GENERAL ASSEMBLY

APIS AND SCRAPING

APIS AND SCRAPING

- API Review
- Scraping Review

THE BASIC PROCEDURE

- Using the API docs, get a URL that requests the desired data.
- Try the URL in your browser. Does it return the desired data?
- Request the data in Python using the requests module. Convert the response from JSON to Python data structures.
- Use Python indexing to access the desired data.

WORDNIK EXAMPLE

- Using the API docs, get a URL that requests the desired data.

  http://developer.wordnik.com/docs.html

- Try the URL in your browser. Does it return the desired data?

http://api.wordnik.com/v4/word.json/python/definitions?
limit=200&
includeRelated=true&
useCanonical=false&
includeTags=false&
api_key=a2a73e7b926c924fad7001ca3111acd55

WORDNIK EXAMPLE

- Using the API docs, get a URL that requests the desired data.

  http://developer.wordnik.com/docs.html
- Try the URL in your browser. Does it return the desired data?
- Request the data in Python using the requests module. Convert the response from JSON to Python data structures.

```
import requests
r = requests.get(URL)
word_definition = r.json()
```

WORDNIK EXAMPLE

```
[{'attributionText': 'from The American Heritage® Dictionary of the English
Language, 4th Edition’,
'partOfSpeech': 'noun’,
'text': 'Any of various nonvenomous snakes of the family Pythonidae, found chiefly
in Asia, Africa, and Australia, that coil around and suffocate their prey. Pythons
often attain lengths of 6 meters (20 feet) or more.’,
'word': 'python’,
...
}]
```

a list [] of dictionaries {}

WORDNIK EXAMPLE


```
[{'attributionText': 'from The American Heritage® Dictionary of the English
Language, 4th Edition’,
'partOfSpeech': 'noun’,
'text': 'Any of various nonvenomous snakes of the family Pythonidae, found chiefly
in Asia, Africa, and Australia, that coil around and suffocate their prey. Pythons
often attain lengths of 6 meters (20 feet) or more.’,
'word': 'python’,
...
}]
```


word_definition[0]['text']

a list [] of dictionaries {}

THE BASIC PROCEDURE

- Using the Web Inspector, identify ids or classes that uniquely identify the data to scrape

- Use requests to get the HTML into Python
- Use BeautifulSoup to convert the HTML string into Python data structures
- Get the ids/classes using **select**. Get the text using the property **text**. To get all of the text in all descendents, use **get_text()**.

THE BASIC PROCEDURE

- Using the Web Inspector, identify ids or classes that uniquely identify the data to scrape

  http://www.nasdaq.com/symbol/yhoo/after-hours

  Stock price represented by id = ?

THE BASIC PROCEDURE

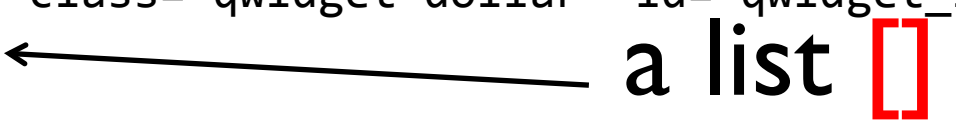
- Using the Web Inspector, identify ids or classes that uniquely identify the data to scrape
- Use requests to get the HTML into Python
- Use BeautifulSoup to convert the HTML string into Python data structures

```
import requests
from bs4 import BeautifulSoup

r = requests.get('http://www.nasdaq.com/symbol/yhoo/after-hours’)
soup = BeautifulSoup(r.content)
```

THE BASIC PROCEDURE

- Using the Web Inspector, identify ids or classes that uniquely identify the data to scrape
- Use requests to get the HTML into Python
- Use BeautifulSoup to convert the HTML string into Python data structures
- Get the ids/classes using **select**. Get the text using the property **text**. To get all of the text in all descendents, use **get_text()**.

```
In [15]: soup.select('#qwidget_lastsale')
Out[15]: [<div class="qwidget-dollar" id="qwidget_lastsale">
$40.47</div>]
```



THE BASIC PROCEDURE

- Using the Web Inspector, identify ids or classes that uniquely identify the data to scrape
- Use requests to get the HTML into Python
- Use BeautifulSoup to convert the HTML string into Python data structures
- Get the ids/classes using **select**. Get the text using the property **text**. To get all of the text in all descendents, use **get_text()**.

```
In [15]: soup.select('#qwidget_lastsale')[0].text
Out[15]: '$40.47'
```

OTHER USEFUL BEAUTIFULSOUP METHODS

find_all(id="main-news-story")

find_all(class_="news-story")

- "class_" because "class" is a Python reserved word

get_text()

- concatenates all text nodes (i.e. strips HTML tags)

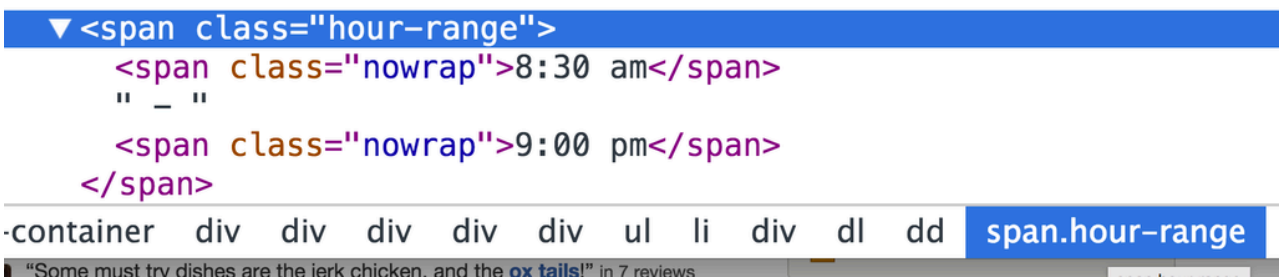
The following examples use the following Yelp page:

Will's Jamaican Cuisine

http://www.yelp.com/biz/wills-jamaican-cuisine-inglewood?hrid=lFvP6CD1LgMYInElXwpsEQ

Step 1: Right-click on the element to scrape. Choose "Inspect Element".

If you are already in the Inspector, you can click on the magnifying glass in the upper left then directly select part of the page.



Step 2: In the Inspector, the code referring to the selected element is highlighted. In the lower right, a CSS selector uniquely identifying that code is in blue!

bs.select('address') # selects all address tags

```
680                    <address>
681          630 N La Brea Ave, Inglewood, CA 90302
682      </address>
```

selects all 'h1' tags with class 'biz-page-title'

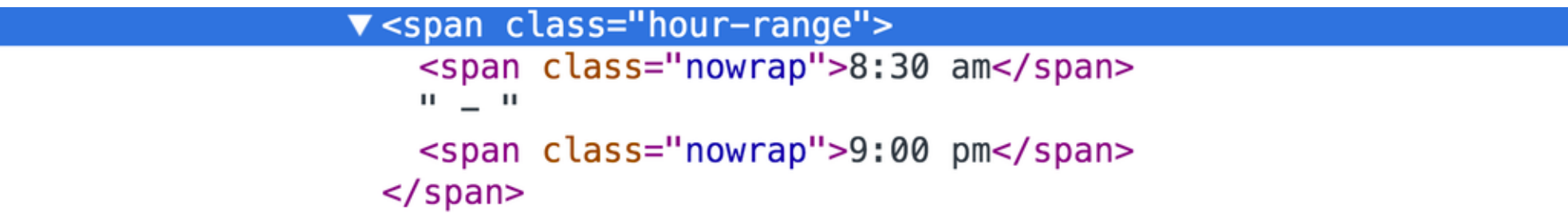
bs.select('h1.biz-page-title')

```
495      <h1 class="biz-page-title embossed-text-white" itemprop="name">
496          Will's Jamaican Cuisine
497      </h1>
```

^^^ An 'h1' tag with classes 'biz-page-title' and 'embossed-text-white'

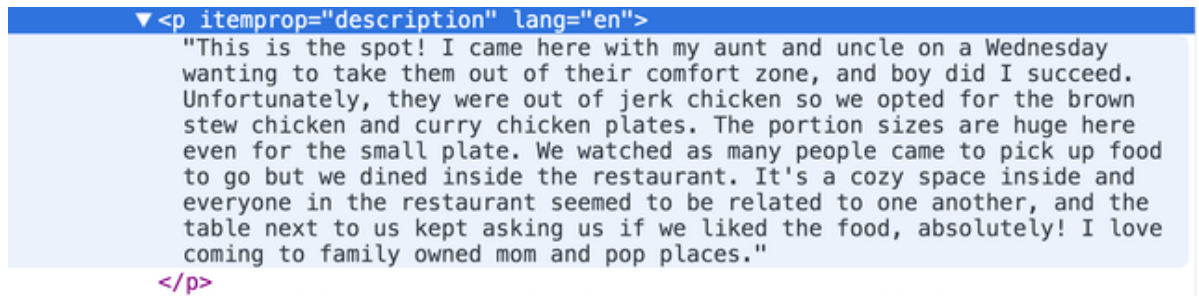
```
# Here, the text portions (i.e. text excluding the tags) are conveniently the exact hour range we want!

# So, we can use the get_text() function to strip away the tags.
```

hour_range = bs.select('span.hour-range')[0].get_text()



```
# selects all ‘p’ tags with attribute ‘itemprop’ that has
      value “description”
```

bs.select(‘p[itemprop=“description”]’)



NOTE: ‘class’ is also an attribute! So, alternatively to the . notation for classes, you could more generally write for the last slide’s example:

bs.select(‘h1[class=“biz-page-title”]’)

```
# Here, the rating value is stored as a tag attribute.
# So, for each tag found, we will use the 'attrs' dictionary
to grab these ratings.

rating_tags = bs.select('meta[itemprop="ratingValue"]')
ratings = [float(tag.attrs['content']) for tag in rating_tags]
```

```
1855                <meta itemprop="ratingValue" content="3.0">
1856        </div>
```

CSS SELECTORS FOR SCRAPING – MORE BUSINESS INFO


```
# This is more tricky. Let's try to put it in a dictionary,
since these are key-value pairs. Let's get a list of the <dl>
elements, then for each <dl> we'll grab the <dt> and <dd>!
```

More business info

Takes Reservations **No**
Delivery **No**
Take-out **Yes**
Accepts Credit Cards **Yes**
Good For **Lunch**
Parking **Private Lot**
Bike Parking **No**
Good for Kids **Yes**
Good for Groups **Yes**
Attire **Casual**
Ambience **Casual**
Noise Level **Average**

```
# '>' means direct descendent. So, all 'dl' tags that are
direct descendents of a 'div' with class 'short-def-list'


biz_info_tags = bs.select('div.short-def-list > dl')


biz_info = {}
for tag in biz_info_tags:
        key = tag.select('dt')[0].text.strip()
        value = tag.select('dd')[0].text.strip()
        biz_info[key] = value
```