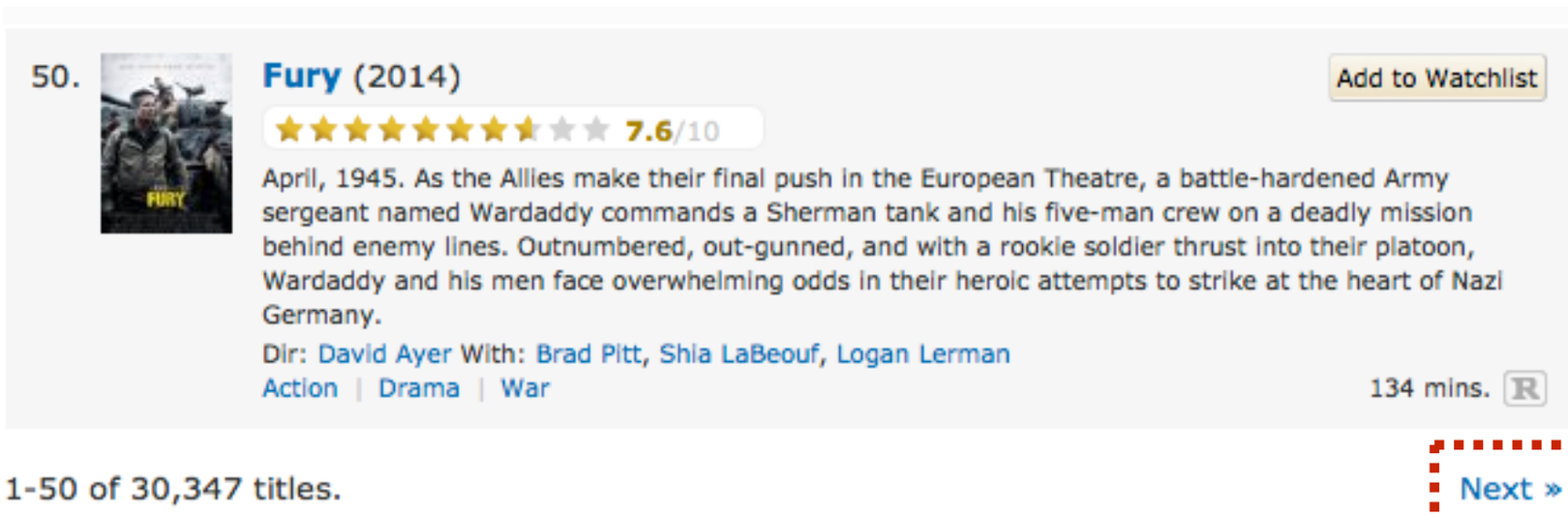
[python]

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

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

A note on URLs

- Back to IMDb
  Most Popular by Genre > Action > 'show more...'



Pages? How many? What? Why?

more Web Scraper's Rule of Thumbs

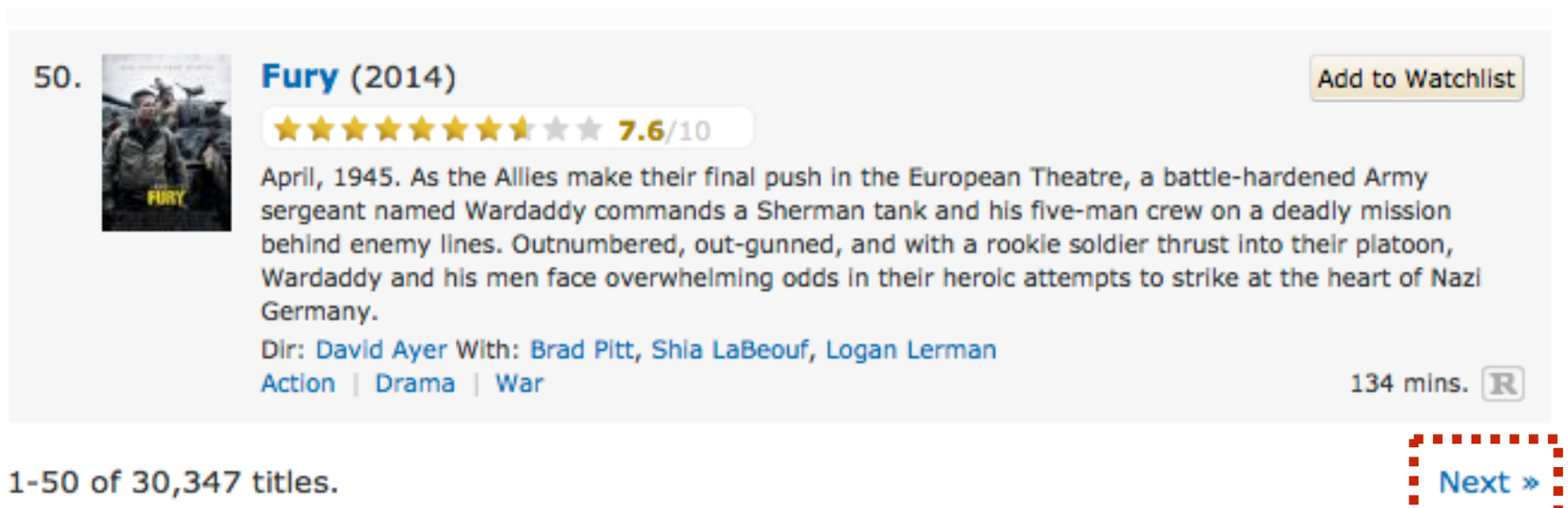
good web scrapers have many thumbs...

If it was organized by a *robot*,

it can be navigated by a *robot*.

A note on URLs

- Back to IMDb
  Most Popular by Genre > Action > 'show more...'



Pages? How many? What? Why?

take a look at the link:

```
http:// ... /title?genres=action&sort=moviemeter,asc&start=51&title_type=feature
```

A note on URLs

- Look closer ...

```
http:// ... /title?
        genres=action&
        sort=moviemeter,asc&
        start=51&
        title_type=feature
```

- May require trial-and-error investigation to identify how each parameter's value(s) affect the result

A note on URLs

- Use `requests` to build URLs with `parameters`

```python
import requests


base_url ='http://www.imdb.com/search/title'
params = {'genres':'action', 'sort':'moviemeter,asc'}


page = requests.get(base_url, params=params)
page.url
```
[python]

```
u'http://www.imdb.com/search/title?genres=action&sort=moviemeter,asc'
```
[output]

Exercise 2

collect the **title**, **release year**, and
grossing US Box Office Income ($)
for each of the **Top 250 comedy movies** sorted by
US Box Office Gross Income
(http://www.imdb.com/search/title?genres=comedy&sort=boxoffice_gross_us)

The **Con**

fly like a browser, sting like a robot

the **GOAL**

scrape your own webmail

get comments from today's headline of the guardian

* Disclaimer: Unless you have some experience developing web pages or at least some kind of experience with javascript, it's unlikely that you'll understand everything in this section at first glance. This section is only included to give you an idea of what's possible, and hopefully you can figure things out more clearly by carefully examining the examples (on github), should you require the methods of this section.

The last resort

- Some web pages are impossible to browse through with python (ok, not impossible, but complicated enough that we're unlikely to have the understanding of computer networks required to make it work)
  - proprietary databases that require login
  - secured (encrypted) connections
  - ...
- But you can still see stuff on your browser!
- What if a robot can just emulate the `copy-paste`?

`selenium`

- `selenium` is a webdriver - it let's you programmatically *drive* your browser
- originally built for automated website testing
- easily repurposed for web scraping
- not many distributions include `selenium` by default, install with pip (in command prompt)

```
$ pip install selenium                                  [command prompt]
```

selenium

- with `selenium`, you can create a browser object within your code, which will launch a real browser!

```python
from selenium import webdriver


browser = webdriver.Firefox()
```

- while `selenium` supports a variety of browsers, setting it up to work with anything other than Firefox has proven quite painful (at least to me)

- see the docs here: https://selenium-python.readthedocs.org/

Load a URL

- you can just type a URL into the browser, or
- load a page programmatically with

```
URL = 'http://webmail.stanford.edu'
browser.get(URL)
[python]
```

- we'll scrape our own Stanford Webmail, which requires two-factor authentication

find elements

- two approaches once a web page is loaded:

1. load the source with `.page_source`, then `bs4`

```
html = browser.page_source
soup = BeautifulSoup(html)
for sender in soup.find_all('div', class_='class_name'):
    print sender.text
```
[python]

2. use selenium's `.find_elements_by_*`

```
for element in browser.find_elements_by_class_name('class_name'):
    print element.text
```
[python]

scrolling

- you might need to scroll certain elements
- selenium can send javascript to the browser
- javascript is not within scope of this workshop, but let's use just a snippet ...

```python
panes = browser.find_elements_by_class_name('scrollContainer')
for pane in panes:
    if '_xlv_e1' in pane.get_attribute('class'):
        browser.execute_script('arguments[0].scrollTop =
                                'arguments[0].scrollHeight', pane)
```

[python]

waiting

- once logged in, the rest can be automated
- even waiting for pages to load
- `selenium` can wait for certain elements to load
- this requires a few extra modules

```python
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
from selenium.common.exceptions import TimeoutException
```

[python]

waiting

- the standard syntax look like:

```python
containers = WebDriverWait(browser, MAX_WAIT).until(
    EC.presence_of_all_elements_located(
        (By.CLASS_NAME, "fc-container__inner")))
```

[python]

- waits until elements with class 'fc-container__inner' are found, and returns the list of elements
- throws `TimeoutException` if `MAX_WAIT` seconds pass without finding at least one element that matches

The **Plumber**

look for the source of data

the **GOAL**

Collect used car price data

* Some critical contents on this section have been masked from the slide deck to protect the target website(s) from potential harm. Source code and related materials can be found on the github repo

“Why is the source code that I `get` from `requests` different from what I see in the browser's Developer Mode?”

– Everyone who has tried scraping dynamically loaded pages

This will only work *sometimes*

- The [Network] tab in your browser's Developer mode will let you investigate from *where* and *how* the web page was loaded

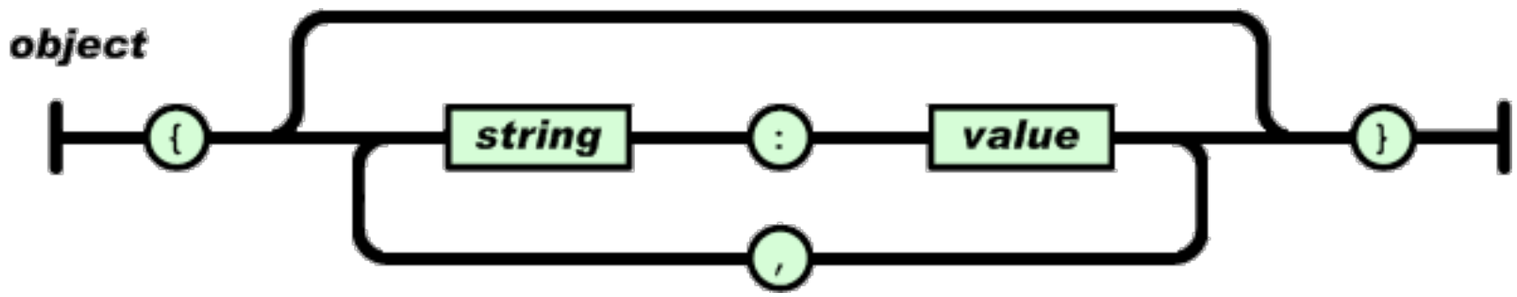
- This may (or may not) give you insight on a better approach to retrieving data that is presented to you

- Sometimes, you might notice that the data on the page is loaded from a `json` file

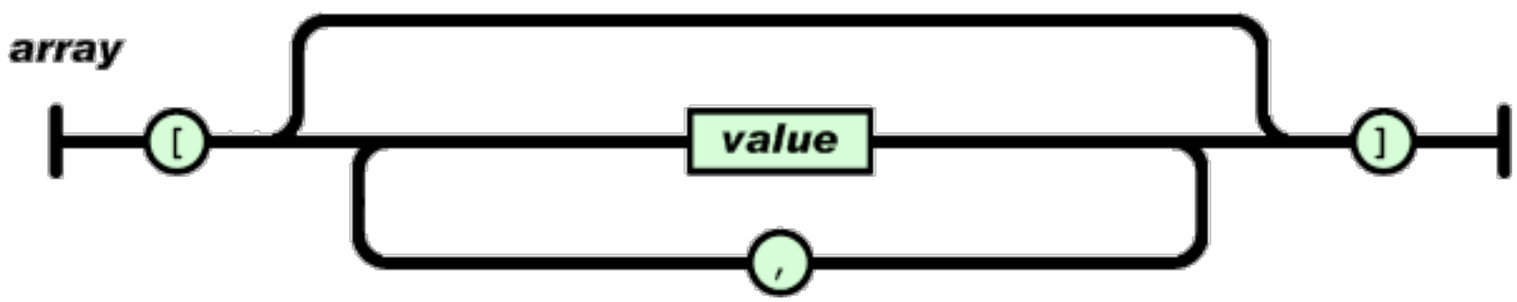
Introduction to JSON

- a text-base data representation format that can be used to represent complex data structures
- often the data format of choice between server-client communication due to **light-weight** (minimal symbols) and **interchangeable** (text-based) **standardized formatting** (cf. csv)

Introduction to JSON



think: `python dict`



think: `python list`

JSON with requests

- the `requests.Response` object (the object returned by `requests.get`) has a `json()` method, which returns the `json`-encoded content of a response, if any

```python
# assuming the base_url returns json content
data = requests.get(base_url, params=params).json()
```

- once the `json`-encoded content is loaded, you can treat it as a regular `python` dictionary

Exercise 3

Choose a make/model of interest (e.g., Honda Accord)

Collect the **VIN**, **year**, **trim**, **mileage**, and **price**
for all used cars of selected make/model,
available within 50 miles of your zip-code.

(you should only need to make a single `requests`)

the **Suit**

no detailed slides
but example codes with comments on git

Using the twitter api with `tweepy`

see: `web-scraping/examples/collect_tweets.py`

the **End**