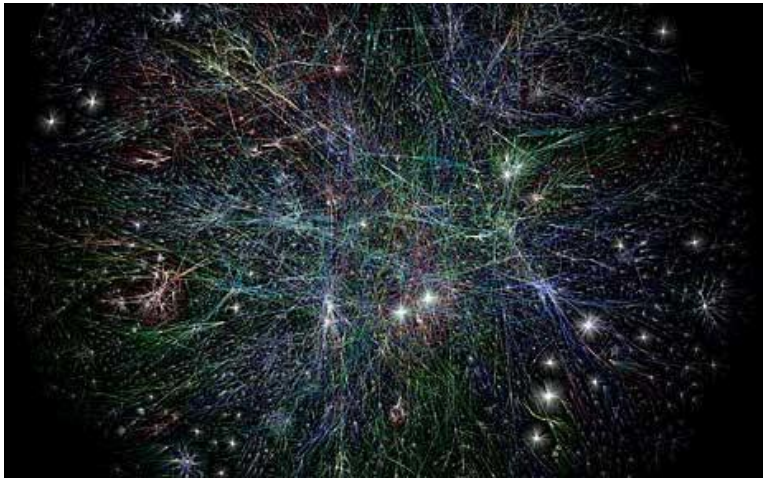
Becoming dangerous with web extraction

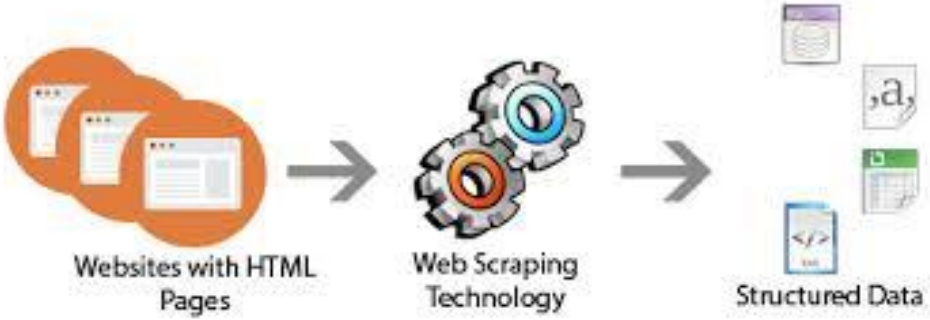
By: Lucas Ou, ICSSC

Introduction

- The “web” is a system of linked documents/pages. (html, markdown, etc)
- These pages are usually powered via remote servers accessed from the internet via the HTTP protocol. “http://”



What is it?

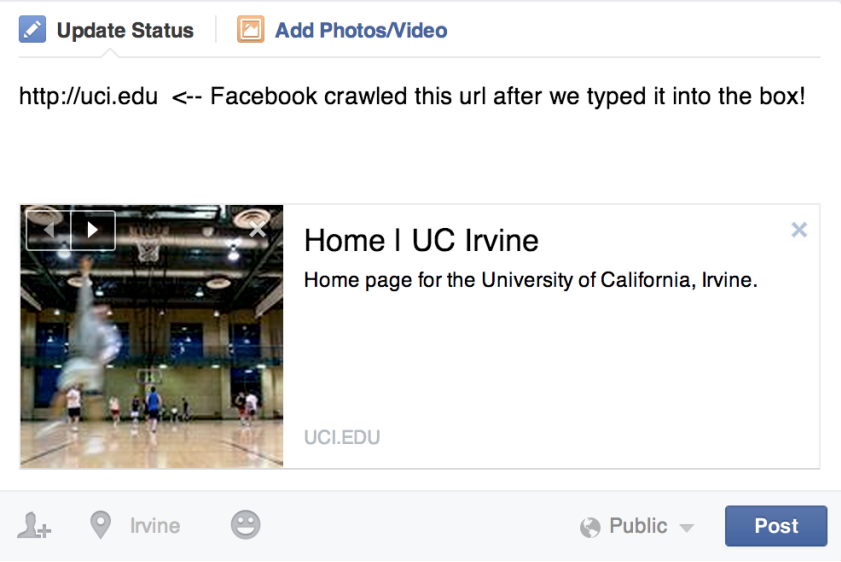


- People usually access the web through browsers or mobile apps.
- But, this process can be automated and mimicked with bots written in your programming language of choice!

^ the definition of web scraping.

Famous examples

- Reddit thumbnails
- Facebook blurbs
- Search Engines (Google, Bing)
- Marketplaces (Zappos, Amazon)

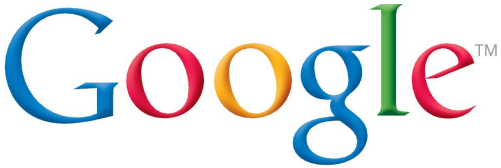


Picking the right tools

- Python 2.7 :)
- *Many* companies use python for extraction related tasks *it's not a coincidence*

Why Python?

- Jobs are IO bound! Not CPU bound.
- Readability >
- Many existing libraries to aid us



Let’s get started

- Simple first task: download the HTML from a website as a *String.* The ‘html’ var holds our str.

```
>>>
>>> import urllib2
>>>
>>> request = urllib2.Request('http://yahoo.com')
>>>
>>> response = urllib2.urlopen(request)
>>>
>>> print response.code
200
>>>
>>> html = response.read()
>>>
>>>
```

Let’s get started

You have the web page's contents as a String!

Try out common String operations:

- printing the html `>>> print html`
- writing it to a file `>>> f = open(‘~/myfile.txt’, ’w’); f.write(html); f.close();`
- printing out a substring of the html `>>> print html[0:100] #  first 100 chars`
- printing out the length of the html `>>> print len(html)`

Reflect on what we just did

HTTP? HTML? Requests & Responses?

- HTTP is the network protocol of the Web. It is both simple and powerful.

- HTTP stands for **Hypertext Transfer Protocol**. It's the network protocol used to deliver virtually all files and other data (collectively called *resources*) on the World Wide Web, whether they're HTML files, image files, query results, or anything else.

- Usually, HTTP takes place through TCP/IP sockets. ← CS132 anyone?

- A browser is an *HTTP client* because it sends requests to an *HTTP server* (Web server), which then sends responses back to the client.

- There is much, much more that won’t be covered in this workshop.

Reflect on what we just did

- The “urllib2” library is simply an *abstraction* of HTTP!
- Instead of worrying about writing a formal HTTP request GET command and firing it towards a remote server, we just write a few lines!

Advanced example

- Print out all the news titles from www.uci.edu

```
1 import urllib2
2 from BeautifulSoup import BeautifulSoup
3
4
5 # Download the HTML
6 request = urllib2.Request('http://www.uci.edu')
7 response = urllib2.urlopen(request)
8
9 print '\r\n\r\n'
10
11 # Verify that everything went ok.
12 # Error codes: 200 == good, 404, 500 == bad
13 print 'The error code is:', response.code
14 print '\r\n\r\n'
15 html = response.read()
16
17 # Parse the HTML into a dom object via our BS library.
18 dom = BeautifulSoup(html)
19
20 # Extract out the <div> tag containing our news.
21 news_tag = dom.find('div', {'id': 'news'})
22
23 # See what the extracted HTML looks like.
24 print 'The extracted news div HTML looks like:'
25 print '===================================='
26 print news_tag
27 print '\r\n\r\n'
28
29 # Further extract out a list of the actual news titles.
30 news_li_tags = news_tag.findAll('li')
31 titles = [tag.text for tag in news_li_tags]
32
33 print 'The top news stories on www.uci.edu are currently:'
34 print '===================================='
35 for title in titles:
36     print title
```

Advanced Example

- Do the same as the last slide but now store the links to those titles! Put it into an array?

- Good idea to read docs:

http://www.crummy.com/software/BeautifulSoup/bs3/documentation.html

Answer Key on next Slide!!

Advanced Example

- Do the same as the last slide but now store the links to those titles! Put it into an array?

```
news_li_tags = news_tag.findAll('li')
titles = [tag.text for tag in news_li_tags]
links = [tag.a['href'] for tag in news_li_tags]
```

- Good idea to read docs:

http://www.crummy.com/software/BeautifulSoup/bs3/documentation.html

More reflection

- How exactly did parsing the HTML happen?

Our HTML is just a String. We converted it into a data structure (hash tables and lists) which is easily searchable and manipulable via a programming language.

“<html>Lucas is here.</html>” → {‘html’: [‘Lucas is.’, {‘b’: ‘here’}]}.

- What is a “DOM”

The Document Object Model (DOM) is a programming interface for valid *HTML* documents. It pretty much churns your HTML string into a manipulatable data structure.

Cool insight: The links you found on that HTML page lead to more HTML pages. You can go to all of those pages and repeat the above step! Download the entire internet!?

More reflection

- The “BeautifulSoup” library is just an *abstraction* for HTML parsing!
- Instead of writing regex’s to capture content within <tags> or whatever, we write simple, clean, and reliable python code.

Super advanced example

- Reddit thumbnail algorithm
- Please visit the URL: https://github.com/codelucas/uci-web-extraction-workshop/blob/master/advanced.py
- Let’s trace the gorgeous code together and then i’ll demo it on my machine!

With great power comes...

- Be responsible when you are crawling websites.
- /robots.txt
- Time intervals

You have more power than you may realize...

```
while True:
    requests.get('http://crash.this.website.com')
```

^ Maybe also spoof your useragent, IP address, etc, be sneaky. But don't actually do this, it's rude.

Advanced topics

- Scheduling your crawlers & bots
- Unicode, Unicode, Unicode
- Multithreading & Async IO
- HTML parsing methods: XPath vs Regex

Thanks for coming!

Now proceed and build great things: